

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО  
ОБРАЗОВАНИЯ  
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

**Факультет безопасности информационных технологий**

**Направление подготовки: Информационная безопасность.**

**Образовательная программа: Технологии защиты информации.**

**Дисциплина:**

«Информационная безопасность баз данных»

**КУРСОВАЯ РАБОТА**

**на тему** «Разработка многопользовательского веб-интерфейса для управления базой данных информационной системы компании, предоставляющей услуги коммуникаций и видеонаблюдения»

**Выполнил студент(ы):**

Группа N3347

Климов Д. И / \_\_\_\_\_

ФИО

Подпись

**Проверил:**

\_\_\_\_\_ / \_\_\_\_\_

ФИО

Подпись

\_\_\_\_\_  
Отметка о выполнении (один из вариантов:  
отлично, хорошо, удовлетворительно)

\_\_\_\_\_  
Дата

Санкт-Петербург

2024г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО  
ОБРАЗОВАНИЯ  
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

**ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ (РАБОТУ)**

Студент	Климов Даниил Иванович
	( Фамилия, И., О. )
Факультет	Безопасности информационных технологий
Группа	N3347
Направление (специальность)	Информационная безопасность
Руководитель	Таранов Сергей Владимирович, ординарный доцент, доцент, кандидат технических наук
	( Фамилия, И.О., должность, ученое звание, степень )
Дисциплина	Информационная безопасность баз данных

Наименование темы	Разработка многопользовательского веб-интерфейса для управления базой данных информационной системы компании, предоставляющей услуги коммуникаций и видеонаблюдения
-------------------	---

Задание	Для выбранной информационной системы требуется выполнить инфологическое моделирование баз данных по методу «сущность-связь», реализовать сервис для взаимодействия с БД, обеспечить резервирование БД и восстановление по контрольным точкам
---------	--

Краткие методические указания	Для реализации БД была выбрана СУБД PostgreSQL, для реализации веб-интерфейса был выбран Python фреймворк FastAPI и React фреймворк Next.js
-------------------------------	---

Содержание пояснительной записки	В отчете будут продемонстрированы следующие шаги выполнения работы: инфологическое моделирование баз данных по методу «сущность-связь», реализация БД в рамках СУБД, обеспечение защиты БД, реализация сервиса для взаимодействия с БД, обеспечение резервирования БД и восстановления по контрольным точкам
----------------------------------	--

Руководитель	Подпись, дата
Студент	Подпись, дата

## ГРАФИК ВЫПОЛНЕНИЯ КУРСОВОГО ПРОЕКТА (РАБОТЫ)

(Фамилия, И.О.)

Группа	N3347
--------	-------

( Фамилия, И.О., место работы, должность, ученое звание, степень )

Наименование темы	Разработка многопользовательского веб-интерфейса для управления базой данных информационной системы компании, предоставляющей услуги коммуникаций и видеонаблюдения
-------------------	---

№ п/п	Наименование этапа	Дата завершения		Оценка и подпись руководителя
		Планируемая	Фактическая	
1	Инфологическое моделирование базы данных	12.09.2024	06.11.2024	
2	Реализация БД в рамках СУБД	26.09.2024	06.11.2024	
3	Реализация защиты базы данных	10.10.2024	06.11.2024	
4	Реализация сервиса для взаимодействия с базой данных	24.10.2024	06.11.2024	
5	Резервирование БД и восстановление по контрольным точкам	07.11.2024	06.11.2024	

подпись, дата

подпись, дата

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО  
ОБРАЗОВАНИЯ  
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

**АННОТАЦИЯ НА КУРСОВОЙ ПРОЕКТ (РАБОТУ)**

Студент	Климов Даниил Иванович
	(Фамилия, И.О.)
Факультет	Безопасности информационных технологий
Группа	N3347
Направление (специальность)	Информационная безопасность
Руководитель	Таранов Сергей Владимирович, ординарный доцент, доцент, к.т.н.
	( Фамилия, И.О., место работы, должность, ученое звание, степень )
Дисциплина	Информационная безопасность баз данных
Наименование темы	Разработка многопользовательского веб-интерфейса для управления базой данных информационной системы компании, предоставляющей услуги коммуникаций и видеонаблюдения

**ХАРАКТЕРИСТИКА КУРСОВОГО ПРОЕКТА (РАБОТЫ)**

**1. Цель и задачи работы**

☒ Предложены  
студентом

☐ Сформулированы при участии  
студента

☐ Определены руководителем

Цель работы: разработать многопользовательский веб-интерфейс для управления базой данных  
информационной системы компании, предоставляющей услуги коммуникаций и видеонаблюдения

**2. Характер работы**

☐ Расчет

☒ Конструирование

☐ Моделирование

☐ Другое,

**3. Содержание работы**

Работа содержит следующие основные шаги: инфологическое моделирование баз данных по методу  
«сущность-связь» , реализация БД в рамках СУБД, обеспечение защиты БД, реализация сервиса  
для взаимодействия с БД, обеспечение резервирования БД и восстановления по контрольным точкам

#### 4. Выводы

В ходе выполнения курсовой работы была спроектирована и реализована база данных для компании, предоставляющей услуги коммуникаций и видеонаблюдения. Далее был разработан

---

данных включает разграничение прав пользователей и ограничение доступа, что обеспечивает защиту

---

многопользовательский веб-интерфейс для управления этой базой данных. Система безопасности базы

---

данных. Для повышения надежности работы базы данных реализованы методы резервного копирования

---

и восстановления.

---

Студент \_\_\_\_\_

(подпись)

Руководитель \_\_\_\_\_

(подпись)

«\_\_» \_\_\_\_\_ 20\_\_ г.

# СОДЕРЖАНИЕ

Список сокращений и условных обозначений .....	9
Введение .....	10
1     Инфологическое моделирование базы данных.....	11
1.1   Системный анализ информационной системы .....	11
1.1.1   Цели: .....	11
1.1.2   Задачи: .....	11
1.1.3   Источники данных: .....	11
1.1.4   Потребители информации: .....	11
1.1.5   Ограничения:.....	12
1.2   Выделение сущностей и построение ER – диаграмм.....	12
1.2.1   Сущности.....	12
1.2.2   Описание связей .....	12
1.2.3   ER-диаграмма.....	14
1.3   Преобразование ER-диаграммы в схему отношений с помощью правил формирования предварительных отношений.....	15
1.3.1   Описание связей .....	15
1.3.2   Схема отношений .....	15
1.4   Приведение схемы предварительных отношений к ЗНФ.....	17
1.5   Моделирование уровня представлений ИС компании, предоставляющей услуги коммуникаций и видеонаблюдения.....	18
1.5.1   Для клиентов:.....	18
1.5.2   Для сотрудников:.....	18
1.5.3   Для складских работников: .....	18
2     Реализация базы данных в рамках СУБД.....	19
2.1   Выбор системы управления базами данных .....	19
2.2   Создание БД в выбранной СУБД.....	20
2.3   Индексация таблиц .....	26
2.4   Установление связей между таблицами .....	27
2.5   Тестовые запросы к БД .....	28
2.6   Создание представлений.....	29
3     Обеспечение защиты базы данных .....	32
3.1   Задачи по мониторингу БД.....	32
3.2   Задачи по шифрованию данных.....	35

3.3	Задачи по разграничению доступа в БД.....	36
4	Реализация сервиса для взаимодействия с разработанной базой данных .....	40
4.1	Выбор средств разработки.....	40
4.1.1	Серверная часть (Backend).....	40
4.1.2	Клиентская часть (Frontend) .....	41
4.1.3	Взаимодействие Frontend и Backend.....	41
4.2	Взаимодействие сервиса и базы данных .....	42
4.3	Функционал сервиса.....	49
4.3.1	Авторизация .....	50
4.3.2	Безопасный доступ к API.....	59
5	Резервное копирование и восстановление .....	77
	Заключение.....	78

## **СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ**

ИС – Информационная система.

БД – База данных.

СУБД – Система управления базами данных.

ПО – Программное обеспечение.



## **ВВЕДЕНИЕ**

Цель работы – разработка многопользовательского веб-интерфейса для управления базой данных информационной системы компании, предоставляющей услуги коммуникаций и видеонаблюдения. Веб-интерфейс должен предоставлять функционал, необходимый для удобного взаимодействия сотрудников с базами данных.

Для достижения поставленной цели необходимо решить следующие задачи:

- смоделировать базу данных;
- реализовать базу данных в рамках СУБД;
- настроить простейшую систему безопасности в созданной БД;
- реализовать сервис для взаимодействия с разработанной базой данных
- изучить возможности резервного копирования БД.

# **1 ИНФОЛОГИЧЕСКОЕ МОДЕЛИРОВАНИЕ БАЗЫ ДАННЫХ**

Выбранной предметной областью является информационная система компании, предоставляющей услуги в сфере коммуникаций и видеонаблюдения. Разрабатываемая база данных должна поддерживать следующие процессы:

- Управление клиентами и их заказами;
- Ведение учета используемого оборудования;
- Координация задач сотрудников, связанных с установкой и обслуживанием систем.

## **1.1 Системный анализ информационной системы**

### **1.1.1 Цели:**

- Упрощение управления данными о клиентах и предоставляемых услугах (установка и обслуживание видеонаблюдения и коммуникаций);
- Обеспечение централизованного хранения и обработки информации для различных категорий пользователей.

### **1.1.2 Задачи:**

- Учет клиентов, включая контактные данные и историю обслуживания;
- Управление заказами на установку и настройку оборудования;
- Ведение сведений об оборудовании (включая статус и местоположение);
- Учет сотрудников, их ролей и текущих задач.

### **1.1.3 Источники данных:**

- Заявки клиентов, поступающие через веб-интерфейс;
- Внутренние отчеты об установках и обслуживании;
- Информация о поставках оборудования.

### **1.1.4 Потребители информации:**

- Администраторы: Управление всей базой данных;
- Сотрудники: Получение сведений о заказах и задачах;
- Клиенты: Доступ к информации о статусе своих заявок.

### **1.1.5 Ограничения:**

- Каждый заказ привязан только к одному клиенту;
- Каждое оборудование может быть связано только с одним заказом в данный момент времени;
- Сотрудник может быть назначен на несколько заказов, но не может выполнять их одновременно.

## **1.2 Выделение сущностей и построение ER – диаграмм**

### **1.2.1 Сущности**

- Клиент
- Заказ
- Оборудование
- Сотрудник
- Задача
- Склад
- Платежи

### **1.2.2 Описание связей**

- Клиент ↔ Заказ: 1:M

Один клиент может иметь множество заказов. Например, один клиент может заказывать разные услуги на установку видеонаблюдения или другие коммуникации в разное время. Заказ всегда должен быть связан с клиентом, так как он создан именно для него. При этом клиент может существовать без заказов, например, до того, как он впервые что-либо закажет.

- Заказ ↔ Оборудование: 1:M

Один заказ может включать несколько единиц оборудования. Например, при установке системы видеонаблюдения могут быть заказаны несколько камер, блоки питания и другие компоненты. Один заказ должен обязательно включать хотя бы одно оборудование, иначе установка не имеет смысла. Оборудование также может быть свободно и не связано с заказом, например, пока оно хранится на складе.

- Заказ ↔ Задача: 1:M

Один заказ может включать несколько задач. Например, для выполнения заказа на установку системы видеонаблюдения может потребоваться выполнение нескольких задач,

таких как монтаж камер, настройка сетевого соединения и тестирование работы системы. Каждая задача должна быть привязана к заказу, чтобы было понятно, какой заказ она обслуживает, но задачи могут оставаться незакрепленными, если они еще не распределены.

– Заказ ↔ Платежи: 1:M

Один заказ может иметь несколько платежей. Например, клиент может оплатить заказ частями или несколько раз за различные этапы установки. Каждый платеж относится к конкретному заказу, и заказ не может существовать без хотя бы одного платежа, чтобы покрыть его стоимость.

– Сотрудник ↔ Задача: 1:M

Один сотрудник может выполнять множество задач. Например, монтажник может заниматься установкой камер для нескольких заказов, поэтому ему присваиваются соответствующие задачи. При этом одна задача всегда должна быть привязана к одному сотруднику, чтобы была ясность, кто именно за нее отвечает.

– Склад ↔ Оборудование: 1:M

Один склад может хранить несколько единиц оборудования. Например, на одном складе могут находиться десятки камер, кабели и другие компоненты. Оборудование должно быть привязано к складу, чтобы было понятно, где оно находится. Оборудование не может существовать вне склада, когда оно не установлено в рамках заказа.

– Сотрудник ↔ Склад: 1:M

Один сотрудник может быть ответственным за несколько складов. Например, менеджер по складам может управлять запасами на нескольких объектах. Каждый склад должен иметь ответственного сотрудника, который будет следить за его состоянием и инвентаризацией.

– Сотрудник ↔ Клиент: 1:M

Один сотрудник может работать с несколькими клиентами, и один клиент может быть связан с несколькими сотрудниками. Например, менеджер по продажам может взаимодействовать с несколькими клиентами, консультируя их по услугам. В то же время один клиент может получать услуги от разных сотрудников (например, менеджера и техподдержки).

### 1.2.3 ER-диаграмма

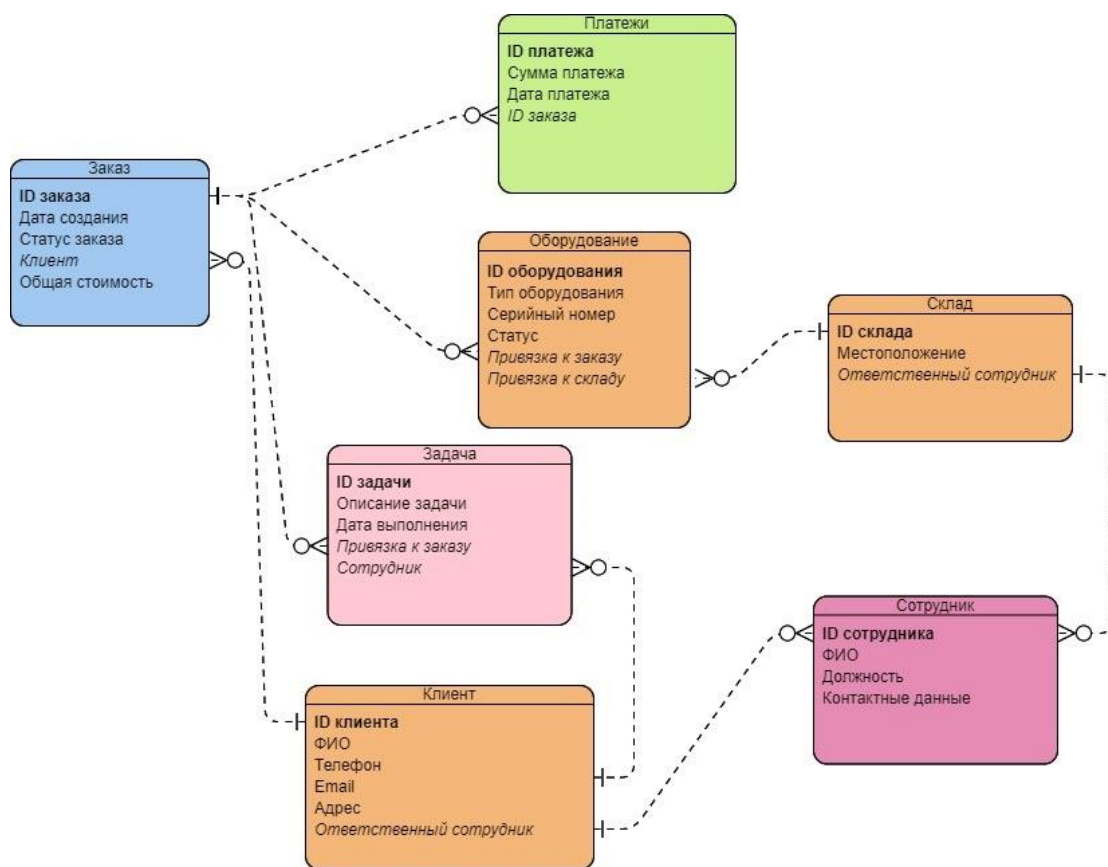


Рисунок 1 – ER-диаграмма информационной системы

### **1.3 Преобразование ER-диаграммы в схему отношений с помощью правил формирования предварительных отношений**

#### **1.3.1 Описание связей**

- Клиент ↔ Заказ: 1:М (один клиент имеет много заказов);
- Заказ ↔ Оборудование: 1:М (один заказ включает несколько единиц оборудования);
- Заказ ↔ Задача: 1:М (один заказ может содержать несколько задач);
- Заказ ↔ Платежи: 1:М (в один заказ может иметь несколько платежей);
- Сотрудник ↔ Задача: 1:М (один сотрудник выполняет несколько задач);
- Склад ↔ Оборудование: 1:М (на одном складе может храниться несколько единиц оборудования);
- Сотрудник ↔ Склад: 1:М (один сотрудник может быть ответственным за несколько складов);
- Сотрудник ↔ Клиент: 1:М (один сотрудник может работать с несколькими клиентами, и один клиент может быть связан с несколькими сотрудниками).

#### **1.3.2 Схема отношений**

- Клиент:
  - ID клиента (первичный ключ);
  - ФИО;
  - Телефон;
  - Email;
  - Адрес.
- Заказ:
  - ID заказа (первичный ключ);
  - Дата создания;
  - Статус заказа;
  - Клиент (внешний ключ);
  - Общая стоимость.
- Оборудование:
  - ID оборудования (первичный ключ);
  - Тип оборудования;
  - Серийный номер;

- Статус (установлено, в ремонте, доступно для использования);
  - Привязка к заказу (внешний ключ);
  - Привязка к складу (внешний ключ).
- Сотрудник:
  - ID сотрудника (первичный ключ);
  - ФИО;
  - Должность;
  - Контактные данные.
- Задача:
  - ID задачи (первичный ключ);
  - Описание задачи;
  - Дата выполнения;
  - Привязка к заказу (внешний ключ);
  - Сотрудник (внешний ключ).
- Склад:
  - ID склада (первичный ключ);
  - Местоположение;
  - Ответственный сотрудник (внешний ключ);
- Платежи:
  - ID платежа (первичный ключ);
  - Сумма платежа;
  - Дата платежа;
  - ID заказа (внешний ключ).

#### **1.4 Приведение схемы предварительных отношений к 3НФ**

- 1 НФ - все атрибуты должны быть простыми.

Все атрибуты в представленной базе данных являются простыми. ФИО в данном случае тоже является атомарным, так как в ходе работы с БД будет обращение всегда ко всему ФИО, нет потребности в обращении конкретно к имени, фамилии или отчеству

- 2 НФ - каждый неключевой атрибут функционально полно зависит от первичного ключа

Данное свойство выполняется во всех отношениях базы данных, а также отношения находятся в 1НФ, поэтому можно утверждать, что отношения находятся в 2 НФ.

- 3 НФ - каждый неключевой атрибут нетранзитивно зависит от первичного ключа

Во всех отношениях неключевые атрибуты нетранзитивно зависят от первичного ключа и отношения находятся в 2НФ, поэтому все отношения находятся в третьей нормальной форме



## **1.5 Моделирование уровня представлений ИС компании, предоставляющей услуги коммуникаций и видеонаблюдения**

### **1.5.1 Для клиентов:**

- Статус заказов и платежей:
  - Это представление содержит информацию о заказах клиентов, включая их статус, дату создания, список используемого оборудования и информацию о платежах.
  - Атрибуты: ID заказа, дата создания, статус заказа, список оборудования, сумма и дата платежей, общая стоимость заказа.

### **1.5.2 Для сотрудников:**

- Текущие задачи и склад:
  - Это представление позволяет сотрудникам видеть свои текущие задачи и информацию о складе, связанную с выполняемыми заказами. Например, оно отображает, какие задачи назначены сотруднику, когда они должны быть выполнены и какое оборудование хранится на складе.
  - Атрибуты: ID задачи, описание задачи, срок выполнения, ID заказа, информация о клиенте, место хранения оборудования, ответственный сотрудник склада.Добавленные представления:

### **1.5.3 Для складских работников:**

- Запасы оборудования на складе:
  - Представление включает информацию об оборудовании, которое хранится на складе, его тип, количество и текущий статус.
  - Атрибуты: ID оборудования, тип оборудования, статус (например, доступно, в ремонте), количество на складе.

## **2 РЕАЛИЗАЦИЯ БАЗЫ ДАННЫХ В РАМКАХ СУБД**

### **2.1 Выбор системы управления базами данных**

В качестве СУБД для проекта я выбрал PostgreSQL, и у этого выбора есть несколько весомых причин, которые соответствуют моему опыту и технологиям, которые я использую как веб-разработчик.

Во-первых, PostgreSQL — это мощная и бесплатная СУБД, которая поддерживает все необходимые функции для надежного хранения и обработки данных, включая работу с внешними ключами, индексацию и создание представлений. Её возможности позволяют эффективно масштабировать приложение и обеспечивать высокую производительность даже при росте нагрузки.

Во-вторых, PostgreSQL прекрасно интегрируется с современными веб-фреймворками, такими как Next.js, с которым я активно работаю. Благодаря поддержке ORM-решений и простоте взаимодействия через API, работа с базой данных становится интуитивной и удобной. В Next.js легко подключиться к PostgreSQL, использовать автоматическое создание миграций и эффективно управлять схемой базы данных, что ускоряет разработку и делает её более гибкой.

Наконец, есть Prisma ORM, в которой доступен PostgreSQL и которая улучшает опыт разработчика, обеспечивая удобные инструменты для работы с данными, автоматическое создание миграций и поддержку типизации. Это значительно упрощает процесс разработки и делает его более гибким и продуктивным. В будущем планируется использовать для управления БД только инструмент Prisma ORM.

## 2.2 Создание БД в выбранной СУБД

Для создания базы данных я использовал IDE от JetBrains — DataGrip, которая позволяет подключиться к серверу PostgreSQL на локальной машине. С её помощью была создана база данных, содержащая все отношения в соответствии с разработанной в ЛР1 схемой. DataGrip обеспечивает удобный интерфейс для управления базой данных, что делает процесс разработки более эффективным и наглядным.

### Листинг 1 – Код для создания базы данных

```
CREATE DATABASE company_services_db
WITH
  OWNER = postgres
  ENCODING = 'UTF8'
  TABLESPACE = pg_default
  CONNECTION LIMIT = -1;
```

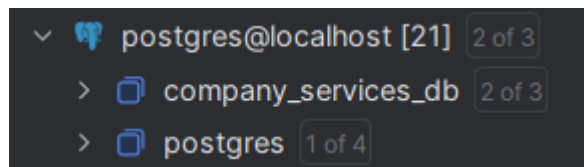


Рисунок 2 – База данных успешно создана

### Листинг 2 – Код для создания таблиц

```
-- Таблица Сотрудников
CREATE TABLE IF NOT EXISTS public.employee (
  employee_id SERIAL PRIMARY KEY,
  full_name VARCHAR(255),
  position VARCHAR(100),
  phone VARCHAR(50),
  email VARCHAR(255)
);

-- Таблица Клиентов
CREATE TABLE IF NOT EXISTS public.client (
  client_id SERIAL PRIMARY KEY,
  full_name VARCHAR(255),
  phone VARCHAR(50),
  email VARCHAR(255),
  address TEXT,
  responsible_employee INTEGER REFERENCES public.employee(employee_id)
);

-- Таблица Заказов
CREATE TABLE IF NOT EXISTS public.order (
  order_id SERIAL PRIMARY KEY,
  creation_date DATE,
  status VARCHAR(50),
  client_id INTEGER REFERENCES public.client(client_id),
  total_cost DECIMAL(10, 2)
);
```

```

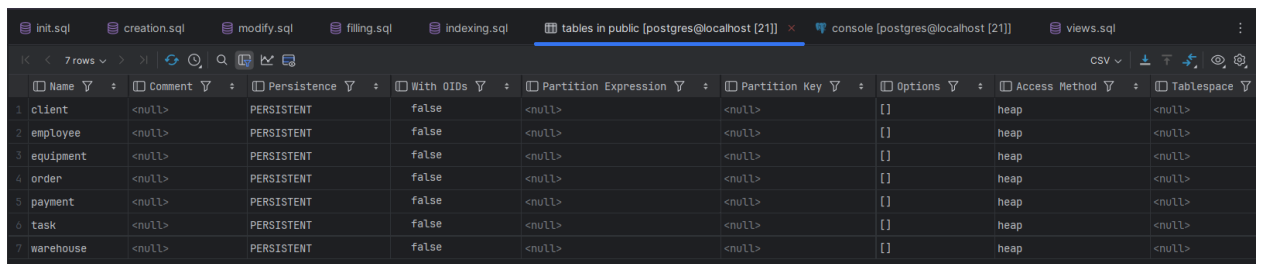
-- Таблица Складов
CREATE TABLE IF NOT EXISTS public.warehouse (
    warehouse_id SERIAL PRIMARY KEY,
    location TEXT,
    responsible_employee INTEGER REFERENCES public.employee(employee_id)
);

-- Таблица Оборудования
CREATE TABLE IF NOT EXISTS public.equipment (
    equipment_id SERIAL PRIMARY KEY,
    equipment_type VARCHAR(255),
    serial_number VARCHAR(255),
    status VARCHAR(50),
    order_id INTEGER REFERENCES public.order(order_id),
    warehouse_id INTEGER REFERENCES public.warehouse(warehouse_id)
);

-- Таблица Задач
CREATE TABLE IF NOT EXISTS public.task (
    task_id SERIAL PRIMARY KEY,
    description TEXT,
    due_date DATE,
    order_id INTEGER REFERENCES public.order(order_id),
    employee_id INTEGER REFERENCES public.employee(employee_id)
);

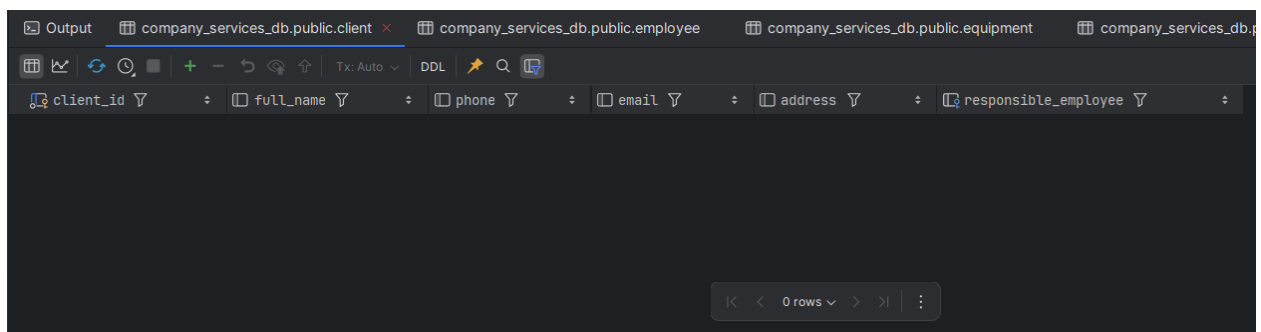
-- Таблица Платежей
CREATE TABLE IF NOT EXISTS public.payment (
    payment_id SERIAL PRIMARY KEY,
    amount DECIMAL(10, 2),
    payment_date DATE,
    order_id INTEGER REFERENCES public.order(order_id)
);

```



Name	Comment	Persistence	With OIDs	Partition Expression	Partition Key	Options	Access Method	Tablespace
client	<null>	PERSISTENT	false	<null>	<null>	[]	heap	<null>
employee	<null>	PERSISTENT	false	<null>	<null>	[]	heap	<null>
equipment	<null>	PERSISTENT	false	<null>	<null>	[]	heap	<null>
order	<null>	PERSISTENT	false	<null>	<null>	[]	heap	<null>
payment	<null>	PERSISTENT	false	<null>	<null>	[]	heap	<null>
task	<null>	PERSISTENT	false	<null>	<null>	[]	heap	<null>
warehouse	<null>	PERSISTENT	false	<null>	<null>	[]	heap	<null>

Рисунок 3 – Список созданных таблиц



client_id	full_name	phone	email	address	responsible_employee
0 rows					

Рисунок 4 – Созданная таблица клиентов

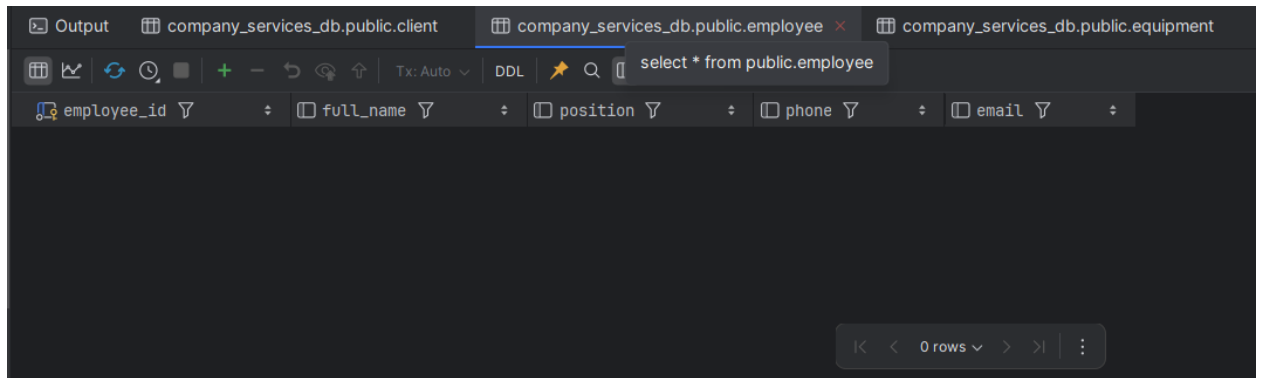


Рисунок 5 – Созданная таблица работников



Рисунок 6 – Созданная таблица оборудования

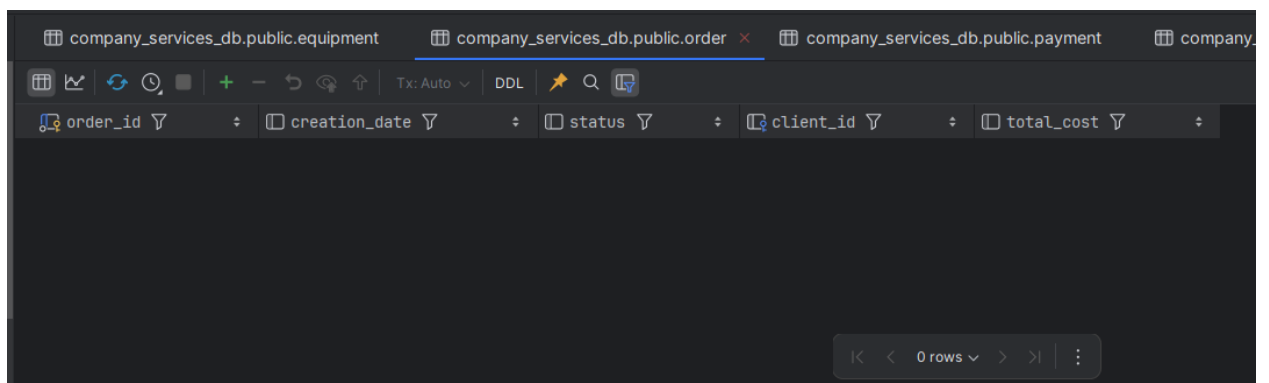


Рисунок 7 – Созданная таблица заказов

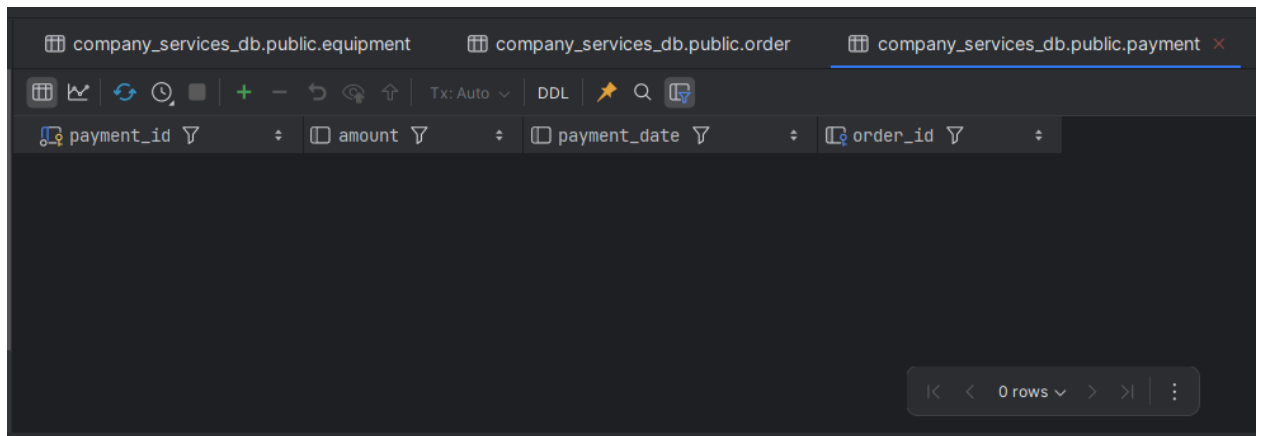


Рисунок 8 – Созданная таблица платежей

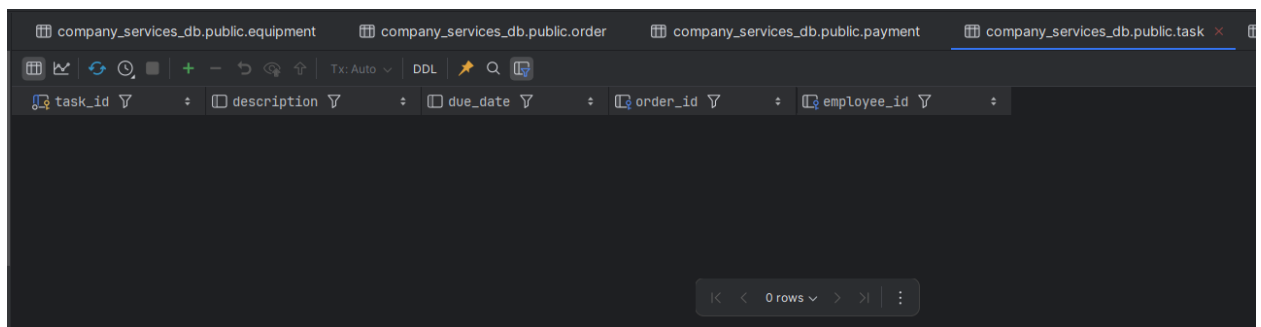


Рисунок 9 – Созданная таблица задач

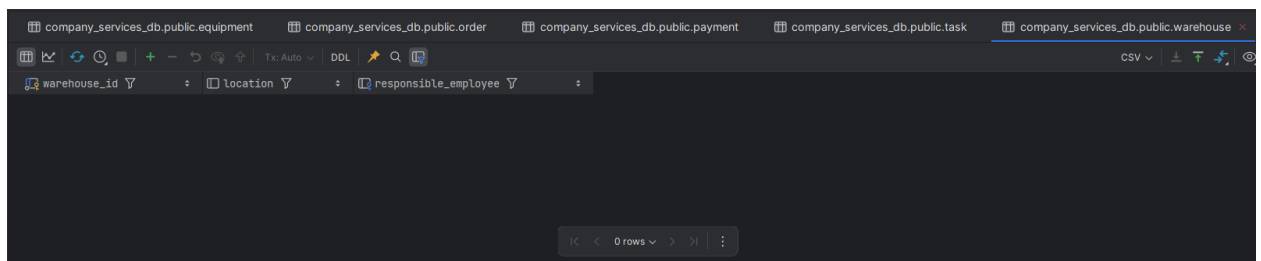


Рисунок 10 – Созданная таблица складов

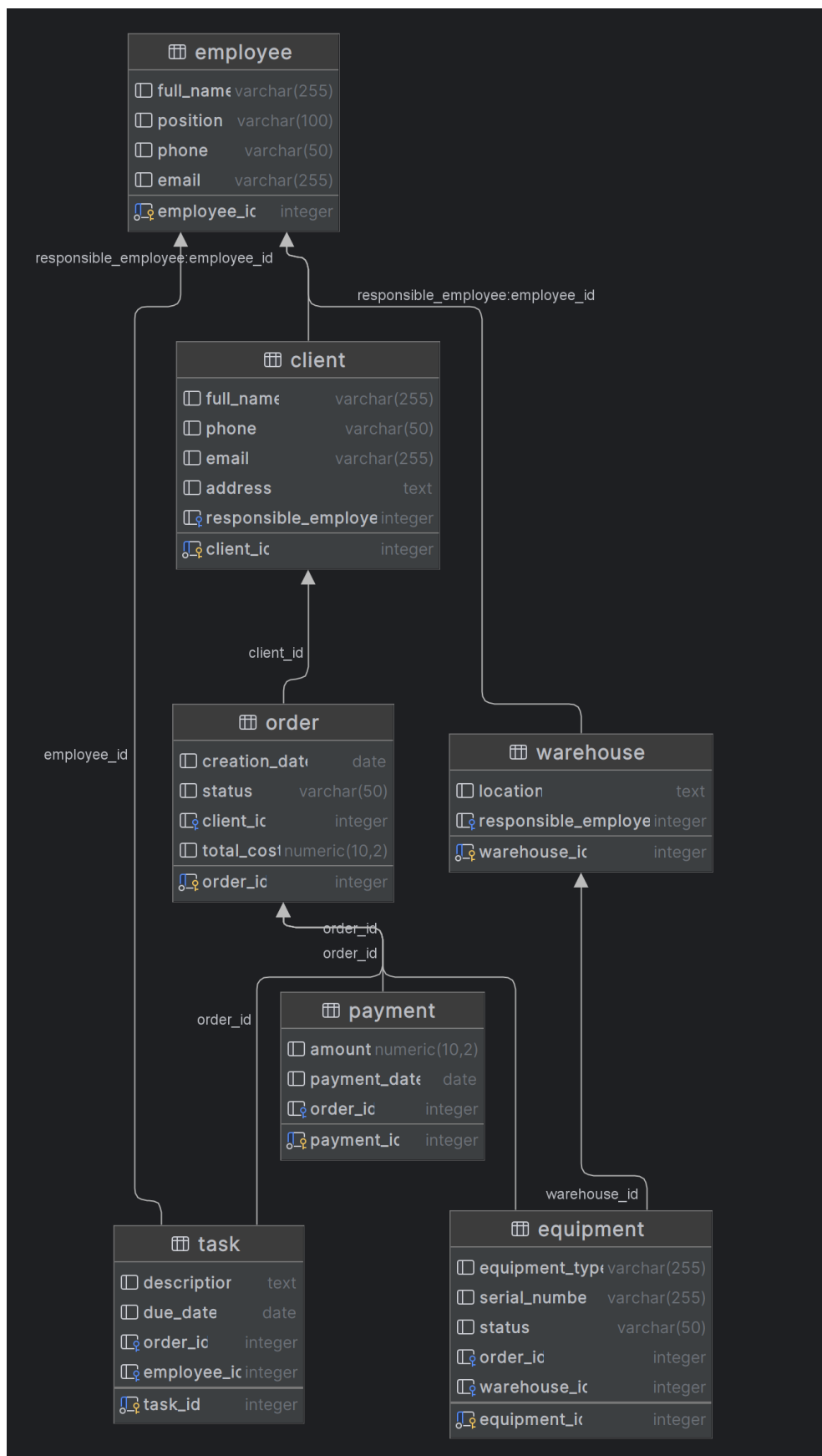


Рисунок 11 – Диаграмма созданных таблиц, соответствующая схеме отношений из ЛР1

### Листинг 3 – Код заполнения таблиц

```
-- Заполнение таблицы Сотрудников
INSERT INTO public.employee (full_name, position, phone, email)
VALUES ('Смирнов Алексей', 'Менеджер', '89998887766', 'smirnov@mail.ru'),
      ('Сидоров Андрей', 'Техник', '89991234567', 'sidorov@mail.ru'),
      ('Ковалев Дмитрий', 'Специалист по установке', '89990001122',
'kovalev@mail.ru'),
      ('Морозова Наталья', 'Кладовщик', '89995554433', 'morozova@mail.ru'),
      ('Зайцев Максим', 'Инженер', '89996667788', 'zaytsev@mail.ru'),
      ('Васильева Ольга', 'Техподдержка', '89991112233',
'vasileva@mail.ru'),
      ('Ильин Игорь', 'Менеджер', '89994443322', 'ilin@mail.ru');

-- Заполнение таблицы Клиентов
INSERT INTO public.client (full_name, phone, email, address,
responsible_employee)
VALUES ('Иванов Иван', '89001234567', 'ivanov@mail.ru', 'ул. Пушкина, д. 10',
1),
      ('Петров Петр', '89009876543', 'petrov@mail.ru', 'ул. Лермонтова, д.
15', 2),
      ('Сидорова Анна', '89001112233', 'sidorova@mail.ru', 'ул. Чехова, д.
8', 3),
      ('Кузнецов Олег', '89002223344', 'kuznetsov@mail.ru', 'ул.
Маяковского, д. 3', 4),
      ('Смирнова Мария', '89009998877', 'smirnova@mail.ru', 'ул. Садовая, д.
12', 1),
      ('Федоров Алексей', '89003334455', 'fedorov@mail.ru', 'ул. Кирова, д.
21', 2),
      ('Михайлова Оксана', '89004445566', 'mihailova@mail.ru', 'ул.
Тверская, д. 18', 3);

-- Заполнение таблицы Заказов
INSERT INTO public.order (creation_date, status, client_id, total_cost)
VALUES ('2024-11-10', 'в процессе', 1, 50000.00),
      ('2024-11-11', 'завершен', 2, 15000.00),
      ('2024-11-12', 'в процессе', 3, 30000.00),
      ('2024-11-13', 'завершен', 4, 25000.00),
      ('2024-11-14', 'в процессе', 5, 40000.00),
      ('2024-11-15', 'отменен', 6, 10000.00),
      ('2024-11-16', 'завершен', 7, 35000.00);

-- Заполнение таблицы Складов
INSERT INTO public.warehouse (location, responsible_employee)
VALUES ('Склад на ул. Ленина', 4),
      ('Склад на ул. Кирова', 1),
      ('Склад на ул. Советская', 2),
      ('Склад на ул. Гагарина', 5),
      ('Склад на ул. Жукова', 3),
      ('Склад на ул. Пушкина', 6),
      ('Склад на ул. Дружбы', 1),
      ('Склад на ул. Победы', 2);

-- Заполнение таблицы Оборудования
INSERT INTO public.equipment (equipment_type, serial_number, status,
order_id, warehouse_id)
VALUES ('Камера видеонаблюдения', 'SN001', 'установлено', 1, 1),
      ('Блок питания', 'SN002', 'доступно', 2, 1),
```



```

('Датчик движения', 'SN003', 'в ремонте', 3, 2),
('Камера видеонаблюдения', 'SN004', 'установлено', 4, 2),
('Коммутатор', 'SN005', 'доступно', 5, 1),
('Блок питания', 'SN006', 'установлено', 6, 3),
('Камера видеонаблюдения', 'SN007', 'установлено', 7, 2);

-- Заполнение таблицы Задач
INSERT INTO public.task (description, due_date, order_id, employee_id)
VALUES ('Установка камер в офисе', '2024-11-15', 1, 2),
('Проверка состояния оборудования', '2024-11-20', 2, 3),
('Настройка системы видеонаблюдения', '2024-11-18', 3, 5),
('Монтаж блока питания', '2024-11-19', 4, 3),
('Проверка подключения', '2024-11-21', 5, 6),
('Установка датчиков движения', '2024-11-22', 1, 2),
('Тестирование системы', '2024-11-23', 3, 4);

-- Заполнение таблицы Платежей
INSERT INTO public.payment (amount, payment_date, order_id)
VALUES (25000.00, '2024-11-12', 1),
(15000.00, '2024-11-13', 2),
(10000.00, '2024-11-14', 3),
(20000.00, '2024-11-15', 4),
(15000.00, '2024-11-16', 5),
(5000.00, '2024-11-17', 6),
(35000.00, '2024-11-18', 7);

```

#### Листинг 4 – Код модификации таблицы

```

ALTER TABLE public.client
ADD COLUMN additional_info TEXT;

```

	client_id	full_name	phone	email	address	re...	additional_info
1	1	Иванов Иван	89001234567	ivanov@mail.ru	ул. Пушкина, д. 10	1	<null>
2	2	Петров Петр	89009876543	petrov@mail.ru	ул. Лермонтова, д. 15	2	<null>
3	3	Сидорова Анна	89001112233	sidorova@mail.ru	ул. Чехова, д. 8	3	<null>
4	4	Кузнецов Олег	89002223344	kuznetsov@mail.ru	ул. Маяковского, д. 3	4	<null>
5	5	Смирнова Мария	89009998877	smirnova@mail.ru	ул. Садовая, д. 12	1	<null>
6	6	Федоров Алексей	89003334455	fedorov@mail.ru	ул. Кирова, д. 21	2	<null>
7	7	Михайлова Оксана	89004445566	mihailova@mail.ru	ул. Тверская, д. 18	3	<null>

Рисунок 12 – Таблица клиентов после добавления столбца с дополнительной информацией

## 2.3 Индексация таблиц

Для оптимизации работы с базой данных были добавлены индексы для атрибутов, по которым происходит объединение таблиц, а также для атрибутов, по которым чаще всего выполняется поиск или фильтрация данных. Индексы позволяют значительно ускорить

операции выборки, особенно в случаях, когда таблицы содержат большое количество записей.

В данном случае индексы созданы для следующих атрибутов:

- `client_id` в таблице `order` — индексирование этого атрибута ускоряет операции, связанные с объединением заказов и клиентов, так как `client_id` часто используется для установления связи между таблицами.
- `employee_id` в таблице `task` — добавление индекса для этого атрибута позволяет быстрее находить задачи, связанные с конкретным сотрудником, что полезно для объединения и фильтрации данных.
- `warehouse_id` в таблице `equipment` — индексирование `warehouse_id` ускоряет доступ к данным об оборудовании, связанном с конкретным складом, что критически важно для операций поиска и фильтрации.

Листинг 5 – Код для создания индексов

```
CREATE INDEX idx_order_client ON public.order (client_id);  
CREATE INDEX idx_task_employee ON public.task (employee_id);  
CREATE INDEX idx_equipment_warehouse ON public.equipment (warehouse_id);
```

## 2.4 Установление связей между таблицами

На этапе создания таблиц в СУБД были установлены все необходимые взаимосвязи между таблицами с использованием внешних ключей. Эти связи обеспечивают целостность данных и позволяют организовать правильные зависимости между сущностями. Ниже приведены основные связи:

- Клиент ↔ Заказ: связь 1:M реализована через внешний ключ `client_id` в таблице `order`, что позволяет каждому заказу быть связанным с конкретным клиентом;
- Заказ ↔ Оборудование: связь 1:M реализована через внешний ключ `order_id` в таблице `equipment`, что позволяет каждому заказу включать несколько единиц оборудования;
- Заказ ↔ Задача: связь 1:M реализована через внешний ключ `order_id` в таблице `task`, что позволяет каждому заказу содержать несколько задач;
- Заказ ↔ Платежи: связь 1:M реализована через внешний ключ `order_id` в таблице `payment`, что позволяет одному заказу иметь несколько платежей;
- Сотрудник ↔ Задача: связь 1:M реализована через внешний ключ `employee_id` в таблице `task`, что позволяет одному сотруднику выполнять несколько задач;

- Склад ↔ Оборудование: связь 1:M реализована через внешний ключ `warehouse_id` в таблице `equipment`, что позволяет одному складу хранить несколько единиц оборудования;

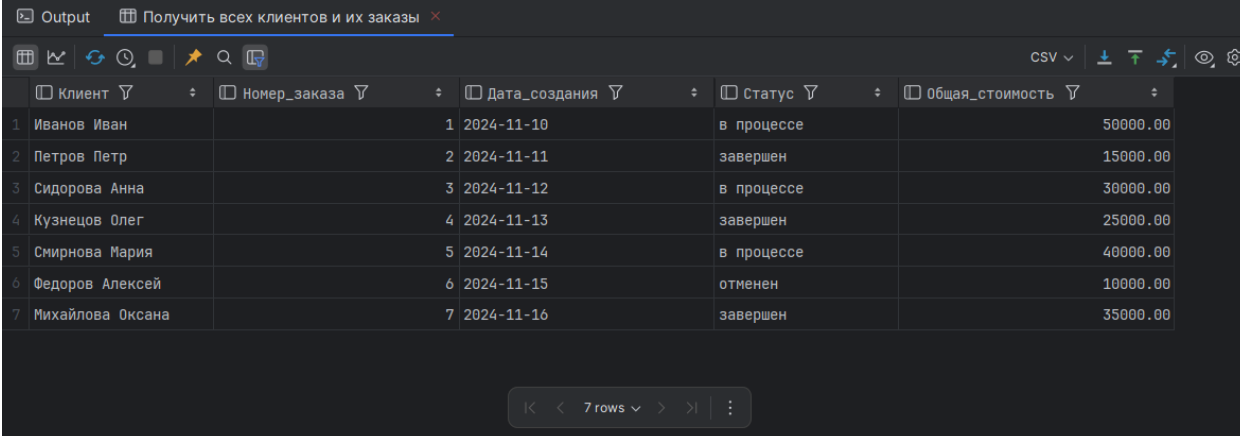
- Сотрудник ↔ Клиент: связь M:M реализована с помощью связующей таблицы `employee_client`, что позволяет одному сотруднику работать с несколькими клиентами и одному клиенту взаимодействовать с несколькими сотрудниками.

Таким образом, все взаимосвязи между таблицами были установлены на этапе создания таблиц, что обеспечивает правильное функционирование базы данных и возможность реализации сложных бизнес-процессов.

## 2.5 Тестовые запросы к БД

Листинг 6 – Код запроса, чтобы получить всех клиентов и их заказы

```
SELECT client.full_name      AS Клиент,  
       "order".order_id     AS Номер_заказа,  
       "order".creation_date AS Дата_создания,  
       "order".status       AS Статус,  
       "order".total_cost   AS Общая_стоимость  
FROM public.client  
      LEFT JOIN  
      public."order" ON client.client_id = "order".client_id;
```



	Клиент	Номер_заказа	Дата_создания	Статус	Общая_стоимость
1	Иванов Иван	1	2024-11-10	в процессе	50000.00
2	Петров Петр	2	2024-11-11	завершен	15000.00
3	Сидорова Анна	3	2024-11-12	в процессе	30000.00
4	Кузнецов Олег	4	2024-11-13	завершен	25000.00
5	Смирнова Мария	5	2024-11-14	в процессе	40000.00
6	Федоров Алексей	6	2024-11-15	отменен	10000.00
7	Михайлова Оксана	7	2024-11-16	завершен	35000.00

Рисунок 13 – Результат выполненного запроса

Листинг 7 – Код запроса, чтобы получить оборудование, привязанное к каждому заказу

```
SELECT "order".order_id      AS Номер_заказа,  
       equipment.equipment_type AS Тип_оборудования,  
       equipment.serial_number AS Серийный номер,  
       equipment.status       AS Статус_оборудования  
FROM public."order"  
      LEFT JOIN  
      public.equipment ON "order".order_id = equipment.order_id;
```

Output: Получить оборудование к каждому заказу

	Номер_заказа	Тип_оборудования	Серийный_номер	Статус_оборудования
1	1	Камера видеонаблюдения	SN001	установлено
2	5	Коммутатор	SN005	доступно
3	7	Камера видеонаблюдения	SN007	установлено
4	3	Датчик движения	SN003	в ремонте
5	4	Камера видеонаблюдения	SN004	установлено
6	6	Блок питания	SN006	установлено
7	2	Блок питания	SN002	доступно

7 rows

Рисунок 14 – Результат выполненного запроса

Листинг 8 – Код запроса, чтобы проверить завершенные заказы и их общую стоимость

```
SELECT "order".order_id AS Номер_заказа,
       client.full_name AS Клиент,
       "order".total_cost AS Общая_стоимость,
       "order".status AS Статус
FROM public."order"
JOIN
    public.client ON "order".client_id = client.client_id
WHERE "order".status = 'завершен';
```

Output: Проверить завершенные... и их общую стоимость

	Номер_заказа	Клиент	Общая_стоимость	Статус
1	2	Петров Петр	15000.00	завершен
2	4	Кузнецов Олег	25000.00	завершен
3	7	Михайлова Оксана	35000.00	завершен

3 rows

Рисунок 15 – Результат выполненного запроса

## 2.6 Создание представлений

Листинг 9 – Код для создания представлений, составленных в пункте 5 лабораторной 1

```
-- Представление: Статус заказов и платежей для клиентов
CREATE VIEW public.client_orders_payments AS
SELECT
    client.full_name,
    "order".creation_date,
    "order".status,
    equipment.equipment_type,
    payment.amount,
    payment.payment_date
```

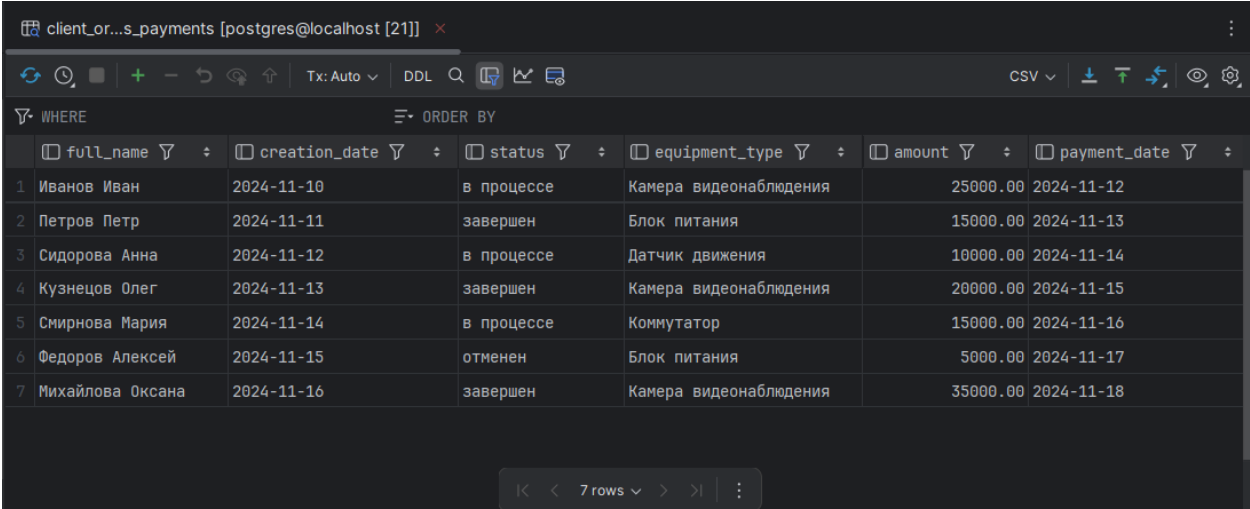
```

FROM public.client
    JOIN public."order" ON client.client_id = "order".client_id
    LEFT JOIN public.equipment ON "order".order_id = equipment.order_id
    LEFT JOIN public.payment ON "order".order_id = payment.order_id;

-- Представление: Текущие задачи для сотрудников
CREATE VIEW public.employee_tasks AS
SELECT
    employee.full_name,
    task.description,
    task.due_date,
    "order".creation_date,
    client.full_name AS client_name
FROM public.employee
    JOIN public.task ON employee.employee_id = task.employee_id
    JOIN public."order" ON task.order_id = "order".order_id
    JOIN public.client ON "order".client_id = client.client_id;

-- Представление: Запасы оборудования на складе
CREATE VIEW public.warehouse_equipment AS
SELECT
    warehouse.location,
    equipment.equipment_type,
    equipment.status,
    COUNT(equipment.equipment_id) AS equipment_count
FROM public.warehouse
    JOIN public.equipment ON warehouse.warehouse_id =
equipment.warehouse_id
GROUP BY warehouse.location, equipment.equipment_type, equipment.status;

```



The screenshot shows a PostgreSQL client window titled "client\_or...s\_payments [postgres@localhost [21]]". The interface displays a table with 7 rows of data. The columns are: full\_name, creation\_date, status, equipment\_type, amount, and payment\_date. The data is as follows:

	full_name	creation_date	status	equipment_type	amount	payment_date
1	Иванов Иван	2024-11-10	в процессе	Камера видеонаблюдения	25000.00	2024-11-12
2	Петров Петр	2024-11-11	завершен	Блок питания	15000.00	2024-11-13
3	Сидорова Анна	2024-11-12	в процессе	Датчик движения	10000.00	2024-11-14
4	Кузнецов Олег	2024-11-13	завершен	Камера видеонаблюдения	20000.00	2024-11-15
5	Смирнова Мария	2024-11-14	в процессе	Коммутатор	15000.00	2024-11-16
6	Федоров Алексей	2024-11-15	отменен	Блок питания	5000.00	2024-11-17
7	Михайлова Оксана	2024-11-16	завершен	Камера видеонаблюдения	35000.00	2024-11-18

Рисунок 16 – Статус заказов и платежей для клиентов

employee\_tasks [postgres@localhost [21]]

WHERE ORDER BY

	full_name	description	due_date	creation_date	client_name
1	Сидоров Андрей	Установка камер в офисе	2024-11-15	2024-11-10	Иванов Иван
2	Сидоров Андрей	Установка датчиков движения	2024-11-22	2024-11-10	Иванов Иван
3	Зайцев Максим	Настройка системы видеонаблюдения	2024-11-18	2024-11-12	Сидорова Анна
4	Морозова Наталья	Тестирование системы	2024-11-23	2024-11-12	Сидорова Анна
5	Васильева Ольга	Проверка подключения	2024-11-21	2024-11-14	Смирнова Мария

5 rows

Рисунок 17 – Текущие задачи сотрудников

warehouse\_equipment [postgres@localhost [21]]

WHERE ORDER BY

	location	equipment_type	status	equipment_count
1	Склад на ул. Кирова	Датчик движения	в ремонте	1
2	Склад на ул. Кирова	Камера видеонаблюдения	установлено	2
3	Склад на ул. Ленина	Блок питания	доступно	1
4	Склад на ул. Ленина	Камера видеонаблюдения	установлено	1
5	Склад на ул. Ленина	Коммутатор	доступно	1
6	Склад на ул. Советская	Блок питания	установлено	1

6 rows

Рисунок 18 – Запасы оборудования на складах

## 3 ОБЕСПЕЧЕНИЕ ЗАЩИТЫ БАЗЫ ДАННЫХ

### 3.1 Задачи по мониторингу БД

#### Листинг 10 – Создание таблицы-лога

```
CREATE TABLE public.audit_log
(
    log_id          SERIAL PRIMARY KEY,
    operation       VARCHAR(50),           -- тип операции: INSERT,
UPDATE, DELETE
    table_name     VARCHAR(255),           -- имя таблицы, в
которой произошли изменения
    changed_data   JSONB,                 -- старые и новые
значения в формате JSON
    changed_by     VARCHAR(50),           -- роль, которая
произвела изменение
    timestamp      TIMESTAMP DEFAULT CURRENT_TIMESTAMP -- время изменения
);
```

#### Листинг 11 – Создание функции-триггера

```
CREATE OR REPLACE FUNCTION log_changes() RETURNS TRIGGER AS
$$
BEGIN
    IF TG_OP = 'INSERT' THEN
        INSERT INTO public.audit_log (operation, table_name, changed_data,
changed_by)
        VALUES ('INSERT', TG_TABLE_NAME, row_to_json(NEW), current_user);
    ELSIF TG_OP = 'UPDATE' THEN
        INSERT INTO public.audit_log (operation, table_name, changed_data,
changed_by)
        VALUES ('UPDATE', TG_TABLE_NAME, jsonb_build_object('old',
row_to_json(OLD), 'new', row_to_json(NEW)),
current_user);
    ELSIF TG_OP = 'DELETE' THEN
        INSERT INTO public.audit_log (operation, table_name, changed_data,
changed_by)
        VALUES ('DELETE', TG_TABLE_NAME, row_to_json(OLD), current_user);
    END IF;
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;
```

#### Листинг 12 – Создание триггеров для основных таблиц

```
-- Триггер для таблицы Клиентов
CREATE TRIGGER client_changes
    AFTER INSERT OR UPDATE OR DELETE
    ON public.client
    FOR EACH ROW
EXECUTE FUNCTION log_changes();

-- Триггер для таблицы Заказов
CREATE TRIGGER order_changes
    AFTER INSERT OR UPDATE OR DELETE
    ON public.order
    FOR EACH ROW
EXECUTE FUNCTION log_changes();
```

```

-- Триггер для таблицы Оборудования
CREATE TRIGGER equipment_changes
  AFTER INSERT OR UPDATE OR DELETE
  ON public.equipment
  FOR EACH ROW
EXECUTE FUNCTION log_changes();

-- Триггер для таблицы Сотрудников
CREATE TRIGGER employee_changes
  AFTER INSERT OR UPDATE OR DELETE
  ON public.employee
  FOR EACH ROW
EXECUTE FUNCTION log_changes();

-- Триггер для таблицы Задач
CREATE TRIGGER task_changes
  AFTER INSERT OR UPDATE OR DELETE
  ON public.task
  FOR EACH ROW
EXECUTE FUNCTION log_changes();

-- Триггер для таблицы Складов
CREATE TRIGGER warehouse_changes
  AFTER INSERT OR UPDATE OR DELETE
  ON public.warehouse
  FOR EACH ROW
EXECUTE FUNCTION log_changes();

-- Триггер для таблицы Платежей
CREATE TRIGGER payment_changes
  AFTER INSERT OR UPDATE OR DELETE
  ON public.payment
  FOR EACH ROW
EXECUTE FUNCTION log_changes();

```

После выполнения команд, данных выше, остается только проверить их работоспособность.

### Листинг 13 – Код для проверки работоспособности логов

```

-- Вставка в таблицу client
INSERT INTO public.client (full_name, phone, email, address,
responsible_employee)
VALUES ('Андреев Алексей', '89007776655', 'andreev@mail.ru', 'ул. Ленина, д.
10', 1);

-- Вставка в таблицу order
INSERT INTO public.order (creation_date, status, client_id, total_cost)
VALUES ('2024-11-22', 'в процессе', 1, 45000.00);

-- Вставка в таблицу equipment
INSERT INTO public.equipment (equipment_type, serial_number, status,
order_id, warehouse_id)
VALUES ('Камера видеонаблюдения', 'SN009', 'доступно', 1, 1);

-- Вставка в таблицу employee
INSERT INTO public.employee (full_name, position, phone, email)
VALUES ('Сергеев Сергей', 'Техник', '89998886677', 'sergeev@mail.ru');

-- Вставка в таблицу task
INSERT INTO public.task (description, due_date, order_id, employee_id)
VALUES ('Проверка монтажа оборудования', '2024-11-25', 1, 2);

```



```
-- Вставка в таблицу warehouse
INSERT INTO public.warehouse (location, responsible_employee)
VALUES ('Склад на ул. Центральная', 2);

-- Вставка в таблицу payment
INSERT INTO public.payment (amount, payment_date, order_id)
VALUES (25000.00, '2024-11-23', 1);

--

-- Обновление данных в таблице client
UPDATE public.client
SET phone = '89997776655', email = 'updated_andreev@mail.ru'
WHERE client_id = 8;

-- Обновление данных в таблице order
UPDATE public.order
SET status = 'завершен', total_cost = 50000.00
WHERE order_id = 8;

-- Обновление данных в таблице equipment
UPDATE public.equipment
SET status = 'установлено', order_id = 2
WHERE equipment_id = 8;

-- Обновление данных в таблице employee
UPDATE public.employee
SET position = 'Старший техник', phone = '89999998888'
WHERE employee_id = 8;

-- Обновление данных в таблице task
UPDATE public.task
SET due_date = '2024-11-30', description = 'Окончательная проверка монтажа
оборудования'
WHERE task_id = 8;

-- Обновление данных в таблице warehouse
UPDATE public.warehouse
SET location = 'Склад на ул. Ленина', responsible_employee = 3
WHERE warehouse_id = 9;

-- Обновление данных в таблице payment
UPDATE public.payment
SET amount = 30000.00, payment_date = '2024-11-24'
WHERE payment_id = 8;

-- Удаление данных из таблицы client
DELETE FROM public.client
WHERE client_id = 8;

-- Удаление данных из таблицы order
DELETE FROM public.order
WHERE order_id = 8;

-- Удаление данных из таблицы equipment
DELETE FROM public.equipment
WHERE equipment_id = 8;

-- Удаление данных из таблицы employee
DELETE FROM public.employee
WHERE employee_id = 8;
```

```
-- Удаление данных из таблицы task
DELETE FROM public.task
WHERE task_id = 8;

-- Удаление данных из таблицы warehouse
DELETE FROM public.warehouse
WHERE warehouse_id = 9;

-- Удаление данных из таблицы payment
DELETE FROM public.payment
WHERE payment_id = 8;

-- Проверка содержимого таблицы логирования (audit_log)
SELECT *
FROM public.audit_log;
```

log_id	operation	table_name	changed_data	changed_by	timestamp
1	1 INSERT	client	{ "email": "andreev@mail.ru", "phone": "89007776655", "address": "ул. Центральная", "warehouse_id": 9 }	postgres	2024-11-24 16:05:46.370654
2	2 INSERT	order	{ "status": "в процессе", "order_id": 8, "client_id": 1, "total": 30000.00 }	postgres	2024-11-24 16:05:46.405919
3	3 INSERT	equipment	{ "status": "доступно", "order_id": 1, "equipment_id": 8, "warehouse_id": 9 }	postgres	2024-11-24 16:05:46.438098
4	4 INSERT	employee	{ "email": "sergeev@mail.ru", "phone": "89998886677", "position": "менеджер" }	postgres	2024-11-24 16:05:46.479675
5	5 INSERT	task	{ "task_id": 8, "due_date": "2024-11-25", "order_id": 1, "description": "задание" }	postgres	2024-11-24 16:05:46.506515
6	6 INSERT	warehouse	{ "location": "Склад на ул. Центральная", "warehouse_id": 9, "responsible": "Иванов" }	postgres	2024-11-24 16:05:46.527469
7	7 INSERT	payment	{ "amount": 25000.00, "order_id": 1, "payment_id": 8, "payer": "Иванов" }	postgres	2024-11-24 16:05:46.542568
8	8 UPDATE	client	{ "new": { "email": "updated_andreev@mail.ru", "phone": "89997776655" } }	postgres	2024-11-24 16:10:02.803780
9	9 UPDATE	order	{ "new": { "status": "завершен", "order_id": 8, "client_id": 1, "total": 30000.00 } }	postgres	2024-11-24 16:10:02.830822
10	10 UPDATE	equipment	{ "new": { "status": "установлено", "order_id": 2, "equipment_id": 8, "warehouse_id": 9 } }	postgres	2024-11-24 16:10:02.852012
11	11 UPDATE	employee	{ "new": { "email": "sergeev@mail.ru", "phone": "89999998888" } }	postgres	2024-11-24 16:10:02.895541
12	12 UPDATE	task	{ "new": { "task_id": 8, "due_date": "2024-11-30", "order_id": 1, "description": "задание" } }	postgres	2024-11-24 16:10:02.923669
13	13 UPDATE	warehouse	{ "new": { "location": "Склад на ул. Ленина", "warehouse_id": 9, "responsible": "Иванов" } }	postgres	2024-11-24 16:10:02.968160
14	14 UPDATE	payment	{ "new": { "amount": 30000.00, "order_id": 1, "payment_id": 8, "payer": "Иванов" } }	postgres	2024-11-24 16:10:03.001862
15	15 DELETE	client	{ "email": "updated_andreev@mail.ru", "phone": "89997776655", "address": "ул. Центральная", "warehouse_id": 9 }	postgres	2024-11-24 16:10:03.039237
16	16 DELETE	order	{ "status": "завершен", "order_id": 8, "client_id": 1, "total": 30000.00 }	postgres	2024-11-24 16:10:03.080925
17	17 DELETE	equipment	{ "status": "установлено", "order_id": 2, "equipment_id": 8, "warehouse_id": 9 }	postgres	2024-11-24 16:10:03.139332
18	18 DELETE	employee	{ "email": "sergeev@mail.ru", "phone": "89999998888", "position": "менеджер" }	postgres	2024-11-24 16:10:03.172970
19	19 DELETE	task	{ "task_id": 8, "due_date": "2024-11-30", "order_id": 1, "description": "задание" }	postgres	2024-11-24 16:10:03.229920
20	20 DELETE	warehouse	{ "location": "Склад на ул. Ленина", "warehouse_id": 9, "responsible": "Иванов" }	postgres	2024-11-24 16:10:03.264396
21	21 DELETE	payment	{ "amount": 30000.00, "order_id": 1, "payment_id": 8, "payer": "Иванов" }	postgres	2024-11-24 16:10:03.293676

Рисунок 19 – Содержимое таблицы логирования

## 3.2 Задачи по шифрованию данных

### Листинг 14 – Создание таблицы с секретными данными

```
CREATE TABLE public.secret_data
(
    secret_id SERIAL PRIMARY KEY,
    user_role VARCHAR(50),
    secret_token BYTEA -- токен в зашифрованном виде
);
```

### Листинг 15 – Добавление pgcrypto

```
CREATE EXTENSION IF NOT EXISTS pgcrypto;
```

## Листинг 16 –

```
-- Пример использования ключа "!stroNgpsw31234" для шифрования токена
"my_secure_token"
INSERT INTO public.secret_data (user_role, secret_token)
VALUES ('admin', pgp_sym_encrypt('my_secure_token', '!stroNgpsw31234'));

-- Дешифровка токена
SELECT secret_id, user_role, pgp_sym_decrypt(secret_token, '!stroNgpsw31234')
AS decrypted_token
FROM public.secret_data;

-- Попытка доступа без ключа
SELECT * FROM public.secret_data;

-- Доступ с использованием правильного ключа
SELECT secret_id, user_role, pgp_sym_decrypt(secret_token, '!stroNgpsw31234')
AS decrypted_token
FROM public.secret_data;
```

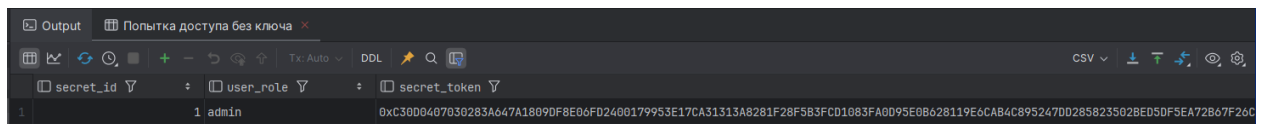


Рисунок 20 – Попытка обычного доступа без ключа

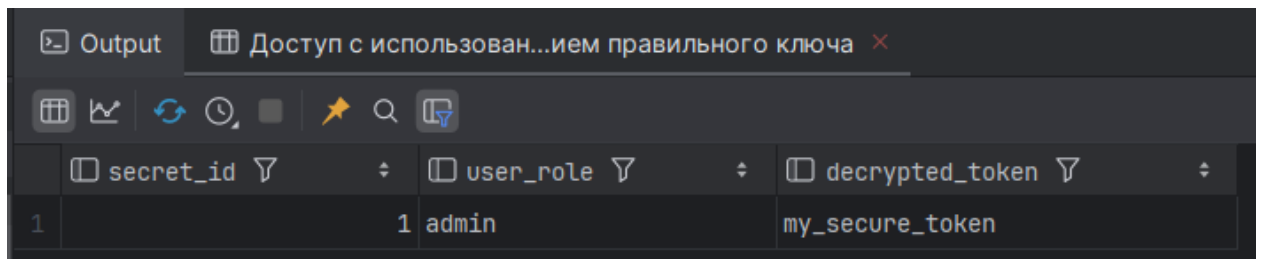


Рисунок 21 – Доступ с использованием ключа "!stroNgpsw31234"

### 3.3 Задачи по разграничению доступа в БД

#### Листинг 17 – Создание ролей для каждого из классов потребителей информации

```
-- Создание роли для менеджеров
CREATE ROLE manager LOGIN PASSWORD 'manager_password';

-- Создание роли для техников
CREATE ROLE technician LOGIN PASSWORD 'technician_password';

-- Создание роли для складских сотрудников
CREATE ROLE warehouse_staff LOGIN PASSWORD 'warehouse_password';
```

#### Листинг 18 – Определение набора привилегий по отношению к таблицам БД для каждой роли

```
-- Доступ к таблице клиентов (просмотр и редактирование)
```

```

GRANT SELECT, INSERT, UPDATE ON public.client TO manager;

-- Доступ к таблице заказов (просмотр и редактирование)
GRANT SELECT, INSERT, UPDATE ON public.order TO manager;

-- Доступ к представлению employee_tasks ограничен (только для чтения)
GRANT SELECT ON public.employee_tasks TO manager;

-- Ограничение доступа к логам (только суперпользователь)
REVOKE ALL ON public.audit_log FROM manager;

-- Доступ к таблице задач (просмотр и редактирование)
GRANT SELECT, INSERT, UPDATE ON public.task TO technician;

-- Доступ к таблице оборудования (просмотр и обновление)
GRANT SELECT, UPDATE ON public.equipment TO technician;

-- Доступ к таблице сотрудников для назначения ответственных лиц задачам
(только чтение)
GRANT SELECT ON public.employee TO technician;

-- Ограничение доступа к логам (только суперпользователь)
REVOKE ALL ON public.audit_log FROM technician;

-- Доступ к таблице складов (просмотр и редактирование)
GRANT SELECT, INSERT, UPDATE ON public.warehouse TO warehouse_staff;

-- Доступ к таблице оборудования (только просмотр)
GRANT SELECT ON public.equipment TO warehouse_staff;

-- Доступ к представлению warehouse_equipment для просмотра информации о
складе (только для чтения)
GRANT SELECT ON public.warehouse_equipment TO warehouse_staff;

-- Ограничение доступа к логам (только суперпользователь)
REVOKE ALL ON public.audit_log FROM warehouse_staff;

```

## Листинг 19 – Демонстрация работы системы разграничения доступа

```

-- Зайдем под ролью менеджера
SET ROLE manager;

-- Попробуем получить доступ к таблице клиентов (должно сработать)
SELECT * FROM public.client;

-- Попробуем получить доступ к таблице задач (должно вызвать ошибку)
SELECT * FROM public.task; -- Ожидается ошибка: permission denied

-- Попробуем получить доступ к таблице логирования (должно вызвать ошибку)
SELECT * FROM public.audit_log; -- Ожидается ошибка: permission denied

-- Зайдем под ролью техника
SET ROLE technician;

-- Попробуем получить доступ к таблице задач (должно сработать)
SELECT * FROM public.task;

-- Попробуем получить доступ к таблице клиентов (должно вызвать ошибку)
SELECT * FROM public.client; -- Ожидается ошибка: permission denied

-- Попробуем получить доступ к таблице логирования (должно вызвать ошибку)
SELECT * FROM public.audit_log; -- Ожидается ошибка: permission denied

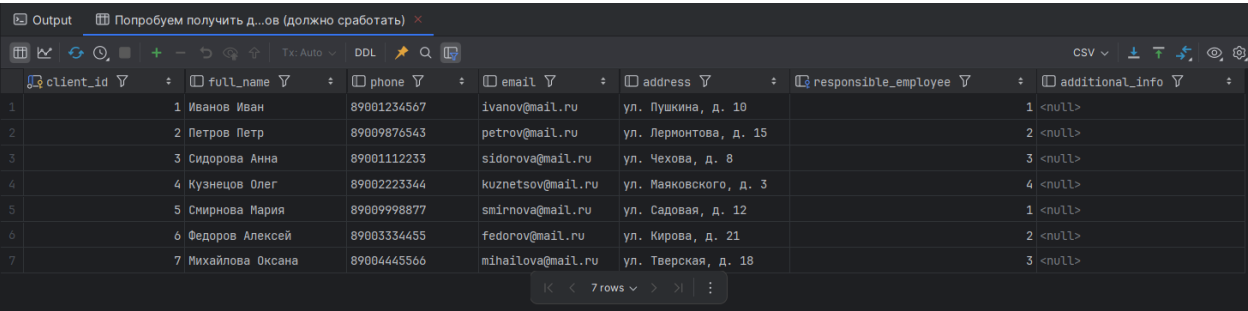
```

```
-- Зайдем под ролью сотрудника склада
SET ROLE warehouse_staff;

-- Попробуем получить доступ к таблице складов (должно сработать)
SELECT * FROM public.warehouse;

-- Попробуем получить доступ к таблице заказов (должно вызвать ошибку)
SELECT * FROM public.order; -- Ожидается ошибка: permission denied

-- Попробуем получить доступ к таблице логирования (должно вызвать ошибку)
SELECT * FROM public.audit_log; -- Ожидается ошибка: permission denied
```



	client_id	full_name	phone	email	address	responsible_employee	additional_info
1	1	Иванов Иван	89001234567	ivanov@mail.ru	ул. Пушкина, д. 10		1 <null>
2	2	Петров Петр	89009876543	petrov@mail.ru	ул. Лермонтова, д. 15		2 <null>
3	3	Сидорова Анна	89001112233	sidorova@mail.ru	ул. Чехова, д. 8		3 <null>
4	4	Кузнецов Олег	89002223344	kuznetsov@mail.ru	ул. Маяковского, д. 3		4 <null>
5	5	Смирнова Мария	89009998877	smirnova@mail.ru	ул. Садовая, д. 12		1 <null>
6	6	Федоров Алексей	89003334455	fedorov@mail.ru	ул. Кирова, д. 21		2 <null>
7	7	Михайлова Оксана	89004445566	mihaилоva@mail.ru	ул. Тверская, д. 18		3 <null>

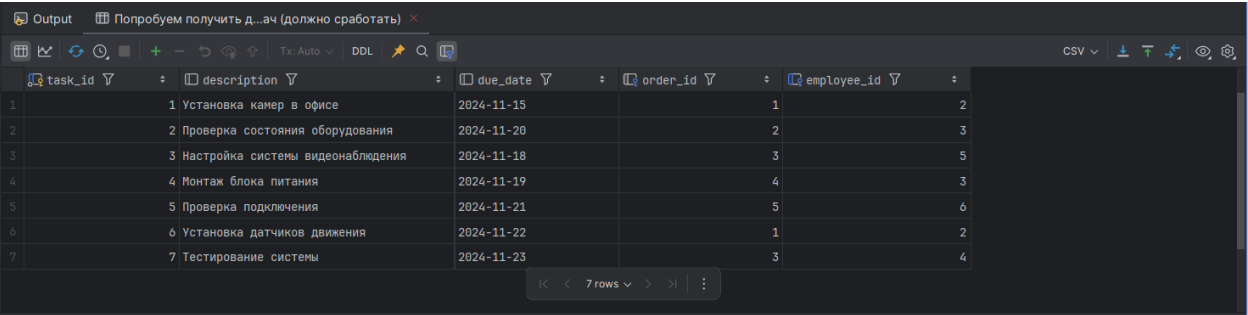
Рисунок 22 – Получение доступа к таблице клиентов под ролью менеджера



Рисунок 23 – Попытка получения доступа к таблице с задачами под ролью менеджера



Рисунок 24 – Попытка получения доступа к таблице-логу под ролью менеджера



	task_id	description	due_date	order_id	employee_id	
1	1	Установка камер в офисе	2024-11-15		1	2
2	2	Проверка состояния оборудования	2024-11-20		2	3
3	3	Настройка системы видеонаблюдения	2024-11-18		3	5
4	4	Монтаж блока питания	2024-11-19		4	3
5	5	Проверка подключения	2024-11-21		5	6
6	6	Установка датчиков движения	2024-11-22		1	2
7	7	Тестирование системы	2024-11-23		3	4

Рисунок 25 – Получение доступа к таблице задач под ролью техника



Рисунок 26 – Попытка получения доступа к таблице клиентов под ролью техника



Рисунок 27 – Попытка получения доступа к таблице-логу под ролью техника

Output    Попробуем получить д...ов (должно сработать) ✕

Tx: Auto ▾    DDL         

	warehouse_id ▾	location ▾	responsible_employee ▾
1		1 Склад на ул. Ленина	4
2		2 Склад на ул. Кирова	1
3		3 Склад на ул. Советская	2
4		4 Склад на ул. Гагарина	5
5		5 Склад на ул. Жукова	3
6		6 Склад на ул. Пушкина	6
7		7 Склад на ул. Дружбы	1
8		8 Склад на ул. Победы	

8 rows ▾

Рисунок 28 – Получение доступа к таблице со складами под ролью сотрудника склада

[42501] ОШИБКА: нет доступа к таблице order

Рисунок 29 – Попытка получения доступа к таблице заказов под ролью сотрудника склада

[42501] ОШИБКА: нет доступа к таблице audit\_log

Рисунок 30 – Попытка получения доступа к таблице-логу под ролью сотрудника склада

## **4 РЕАЛИЗАЦИЯ СЕРВИСА ДЛЯ ВЗАИМОДЕЙСТВИЯ С РАЗРАБОТАННОЙ БАЗОЙ ДАННЫХ**

### **4.1 Выбор средств разработки**

Для реализации проекта были использованы средства разработки, подходящие как для серверной, так и для клиентской части. Эти инструменты обеспечивают высокую производительность, удобство разработки и масштабируемость системы.

#### **4.1.1 Серверная часть (Backend)**

Для реализации проекта были использованы средства разработки, обеспечивающие производительность, удобство разработки и масштабируемость системы. Особое внимание уделено выбору инструментов для серверной и клиентской части, а также их интеграции.

- Язык программирования:
  - Python — используется для разработки серверной части. Python обеспечивает гибкость, позволяет быстро разрабатывать REST API и интегрироваться с базами данных.
- Фреймворк для серверной части:
  - FastAPI— высокопроизводительный фреймворк на Python для создания API. Его преимущества:
  - Автоматическая генерация документации (OpenAPI);
  - Простота написания асинхронного кода;
  - Удобная работа с запросами и ответами.
- Библиотека для работы с базой данных:
  - `psycopg2` — используется для взаимодействия с реляционной базой данных PostgreSQL.
- База данных:
  - PostgreSQL — мощная реляционная база данных, обеспечивающая стабильную работу системы.
- ORM для работы с базой данных:
  - SQLAlchemy — объектно-реляционная модель, которая упрощает взаимодействие с базой данных PostgreSQL. Использование ORM позволяет избежать написания большого количества SQL-запросов, предоставляя удобный интерфейс для работы с данными.
- Контейнеризация:

- Docker — применяется для создания изолированных контейнеров для серверной части, базы данных и других компонентов приложения.
  - docker-compose — используется для упрощения настройки и запуска нескольких связанных контейнеров.
- Веб-сервер:
  - Nginx — используется в качестве обратного прокси-сервера для управления запросами между клиентской и серверной частями. Он обеспечивает балансировку нагрузки, ускоряет доставку статических файлов и повышает общую производительность системы.

#### **4.1.2 Клиентская часть (Frontend)**

- Фреймворк для клиентской части:
  - Next.js — современный React-фреймворк для серверного рендеринга и разработки динамических веб-приложений. Он обеспечивает высокую производительность и улучшенный SEO.
- Язык программирования и стилизация:
  - TypeScript — используется для строгой типизации компонентов, что упрощает разработку и снижает вероятность ошибок.
  - Tailwind CSS — применяется для стилизации пользовательского интерфейса. Tailwind CSS позволяет быстро и эффективно создавать адаптивный и минималистичный дизайн.
- Сборка и управление зависимостями:
  - npm — инструмент для быстрого управления зависимостями. Он обеспечивает экономию места за счет уникальной структуры хранения пакетов и повышает производительность сборки.
- Библиотека для взаимодействия с API:
  - Axios — используется для выполнения HTTP-запросов к серверной части и получения данных.

#### **4.1.3 Взаимодействие Frontend и Backend**

Клиентская часть (frontend) взаимодействует с серверной частью (backend) через REST API, разработанный с использованием FastAPI. Обмен данными происходит по HTTP-запросам, что позволяет динамически обновлять информацию и обрабатывать ввод данных пользователей.



Выбор этих технологий был сделан с учетом современных стандартов разработки, их популярности, надежности и удобства интеграции. Использование указанных инструментов позволило создать стабильное, производительное и масштабируемое приложение.

## 4.2 Взаимодействие сервиса и базы данных

Взаимодействие между сервисом (серверной частью) и базой данных организовано с использованием современных подходов, которые обеспечивают надежность, производительность и масштабируемость.

Сервис построен на основе модели клиент-сервер, где серверная часть отвечает за обработку запросов и взаимодействие с базой данных. Клиент отправляет запросы через REST API, а серверная часть передает эти запросы в базу данных, преобразуя их в SQL-запросы.

Для упрощения взаимодействия с базой данных используется SQLAlchemy. ORM позволяет работать с базой данных на уровне объектов, что значительно ускоряет разработку, делает код читаемым и минимизирует вероятность ошибок, связанных с SQL-запросами.

PostgreSQL используется в качестве основного хранилища данных. Она выбрана за надежность, производительность и поддержку сложных запросов, необходимых для реализации функциональности сервиса.

SQLAlchemy управляет пулом подключений, что позволяет эффективно обрабатывать множественные запросы без создания новых соединений каждый раз. Ниже приведен код подключения базы данных в SQLAlchemy. За них отвечает database.py.

Листинг 20 – database.py

```
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

DATABASE_URL = "postgresql://postgres:password@db:5432/company_services_db"

engine = create_engine(DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()

def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

## Листинг 21 – models.py

```
from sqlalchemy import Column, ForeignKey, Integer, String, Text, Date,
DECIMAL, create_engine, Boolean, TIMESTAMP, \
    JSON, text
from sqlalchemy.orm import declarative_base, relationship

Base = declarative_base()

class Employee(Base):
    __tablename__ = "employees"
    employee_id = Column(Integer, primary_key=True, autoincrement=True)
    full_name = Column(String(255), nullable=False)
    position = Column(String(100))
    phone = Column(String(50))
    email = Column(String(255))

    clients = relationship("Client",
        back_populates="responsible_employee_relation")
    warehouses = relationship("Warehouse",
        back_populates="responsible_employee_relation")
    tasks = relationship("Task", back_populates="employee_relation")

class Client(Base):
    __tablename__ = "clients"
    client_id = Column(Integer, primary_key=True, autoincrement=True)
    full_name = Column(String(255), nullable=False)
    phone = Column(String(50))
    email = Column(String(255))
    address = Column(Text)
    responsible_employee = Column(Integer,
        ForeignKey("employees.employee_id"))

    responsible_employee_relation = relationship("Employee",
        back_populates="clients")
    orders = relationship("Order", back_populates="client_relation")

class Order(Base):
    __tablename__ = "orders"
    order_id = Column(Integer, primary_key=True, autoincrement=True)
    creation_date = Column(Date)
    status = Column(String(50))
    client_id = Column(Integer, ForeignKey("clients.client_id"))
    total_cost = Column(DECIMAL(10, 2))

    client_relation = relationship("Client", back_populates="orders")
    equipments = relationship("Equipment", back_populates="order_relation")
    tasks = relationship("Task", back_populates="order_relation")
    payments = relationship("Payment", back_populates="order_relation")

class Warehouse(Base):
    __tablename__ = "warehouses"
    warehouse_id = Column(Integer, primary_key=True, autoincrement=True)
    location = Column(Text)
    responsible_employee = Column(Integer,
        ForeignKey("employees.employee_id"))
```

```

    responsible_employee_relation = relationship("Employee",
back_populates="warehouses")
    equipments = relationship("Equipment",
back_populates="warehouse_relation")

class Equipment(Base):
    __tablename__ = "equipments"
    equipment_id = Column(Integer, primary_key=True, autoincrement=True)
    equipment_type = Column(String(255))
    serial_number = Column(String(255))
    status = Column(String(50))
    order_id = Column(Integer, ForeignKey("orders.order_id"))
    warehouse_id = Column(Integer, ForeignKey("warehouses.warehouse_id"))

    order_relation = relationship("Order", back_populates="equipments")
    warehouse_relation = relationship("Warehouse",
back_populates="equipments")

class Task(Base):
    __tablename__ = "tasks"
    task_id = Column(Integer, primary_key=True, autoincrement=True)
    description = Column(Text)
    due_date = Column(Date)
    order_id = Column(Integer, ForeignKey("orders.order_id"))
    employee_id = Column(Integer, ForeignKey("employees.employee_id"))

    order_relation = relationship("Order", back_populates="tasks")
    employee_relation = relationship("Employee", back_populates="tasks")

class Payment(Base):
    __tablename__ = "payments"
    payment_id = Column(Integer, primary_key=True, autoincrement=True)
    amount = Column(DECIMAL(10, 2))
    payment_date = Column(Date)
    order_id = Column(Integer, ForeignKey("orders.order_id"))

    order_relation = relationship("Order", back_populates="payments")

class User(Base):
    __tablename__ = "users"
    id = Column(Integer, primary_key=True, index=True)
    full_name = Column(String, nullable=False)
    username = Column(String, unique=True, index=True, nullable=False)
    hashed_password = Column(String, nullable=False)
    is_active = Column(Boolean, default=True)
    role = Column(String, default="user", nullable=False)

class AuditLog(Base):
    __tablename__ = "audit_log"

    log_id = Column(Integer, primary_key=True, autoincrement=True)
    operation = Column(String(50), nullable=False)
    table_name = Column(String(255), nullable=False)
    changed_data = Column(JSON, nullable=True)
    changed_by = Column(String(50), nullable=False)
    timestamp = Column(TIMESTAMP, nullable=False,
server_default=text("CURRENT_TIMESTAMP"))

```

Основные этапы взаимодействия:

- Пользователь отправляет HTTP-запрос к API сервиса (например, создание, чтение, обновление или удаление данных);
- Серверная часть обрабатывает запрос, проверяет входные данные с использованием валидаторов, таких как Pydantic;
- Если запрос прошел проверку, серверная часть использует SQLAlchemy для преобразования запроса в SQL;
- Запрос передается в PostgreSQL, где он выполняется;
- PostgreSQL возвращает данные в ответ на запрос;
- Эти данные обрабатываются серверной частью, преобразуются в JSON-формат и возвращаются клиенту.

FastAPI поддерживает асинхронное программирование, что позволяет выполнять несколько операций взаимодействия с базой данных одновременно, не блокируя выполнение других задач. Такой подход обеспечивает стабильную и эффективную работу сервиса, а также удобство в его дальнейшем развитии.

За обработку данных серверной частью отвечает файл `main.py` (листинг 22), который содержит настройки эндпоинтов для файла `crud.py` (листинг 23), содержащего логику взаимодействия с сущностями в БД.

Листинг 22 – `main.py`

```
from typing import List

from fastapi import FastAPI, Depends, status, HTTPException, APIRouter
from sqlalchemy.orm import Session

from . import models, schemas, crud
from .database import engine, Base, get_db
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm
from .auth import create_access_token, get_current_user, get_password_hash,
verify_password
from fastapi.middleware.cors import CORSMiddleware

from .schemas import ClientOrdersPayments, WarehouseEquipment, EmployeeTasks
from .models import User

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")
app = FastAPI()

api_router = APIRouter()

app.add_middleware(
    CORSMiddleware,
    allow_origins=["http://localhost:3000"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

```

Base.metadata.create_all(bind=engine)

def role_required(allowed_roles: list[str]):
    def role_checker(current_user: User = Depends(get_current_user)):
        print(current_user.role)
        if current_user.role not in allowed_roles:
            raise HTTPException(
                status_code=status.HTTP_403_FORBIDDEN,
                detail="You do not have the required permissions."
            )
        return current_user
    return role_checker

def setup_crud_routes(app, name, model, schema, schema_create,
allowed_roles=None):
    allowed_roles = allowed_roles or ["manager", "admin", "technician",
"warehouse_staff"]

    @app.post(f"/{name}/", response_model=schema)
    def create_item(
        item: schema_create,
        db: Session = Depends(get_db),
        current_user: User = Depends(role_required(allowed_roles))
    ):
        return crud.create_entity(db=db, model=model, schema=item)

    @app.get(f"/{name}/", response_model=list[schema])
    def read_items(
        skip: int = 0,
        limit: int = 10,
        db: Session = Depends(get_db),
        current_user: User = Depends(role_required(allowed_roles))
    ):
        return crud.get_entities(db=db, model=model, skip=skip, limit=limit)

    @app.get(f"/{name}/{item_id}", response_model=schema)
    def read_item(
        item_id: int,
        db: Session = Depends(get_db),
        current_user: User = Depends(role_required(allowed_roles))
    ):
        entity = crud.get_entity(db=db, model=model, entity_id=item_id)
        if not entity:
            raise HTTPException(status_code=404, detail=f"{name.capitalize()}
not found")
        return entity

    @app.put(f"/{name}/{item_id}", response_model=schema)
    def update_item(
        item_id: int,
        item: schema_create,
        db: Session = Depends(get_db),
        current_user: User = Depends(role_required(allowed_roles))
    ):
        updated = crud.update_entity(db=db, model=model, entity_id=item_id,
schema=item)
        if not updated:
            raise HTTPException(status_code=404, detail=f"{name.capitalize()}
not found")
        return updated

```

```

@app.delete(f"/{name}/{item_id}")
def delete_item(
    item_id: int,
    db: Session = Depends(get_db),
    current_user: User = Depends(role_required(allowed_roles))
):
    deleted = crud.delete_entity(db=db, model=model, entity_id=item_id)
    if not deleted:
        raise HTTPException(status_code=404, detail=f"{name.capitalize()}
not found")
    return {"message": f"{name.capitalize()} deleted successfully"}

```

```

setup_crud_routes(app, "clients", models.Client, schemas.ClientRead,
schemas.ClientCreate, ["manager", "admin"])
setup_crud_routes(app, "orders", models.Order, schemas.OrderRead,
schemas.OrderCreate, ["manager", "admin", "technician"])
setup_crud_routes(app, "employees", models.Employee, schemas.EmployeeRead,
schemas.EmployeeCreate, ["manager", "admin"])
setup_crud_routes(app, "equipment", models.Equipment, schemas.EquipmentRead,
schemas.EquipmentCreate, ["technician", "warehouse_staff", "manager",
"admin"])
setup_crud_routes(app, "tasks", models.Task, schemas.TaskRead,
schemas.TaskCreate, ["technician", "manager", "admin"])
setup_crud_routes(app, "warehouses", models.Warehouse, schemas.WarehouseRead,
schemas.WarehouseCreate, ["warehouse_staff", "manager", "admin"])
setup_crud_routes(app, "payments", models.Payment, schemas.PaymentRead,
schemas.PaymentCreate, ["manager", "admin"])
setup_crud_routes(app, "users", models.User, schemas.UserRead,
schemas.UserCreate, ["manager", "admin"])
setup_crud_routes(app, "audit_log", models.AuditLog, schemas.AuditLogRead,
schemas.AuditLogCreate, ["manager", "admin"])

```

```

@app.get("/client-orders-payments",
response_model=List[ClientOrdersPayments])
def get_client_orders_payments(db: Session = Depends(get_db)):
    result = db.execute("""
        SELECT full_name, creation_date, status, equipment_type, amount,
payment_date
        FROM public.client_orders_payments
    """).fetchall()
    return [dict(row) for row in result]

```

```

@app.get("/employee-tasks", response_model=List[EmployeeTasks])
def get_employee_tasks(db: Session = Depends(get_db)):
    result = db.execute("""
        SELECT full_name, description, due_date, creation_date, client_name
        FROM public.employee_tasks
    """).fetchall()
    return [dict(row) for row in result]

```

```

@app.get("/warehouse-equipment", response_model=List[WarehouseEquipment])
def get_warehouse_equipment(db: Session = Depends(get_db)):
    result = db.execute("""
        SELECT location, equipment_type, status, equipment_count
        FROM public.warehouse_equipment
    """).fetchall()
    return [dict(row) for row in result]

```

```

# Регистрация пользователя
@app.post("/register", response_model=schemas.UserRead)
def register_user(user: schemas.UserCreate, db: Session = Depends(get_db)):
    print(user)
    existing_user = crud.get_user_by_username(db, username=user.username)
    if existing_user:
        raise HTTPException(status_code=400, detail="Username already
registered")
    return crud.create_user(db, user=user)

@app.get("/register")
def register_page(db: Session = Depends(get_db)):
    return {"message": "Register"}

# Получение токена
@app.post("/login")
def login(form_data: OAuth2PasswordRequestForm = Depends(), db: Session =
Depends(get_db)):
    print(form_data)
    user = crud.get_user_by_username(db, username=form_data.username)
    if not user or not verify_password(form_data.password,
user.hashed_password):
        raise HTTPException(status_code=401, detail="Incorrect username or
password")
    print(f"user:{user.username}\nrole:{user.role}")
    access_token = create_access_token(data={"sub": user.username, "role":
user.role})
    return {"access_token": access_token}

# Защищённый эндпоинт
@app.get("/me/", response_model=schemas.UserRead)
def read_users_me(current_user: schemas.UserRead =
Depends(get_current_user)):
    return current_user

@app.get("/get_user_by_username")
def get_user_by_username(username: str, db: Session = Depends(get_db)):
    user = crud.get_user_by_username(db, username=username)
    if not user:
        raise HTTPException(status_code=404, detail="User not found")
    return {
        "id": user.id,
        "full_name": user.full_name,
        "username": user.username,
        "is_active": user.is_active,
        "role": user.role
    }

```

### Листинг 23 – crud.py

```

from sqlalchemy.orm import Session
from . import models, schemas
from .auth import get_password_hash

# CRUD для универсальной сущности
def get_entity(db: Session, model, entity_id: int):
    return db.query(model).filter(model.id == entity_id).first()

def get_entities(db: Session, model, skip: int = 0, limit: int = 10):

```

```

        return db.query(model).offset(skip).limit(limit).all()

def create_entity(db: Session, model, schema):
    db_entity = model(**schema.dict())
    db.add(db_entity)
    db.commit()
    db.refresh(db_entity)
    return db_entity

def update_entity(db: Session, model, entity_id: int, schema):
    db_entity = db.query(model).filter(model.id == entity_id).first()
    if not db_entity:
        return None
    for key, value in schema.dict().items():
        setattr(db_entity, key, value)
    db.commit()
    db.refresh(db_entity)
    return db_entity

def delete_entity(db: Session, model, entity_id: int):
    db_entity = db.query(model).filter(model.id == entity_id).first()
    if db_entity:
        db.delete(db_entity)
        db.commit()
    return db_entity

# Пользователи
def get_user_by_username(db: Session, username: str):
    return db.query(models.User).filter(models.User.username ==
username).first()

def create_user(db: Session, user: schemas.UserCreate):
    hashed_password = get_password_hash(user.password)
    db_user = models.User(
        full_name=user.full_name,
        username=user.username,
        hashed_password=hashed_password,
        is_active=True,
        role="user"
    )
    db.add(db_user)
    db.commit()
    db.refresh(db_user)
    return db_user

```

### 4.3 Функционал сервиса

Сервис предоставляет широкий спектр функциональных возможностей для управления сотрудниками, клиентами, заказами, складами и оборудованием, а также для отслеживания задач, платежей и изменений в системе. Рассмотрим основные аспекты функциональности, которые реализованы в системе.



### **4.3.1 Авторизация**

Авторизация в системе обеспечивает доступ пользователей к функциональности сервиса в зависимости от их роли. Это ключевой компонент безопасности и удобства работы, который позволяет разграничить права доступа и предотвратить несанкционированные действия.

Функциональные особенности

#### **4.3.1.1 Вход в учетную запись:**

Пользователь вводит свои учетные данные (логин и пароль) на странице авторизации.

Система проверяет корректность введенных данных, сверяя их с записями в базе данных.

В случае успешной авторизации пользователь получает токен доступа, который используется для подтверждения его личности при выполнении дальнейших операций.

#### **4.3.1.2 Распределение ролей:**

Система поддерживает несколько ролей пользователей, таких как:

Администратор: полный доступ ко всем функциям системы, включая управление пользователями, настройку базы данных и просмотр журналов аудита.

Сотрудник: доступ только к своим задачам, складам, клиентам и заказам.

Клиент: доступ к информации о собственных заказах и платежах.

Роль пользователя определяется на этапе регистрации и хранится в базе данных.

#### **4.3.1.3 Разграничение доступа:**

После авторизации система проверяет роль пользователя и открывает доступ только к функциям и таблицам, разрешенным для данной роли.

Например, клиент не может просматривать задачи сотрудников или складские остатки, а сотрудник не имеет доступа к настройкам учетных записей.

#### **4.3.1.4 Безопасность:**

Пароли хранятся в зашифрованном виде (хэшированы) с использованием современных криптографических алгоритмов, что предотвращает их утечку в случае компрометации базы данных.

Сессии авторизации ограничены по времени, а токены доступа регулярно обновляются для предотвращения их использования злоумышленниками.

На стороне клиента за авторизацию и регистрацию в системе отвечают следующие скрипты:

- [...nextauth].ts: оболочка для провайдера авторизации nextAuth.js;
- auth/login.tsx:
- auth/register.tsx:

#### Листинг 24 – [...nextauth].ts

```
import NextAuth, { NextAuthOptions } from "next-auth";
import CredentialsProvider from "next-auth/providers/credentials";
import axios from "axios";
import { jwtDecode } from "jwt-decode";

import api from "@src/utils/api";

type DecodedToken = {
  sub: string;
  exp: number;
};

export const authOptions: NextAuthOptions = {
  providers: [
    CredentialsProvider({
      name: "Credentials",
      credentials: {
        username: { label: "Username", type: "text" },
        password: { label: "Password", type: "password" },
      },
      async authorize(credentials) {
        console.log("Credentials received:", credentials);
        if (!credentials) {
          throw new Error("Missing credentials");
        }

        try {
          const response = await axios.post(
            "http://localhost/api/login",
            new URLSearchParams({
              username: credentials.username,
              password: credentials.password,
            }).toString(),
            {
              headers: { "Content-Type": "application/x-www-
form-urlencoded" },
            }
          );
          const user = response.data;

          if (user && user.access_token) {
            const decoded: DecodedToken =
              jwtDecode(user.access_token);
            console.log("Decoded token:", decoded);
            return {
              id: decoded.sub,
              access_token: user.access_token,
            };
          }
        }
      }
    })
  ],
}
```

```

        }
        console.log("Authorization response:", response.data);
        return null;
    } catch (error) {
        console.error("Authorization error:", error);
        return null;
    }
    },
  )),
],
session: {
  strategy: "jwt" as const,
},
pages: {
  signIn: "/api/login",
  error: "/auth/login"
},
secret: process.env.NEXTAUTH_SECRET,
callbacks: {
  async jwt({ token, user }: { token: any; user?: any }) {
    if (user) {
      token.accessToken = user.access_token;
      token.id = user.id;
    }
    return token;
  },
  async session({ session, token }: { session: any; token: any }) {
    session.user = {
      id: token.id,
      accessToken: token.accessToken,
    };
    return session;
  },
},
};

export default NextAuth(authOptions);

```

## Листинг 25 – auth/login.tsx

```

import { useState } from "react";
import { useRouter } from "next/router";
import { Button } from "@components/ui/button";
import { Input } from "@components/ui/input";
import { signIn } from "next-auth/react";
import { Card, CardContent, CardDescription, CardHeader, CardTitle } from
"@components/ui/card";
import Link from "next/link";
import { Label } from "@components/ui/label";

const LoginPage = () => {
  const [credentials, setCredentials] = useState({ username: "", password:
"" });
  const [error, setError] = useState<string | null>(null);
  const router = useRouter();

  const handleLogin = async () => {
    setError(null);
    const result = await signIn("credentials", {
      username: credentials.username,
      password: credentials.password,
      redirect: false,

```

```

    });
    console.log(credentials);
    if (result?.error) {
        setError(result.error);
    }

    await router.push("/dashboard");

};

return (
    <Card className="mx-auto max-w-sm my-auto">
        <CardHeader>
            <CardTitle className="text-2xl">Login</CardTitle>
            <CardDescription>
                Enter your email below to login to your account
            </CardDescription>
        </CardHeader>
        <CardContent>
            <div className="grid gap-4">
                <div className="grid gap-2">
                    <Label htmlFor="email">Email</Label>
                    <Input
                        placeholder="mail@example.com"
                        value={credentials.username}
                        required
                        onChange={(e) => setCredentials({...credentials,
username: e.target.value})}
                    />
                </div>
                <div className="grid gap-2">
                    <Input
                        placeholder="Password"
                        type="password"
                        required
                        value={credentials.password}
                        onChange={(e) => setCredentials({...credentials,
password: e.target.value})}
                    />
                </div>
                <div>
                    {error && <p className="text-red-500">{error}</p>}
                    <Button onClick={handleLogin}>Login</Button>
                </div>
                <div className="mt-4 text-center text-sm">
                    Don't have an account?{" "}
                    <Link href="/auth/register" className="underline">
                        Sign up
                    </Link>
                </div>
            </CardContent>
        </Card>
    );
};

export default LoginPage;

```

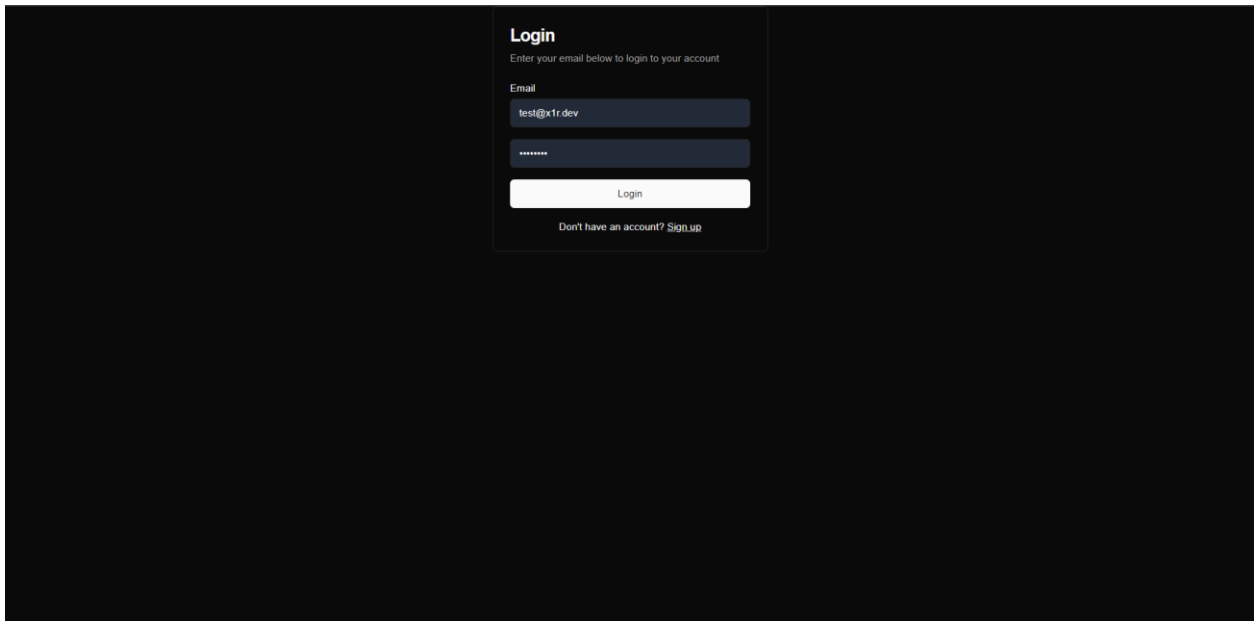


Рисунок 31 – Форма авторизации

#### Листинг 26 – auth/register.tsx

```
import { useState } from "react";
import { useRouter } from "next/router";
import { Button } from "@components/ui/button";
import { Input } from "@components/ui/input";
import { z } from "zod";
import { zodResolver } from "@hookform/resolvers/zod";
import { useForm } from "react-hook-form";
import axios from "axios";
import { Card, CardContent, CardDescription, CardHeader, CardTitle } from
"@components/ui/card";

// Схема валидации
const registrationSchema = z.object({
  full_name: z.string().min(3, "Full name must be at least 3 characters"),
  username: z.string().min(3, "Username must be at least 3 characters"),
  password: z.string().min(6, "Password must be at least 6 characters"),
});

type RegistrationFormInputs = z.infer<typeof registrationSchema>;

const RegisterPage = () => {
  const [error, setError] = useState<string | null>(null);
  const router = useRouter();

  const {
    register,
    handleSubmit,
    formState: { errors },
  } = useForm<RegistrationFormInputs>({
    resolver: zodResolver(registrationSchema),
  });

  const handleRegister = async (data: RegistrationFormInputs) => {
    setError(null);
    try {
      console.log(data);
    }
  }
}
```

```

        await axios.post("http://localhost/api/register", data);
        router.push("/auth/login"); // Перенаправляем на страницу логина
    } catch (err: any) {
        setError(err.response?.data?.detail || "Registration failed");
    }
};

return (
    <Card className="mx-auto max-w-sm my-auto">
        <CardHeader>
            <CardTitle className="text-2xl">Registration</CardTitle>
            <CardDescription>
                Enter data below to register your account
            </CardDescription>
        </CardHeader>
        <CardContent>
            <form onSubmit={handleSubmit(handleRegister)} className="grid
gap-4">
                <div>
                    <Input
                        placeholder="Full Name"
                        {...register("full_name")}
                    />
                    {errors.full_name && (
                        <p className="text-red-500 text-sm">{errors.full_name.message}</p>
                    )}
                </div>
                <div>
                    <Input
                        placeholder="Username"
                        {...register("username")}
                    />
                    {errors.username && (
                        <p className="text-red-500 text-sm">{errors.username.message}</p>
                    )}
                </div>
                <div>
                    <Input
                        placeholder="Password"
                        type="password"
                        {...register("password")}
                    />
                    {errors.password && (
                        <p className="text-red-500 text-sm">{errors.password.message}</p>
                    )}
                </div>
                {error && <p className="text-red-500">{error}</p>}
                <Button type="submit">Register</Button>
            </form>
        </CardContent>
    </Card>
);
};

export default RegisterPage;

```

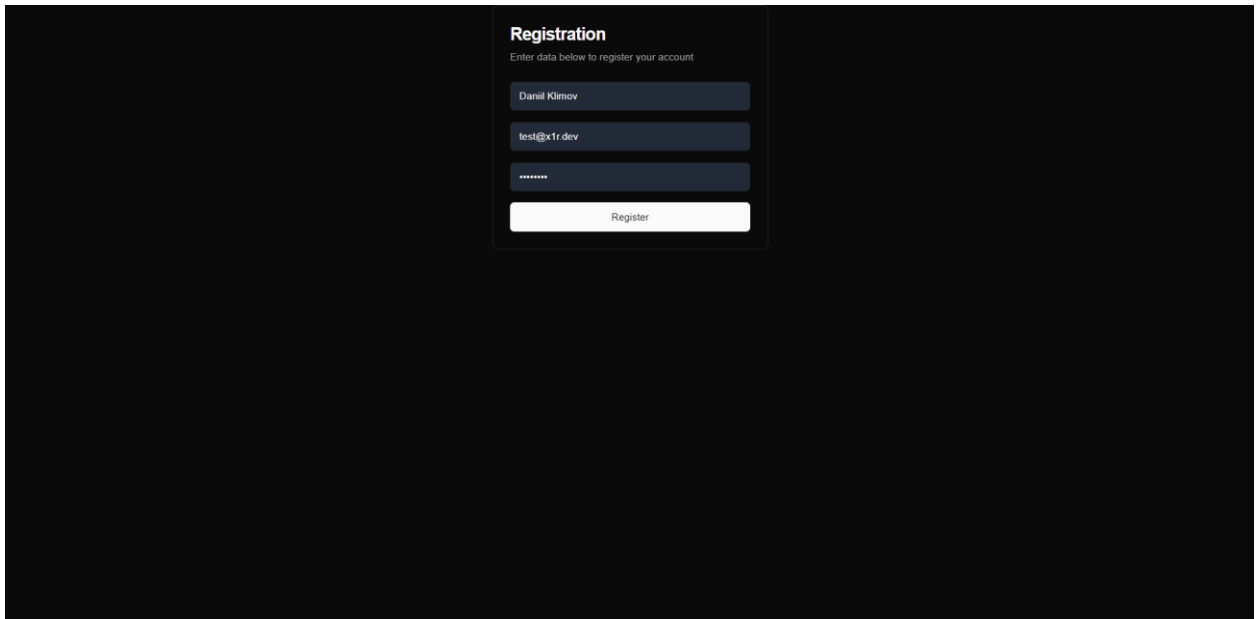


Рисунок 32 – Форма регистрации

Листинг 27 – pages/index.tsx

```
import Link from "next/link";
import { Button } from "@components/ui/button";
import { Card, CardContent, CardHeader, CardTitle } from
"@components/ui/card";

export default function Home() {
  return (
    <div className="flex min-h-screen items-center justify-center">
      <Card className="w-full max-w-md p-6">
        <CardHeader>
          <CardTitle className="text-center text-2xl font-semibold">
            Welcome to Our Platform
          </CardTitle>
        </CardHeader>
        <CardContent>
          <div className="flex flex-col space-y-4">
            <Button asChild className="w-full">
              <Link href="/auth/login">Login</Link>
            </Button>
            <Button variant="outline" asChild className="w-full">
              <Link href="/auth/register">Register</Link>
            </Button>
          </div>
        </CardContent>
      </Card>
    </div>
  );
}
```

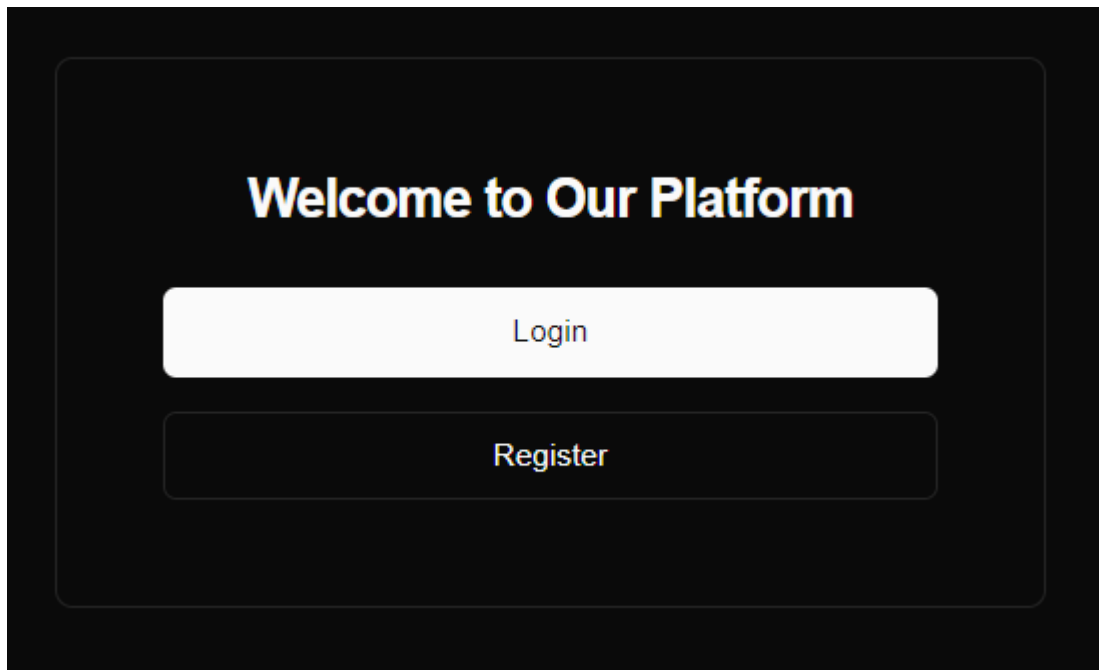


Рисунок 33 – Корневой каталог сайта с перенаправлением на авторизацию или регистрацию

Листинг 28 – pages/\_app.tsx

```
// src/pages/_app.tsx
import {httpBatchLink} from "@trpc/client/links/httpBatchLink";
import {loggerLink} from "@trpc/client/links/loggerLink";
import {withTRPC} from "@trpc/next";
import type {AppType} from "next/dist/shared/lib/utils";
import superjson from "superjson";
import type {AppRouter} from "../server/router";
import "../styles/globals.css";
import {ThemeProvider} from "@components/themes-provider";
import {Inter} from "next/font/google";
import {SessionProvider} from "next-auth/react";
import {AppProps} from "next/app";

const inter = Inter({subsets: ["latin"]});

const MyApp = ({Component, pageProps}: AppProps<{ session: any }>) => {
  return (
    <main>
      <ThemeProvider
        attribute="class"
        defaultTheme="dark"
        enableSystem
        disableTransitionOnChange
      >
        <SessionProvider session={pageProps.session}>
          <Component {...pageProps} />
        </SessionProvider>
      </ThemeProvider>
    </main>
  );
};

const getBaseUrl = () => {
```



```

    if (typeof window !== "undefined") return ""; // browser should use
    relative url
    if (process.env.VERCEL_URL) return `https://${process.env.VERCEL_URL}`;
    // SSR should use vercel url
    return `http://localhost:${process.env.PORT ?? 3000}`; // dev SSR should
    use localhost
  };

export default withTRPC<AppRouter>({
  config({ctx}) {

    const url = `${getBaseUrl()}/api/trpc`;

    return {
      links: [
        loggerLink({
          enabled: (opts) =>
            process.env.NODE_ENV === "development" ||
            (opts.direction === "down" && opts.result instanceof
Error),
        }),
        httpBatchLink({url}),
      ],
      url,
      transformer: superjson,
    };
  },

  ssr: false,
})(MyApp);

```

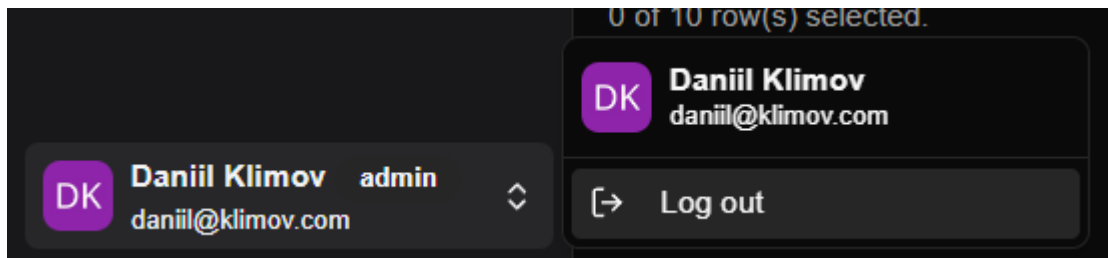


Рисунок 34 – Возможность выйти и сменить пользователя

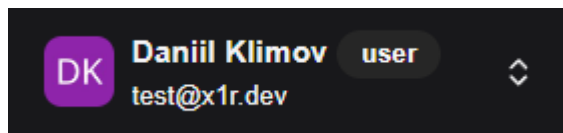


Рисунок 35 – При смене пользователя меняются и роли, они отображаются около имени пользователя

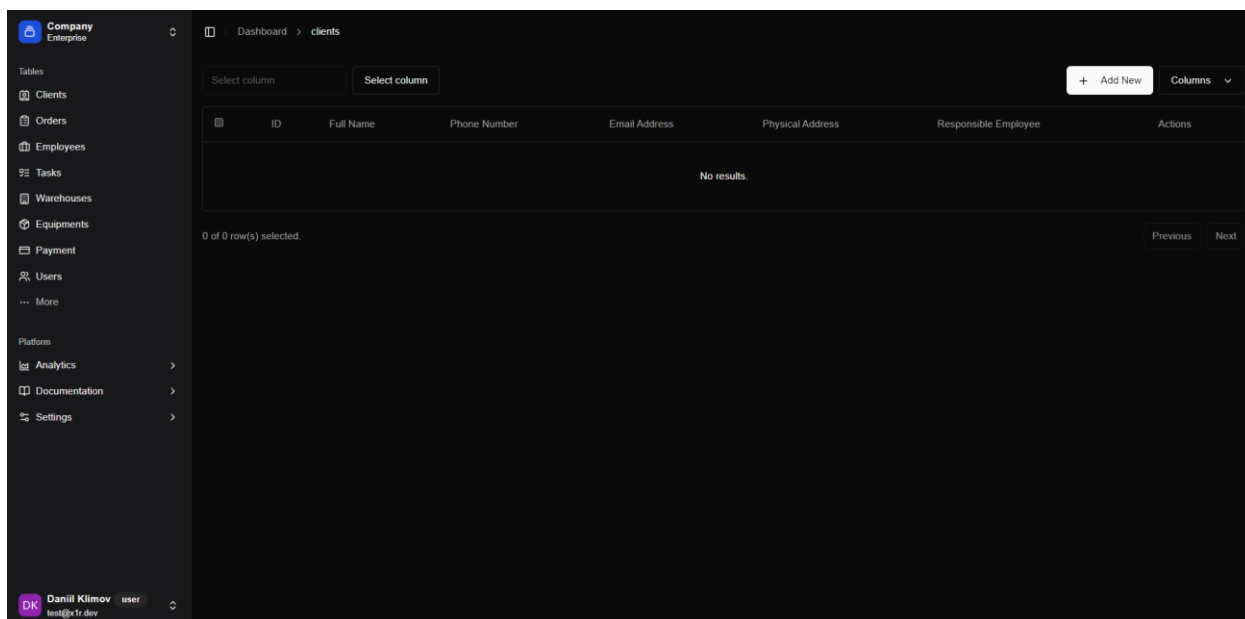


Рисунок 36 – При смене пользователя доступа к чтению таблицы «Клиенты» у роли «user» нет, поэтому записи не отображаются

### 4.3.2 Безопасный доступ к API

Безопасный доступ к API реализован через систему запросов с аутентификацией и авторизацией, что обеспечивает защиту данных от несанкционированного доступа. Для взаимодействия с сервером используется библиотека `axios`, которая позволяет создавать удобный и масштабируемый клиент для API. Все запросы отправляются на базовый URL через модуль `api.ts` (листинг 29), где автоматизирована работа с заголовками и параметрами. Перед отправкой каждого запроса к API в заголовок добавляется токен доступа (JWT), хранящийся в `localStorage`. Это гарантирует, что только авторизованные пользователи могут получить доступ к защищенным ресурсам.

Создан универсальный интерфейс для выполнения базовых операций (создание, чтение, обновление, удаление) на сервере. Это упрощает разработку новых функций и ускоряет внедрение изменений. Для управления данными используется система динамических компонентов, таких как `RecordDialog` (листинг 30) и `DataTable` (листинг 31). Они обеспечивают удобный пользовательский интерфейс для взаимодействия с API, включая фильтрацию, сортировку и управление данными.

Модуль `api.ts` является основой для всех запросов к серверу. `RecordDialog` отвечает за создание и редактирование записей в модальном окне, генерируя формы на основе структуры таблиц. `DashboardTable` (листинг 32) представляет собой интерфейс для

отображения данных в таблицах и взаимодействия с API для выполнения операций получения, обновления и удаления данных. DataTable(листинг 33) обеспечивает сортировку, фильтрацию, выбор колонок и строк, что делает его универсальным инструментом для работы с различными наборами данных. AppSidebar(листинг 34) используется для навигации по сайту, динамически адаптируясь к роли пользователя. Компонент [table].tsx(листинг 35) позволяет отображать данные для конкретной таблицы на основе URL-параметров, что делает его гибким и удобным для повторного использования.

Эти элементы работают совместно, формируя надежную и удобную систему взаимодействия клиента с сервером, обеспечивая безопасность, удобство работы и масштабируемость приложения.

#### Листинг 29 – api.ts

```
import axios from "axios";

const api = axios.create({
  baseURL: "http://localhost/api",
  headers: {
    "Content-Type": "application/json",
  },
});

export const CRUD = {
  getAll: async (endpoint: string, params = {}) => {
    const response = await api.get(`/${endpoint}/`, { params });
    return response.data;
  },
  getOne: async (endpoint: string, id: number) => {
    const response = await api.get(`/${endpoint}/${id}`);
    return response.data;
  },
  create: async (endpoint: string, data: any) => {
    const response = await api.post(`/${endpoint}/`, data);
    return response.data;
  },
  update: async (endpoint: string, id: number, data: any) => {
    const response = await api.put(`/${endpoint}/${id}`, data);
    return response.data;
  },
  delete: async (endpoint: string, id: number) => {
    const response = await api.delete(`/${endpoint}/${id}`);
    return response.data;
  },
};

api.interceptors.request.use((config) => {
  const token = localStorage.getItem("access_token");
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});
```

```
export default api;
```

### Листинг 30 – Record-dialog.tsx

```
import {useState} from "react";
import {Dialog, DialogHeader, DialogFooter, DialogContent, DialogTitle} from
"@/components/ui/dialog";
import {Input} from "@/components/ui/input";
import {baseColumns} from "@/lib/tables-columns";
import {Button} from "@/components/ui/button";
import {VisuallyHidden} from "@radix-ui/react-visually-hidden";

const RecordDialog = ({
    mode,
    table,
    data,
    onSubmit,
    onClose,
}): {
    mode: "add" | "edit";
    table: keyof typeof baseColumns;
    data: any;
    onSubmit: (table: any, data: any) => Promise<void>;
    onClose: () => void;
} => {
    const columns = baseColumns[table];
    const [formData, setFormData] = useState(data || {});

    const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {
        setFormData({...formData, [e.target.name]: e.target.value});
    };

    const handleSubmit = () => {
        onSubmit(table, formData);
        onClose();
    };

    return (
        <Dialog open={true}>
            <DialogContent>
                <VisuallyHidden>
                    <DialogTitle/>
                </VisuallyHidden>
                <DialogHeader>
                    {mode === "edit" ? `Edit ${table}` : `Add New ${table}`}
                </DialogHeader>
                <div className="space-y-4">
                    {columns
                        .filter((column) => column.header !== "ID")
                        .map((column) => (
                            <div key={column.accessorKey}>
                                <label htmlFor={column.accessorKey}
                                    className="block text-sm font-medium">
                                        {column.header}
                                    </label>
                                <Input
                                    id={column.accessorKey}
                                    name={column.accessorKey}
                                    placeholder={column.header as string}
                                    value={formData[column.accessorKey] ||
                                        ""}
                                    onChange={handleChange}
                                >
```

```

        />
      </div>
    )))
  </div>
  <DialogFooter>
    <Button variant={"outline"} className="btn btn-secondary"
onClick={onClose}>
      Cancel
    </Button>
    <Button className="btn btn-primary"
onClick={handleSubmit}>
      {mode === "edit" ? "Save Changes" : "Add Record"}
    </Button>
  </DialogFooter>
</DialogContent>
</Dialog>
);
};

export default RecordDialog;

```

### Листинг 31 – [table].tsx

```

import {useRouter} from "next/router";
import {useEffect, useState} from "react";
import {AppSidebar} from "@/components/app-sidebar";
import {DataTable} from "@/components/table/data-table";
import {baseColumns, selectionColumn, actionsColumn} from "@/lib/tables-
columns";
import {CRUD} from "@/src/utils/api";
import {SidebarInset, SidebarProvider, SidebarTrigger} from
"@/components/ui/sidebar";
import {Separator} from "@/components/ui/separator";
import {
  Breadcrumb,
  BreadcrumbItem,
  BreadcrumbLink,
  BreadcrumbList,
  BreadcrumbPage,
  BreadcrumbSeparator
} from "@/components/ui/breadcrumb";
import RecordDialog from "@/components/record-dialog";
import {useSession} from "next-auth/react";

const DashboardTable = () => {
  const { data: session, status } = useSession();
  const router = useRouter();
  const {table} = router.query;
  const [data, setData] = useState([]);
  const [loading, setLoading] = useState(false);

  const [isDialogOpen, setIsDialogOpen] = useState(false);
  const [dialogMode, setDialogMode] = useState<"add" | "edit">("add");
  const [currentData, setCurrentData] = useState<any>(null);

  if (status === "unauthenticated") {
    router.push("/auth/login");
    return null;
  }

  useEffect(() => {
    // @ts-ignore

```

```

    if (!table || !baseColumns[table]) return;

    const fetchData = async () => {
      setLoading(true);
      try {
        // @ts-ignore
        const data = await CRUD.getAll(table);
        setData(data);
      } catch (error) {
        console.error("Failed to fetch data:", error);
      } finally {
        setLoading(false);
      }
    };

    fetchData().then(r => r);
  }, [table]);

  // @ts-ignore
  if (!table || !baseColumns[table]) {
    return <p className="p-4">Table not found</p>;
  }

  const handleDelete = (row: any) => {
    console.log(`Delete ${table}:`, row);
  };

  const handleEdit = (row: any) => {
    setDialogMode("edit");
    setCurrentData(row);
    setIsDialogOpen(true);
  };

  const handleAdd = () => {
    setDialogMode("add");
    setCurrentData(null);
    setIsDialogOpen(true);
  };

  const handleSubmit = async (table: any, formData: any) => {
    try {
      if (dialogMode === "edit") {
        console.log(`Editing ${table}:`, formData);

        // Ensure the edited row ID exists
        if (!formData.id) {
          console.error("Edit operation requires an entity ID.");
          return;
        }

        await CRUD.update(table, formData.id, formData); // Use the
correct ID from formData
        console.log(`Entity updated successfully in ${table}.`);
      } else {
        console.log(`Adding to ${table}:`, formData);

        await CRUD.create(table, formData); // Use `formData` for
creating a new entity
        console.log(`Entity added successfully to ${table}.`);
      }

      setIsDialogOpen(false);
    }
  };

```

```

        // Refresh table data
        // @ts-ignore
        const refreshedData = await CRUD.getAll(table);
        setData(refreshedData);
    } catch (err) {
        console.error(`Failed to process operation on ${table}:`, err);
    }
};

const columns = [
    selectionColumn,
    // @ts-ignore
    ...baseColumns[table],
    actionsColumn(handleEdit, handleDelete),
];
return (
    <SidebarProvider>
        <AppSidebar/>
        <SidebarInset>
            <header
                className="flex h-16 shrink-0 items-center gap-2
transition-[width,height] ease-linear group-has-[[data-
collapsible=icon]]/sidebar-wrapper:h-12">
                <div className="flex items-center gap-2 px-4">
                    <SidebarTrigger className="-ml-1"/>
                    <Separator orientation="vertical" className="mr-2 h-
4"/>
                    <Breadcrumbs>
                        <BreadcrumbsList>
                            <BreadcrumbItem className="hidden md:block">
                                <BreadcrumbLink href="/dashboard">
                                    Dashboard
                                </BreadcrumbLink>
                            </BreadcrumbItem>
                            <BreadcrumbSeparator className="hidden
md:block"/>
                            <BreadcrumbItem>
                                <BreadcrumbPage>{table}</BreadcrumbPage>
                            </BreadcrumbItem>
                        </BreadcrumbsList>
                    </Breadcrumbs>
                </div>
            </header>
            <div>
                <DataTable
                    columns={columns}
                    data={data}
                    onAdd={handleAdd}
                    onDelete={handleDelete}
                />
                {isDialogOpen && (
                    <RecordDialog
                        mode={dialogMode}
                        table={table as keyof typeof baseColumns}
                        data={currentData}
                        onSubmit={handleSubmit}
                        onClose={() => setIsDialogOpen(false)}
                    />
                )}
            </div>
        </SidebarInset>
    </SidebarProvider>
);

```

```

        </SidebarProvider>
    );
};

export default DashboardTable;

```

## Листинг 32 – data-table.tsx

```

"use client";
import {
    ColumnDef,
    ColumnFiltersState,
    SortingState,
    VisibilityState,
    flexRender,
    getCoreRowModel,
    getFilteredRowModel,
    getPaginationRowModel,
    getSortedRowModel,
    useReactTable,
} from "@tanstack/react-table";
import {ChevronDown, Plus, Trash2} from "lucide-react";
import {useState} from "react";

import {Button} from "@components/ui/button";
import {
    DropdownMenu,
    DropdownMenuCheckboxItem,
    DropdownMenuContent,
    DropdownMenuTrigger,
} from "@components/ui/dropdown-menu";
import {Input} from "@components/ui/input";
import {
    Table,
    TableBody,
    TableCell,
    TableHead,
    TableHeader,
    TableRow,
} from "@components/ui/table";

interface DataTableProps<TData, TValue> {
    columns: ColumnDef<TData, TValue>[];
    data: TData[];
    onAdd?: () => void;
    onDelete?: (rows: TData[]) => void;
}

export function DataTable<TData, TValue>({
    columns,
    data,
    onAdd,
    onDelete,
}: DataTableProps<TData, TValue>) {
    const [sorting, setSorting] = useState<SortingState>([]);
    const [columnFilters, setColumnFilters] =
        useState<ColumnFiltersState>([]);
    const [columnVisibility, setColumnVisibility] =
        useState<VisibilityState>({});
    const [rowSelection, setRowSelection] = useState({});
    const [selectedFilterColumn, setSelectedFilterColumn] =
        useState<string>("");

```



```

const table = useReactTable({
  data,
  columns,
  onSortingChange: setSorting,
  onColumnFiltersChange: setColumnFilters,
  getCoreRowModel: getCoreRowModel(),
  getPaginationRowModel: getPaginationRowModel(),
  getSortedRowModel: getSortedRowModel(),
  getFilteredRowModel: getFilteredRowModel(),
  onColumnVisibilityChange: setColumnVisibility,
  enableRowSelection: true,
  onRowSelectionChange: setRowSelection,
  state: {
    sorting,
    columnFilters,
    columnVisibility,
    rowSelection,
  },
});

const selectedRows = table
  .getSelectedRowModel()
  .rows.map((row) => row.original);

return (
  <div className="h-full w-full p-4">
    <div className="flex items-center justify-between mb-4">
      <div className="flex items-center gap-2">
        {/* Dropdown для выбора колонки */}
        {/* Поле ввода для фильтрации */}
        <Input
          placeholder=
            {selectedFilterColumn
              ? `Type ${
                  columns.find((col) => col.id ===
selectedFilterColumn)?.header || selectedFilterColumn
                }`
              : "Select column"}
          value={
            selectedFilterColumn
              ?
(table.getColumn(selectedFilterColumn)?.getFilterValue() as string) ?? ""
              : ""
            }
          onChange={ (event) =>
            selectedFilterColumn &&
            table.getColumn(selectedFilterColumn)?.setFilterValue(event.target.value)
          }
          className="max-w-sm"
          disabled={!selectedFilterColumn}
        />
        <DropdownMenu>
          <DropdownMenuTrigger asChild>
            <Button variant="outline">
              {selectedFilterColumn
                ? `Filter by: ${
                    columns.find((col) => col.id ===
selectedFilterColumn)?.header || selectedFilterColumn
                  }`
                : "Select column"}
            </Button>
          </DropdownMenuTrigger>
        </DropdownMenu>
      </div>
    </div>
  </div>
);

```

```

        </Button>

        </DropdownMenuTrigger>
        <DropdownMenuContent align="start">
            {columns
                .filter((column) => column.header &&
column.header !== "Actions" && column.id !== "select")
                .map((column) => (
                    <DropdownMenuCheckboxItem
                        key={column.id}
                        checked={selectedFilterColumn ===
column.id}
                        onCheckedChange={() =>
setSelectedFilterColumn(column.id as string)}
                    >
                        {column.header as string}
                    </DropdownMenuCheckboxItem>
                ))}
        </DropdownMenuContent>
    </DropdownMenu>

</div>
<div className="flex items-center gap-2">
    {onDelete && selectedRows.length > 0 && (
        <Button
            variant="destructive"
            onClick={() => onDelete(selectedRows)}
        >
            <Trash2 className="mr-2 h-4 w-4"/>
            Delete {selectedRows.length} Item(s)
        </Button>
    )}
    {onAdd && (
        <Button onClick={onAdd}>
            <Plus className="mr-2 h-4 w-4"/> Add New
        </Button>
    )}
    <DropdownMenu>
        <DropdownMenuTrigger asChild>
            <Button variant="outline" className="ml-auto">
                Columns <ChevronDown className="ml-2 h-4 w-
4"/>
            </Button>
        </DropdownMenuTrigger>
        <DropdownMenuContent align="end">
            {table
                .getAllColumns()
                .filter((column) => column.getCanHide())
                .map((column) => {
                    return (
                        <DropdownMenuCheckboxItem
                            key={column.id}
                            className="capitalize"
                            checked={column.getIsVisible()}
                            onCheckedChange={(value) =>
column.toggleVisibility(!value)}
                        >
                            {column.id}
                        </DropdownMenuCheckboxItem>
                    );
                })
            }
        </DropdownMenuContent>
    </DropdownMenu>

```

```

        }}}
      </DropdownMenuContent>
    </DropdownMenu>
  </div>
</div>

<div className="rounded-md border">
  <Table>
    <TableHeader>
      {table.getHeaderGroups().map((headerGroup) => (
        <TableRow key={headerGroup.id}>
          {headerGroup.headers.map((header) => {
            return (
              <TableHead key={header.id}>
                {header.isPlaceholder
                  ? null
                  : flexRender(
header.column.columnDef.header,
                    header.getContext()
                )}
              </TableHead>
            );
          })}
        </TableRow>
      )})}
    </TableHeader>
    <TableBody>
      {table.getRowModel().rows?.length ? (
        table.getRowModel().rows.map((row) => (
          <TableRow
            key={row.id}
            data-state={row.getIsSelected() &&
"selected"}
          >
            {row.getVisibleCells().map((cell) => (
              <TableCell key={cell.id}>
                {flexRender(
                  cell.column.columnDef.cell,
                  cell.getContext()
                )}
              </TableCell>
            )})}
          </TableRow>
        )}
      ) : (
        <TableRow>
          <TableCell
            colSpan={columns.length}
            className="h-24 text-center"
          >
            No results.
          </TableCell>
        </TableRow>
      )}
    </TableBody>
  </Table>
</div>
<div className="flex items-center justify-between space-x-2 py-
4">
  <div className="flex-1 text-sm text-muted-foreground">
    {table.getFilteredSelectedRowModel().rows.length} of{" "}

```

```

        {table.getFilteredRowModel().rows.length} row(s)
selected.
      </div>
      <div className="space-x-2">
        <Button
          variant="outline"
          size="sm"
          onClick={() => table.previousPage()}
          disabled={!table.getCanPreviousPage()}
        >
          Previous
        </Button>
        <Button
          variant="outline"
          size="sm"
          onClick={() => table.nextPage()}
          disabled={!table.getCanNextPage()}
        >
          Next
        </Button>
      </div>
    </div>
  );
}

```

### Листинг 33 – app-sidebar.tsx

```

"use client"

import * as React from "react"
import {
  BookOpen,
  Bot,
  Briefcase,
  Building,
  ChartArea,
  ClipboardList,
  Contact,
  CreditCard,
  GalleryVerticalEnd,
  ListTodo,
  Package,
  ScrollText,
  Settings2,
  SquareTerminal,
  Users,
} from "lucide-react"

import {NavMain} from "@/components/nav-main"
import {NavTables} from "@/components/nav-tables"
import {NavUser} from "@/components/nav-user"
import {TeamSwitcher} from "@/components/team-switcher"
import {
  Sidebar,
  SidebarContent,
  SidebarFooter,
  SidebarHeader,
  SidebarRail,
} from "@/components/ui/sidebar"
import {useSession} from "next-auth/react";
import api from "@/src/utills/api";

```

```

import {headers} from "next/headers";
import axios from "axios"
import {useEffect, useState} from "react";

const data = {
  teams: [
    {
      name: "Company",
      logo: GalleryVerticalEnd,
      plan: "Enterprise",
    },
  ],
  navMain: [
    {
      title: "Analytics",
      url: "#",
      icon: ChartArea,
      items: [
        {
          title: "Views",
          url: "/dashboard/views",
        },
        {
          title: "Log",
          url: "/dashboard/log",
        }
      ]
    },
    {
      title: "Documentation",
      url: "#",
      icon: BookOpen,
      items: [
        {
          title: "Introduction",
          url: "#",
        },
        {
          title: "Get Started",
          url: "#",
        },
        {
          title: "Tutorials",
          url: "#",
        },
        {
          title: "Changelog",
          url: "#",
        }
      ]
    },
    {
      title: "Settings",
      url: "#",
      icon: Settings2,
      items: [
        {
          title: "General",
          url: "#",
        },
        {
          title: "Billing",
          url: "#",
        }
      ]
    }
  ]
}

```

```

        },
    ],
},
],
tables: [
    {
        name: "Clients",
        url: "/dashboard/clients",
        icon: Contact,
    },
    {
        name: "Orders",
        url: "/dashboard/orders",
        icon: ClipboardList,
    },
    {
        name: "Employees",
        url: "/dashboard/employees",
        icon: Briefcase,
    },
    {
        name: "Tasks",
        url: "/dashboard/tasks",
        icon: ListTodo,
    },
    {
        name: "Warehouses",
        url: "/dashboard/warehouses",
        icon: Building,
    },
    {
        name: "Equipments",
        url: "/dashboard/equipment",
        icon: Package,
    },
    {
        name: "Payment",
        url: "/dashboard/payments",
        icon: CreditCard,
    },
    {
        name: "Users",
        url: "/dashboard/users",
        icon: Users,
    },
],
},
],
}

export function AppSidebar({...props}: React.ComponentProps<typeof Sidebar>)
{
    const {data: session, status} = useSession();

    if (status === "unauthenticated") {
        return null;
    }

    const [userData, setUserData] = useState<any>(null);
    const [loading, setLoading] = useState(true);

    useEffect(() => {
        const fetchData = async () => {
            try {

```

```

        const response = await
axios.get("http://localhost/api/get_user_by_username", {
    // @ts-ignore
    params: {username: session?.user?.id},
});
    setUserData(response.data);
} catch (e) {
    console.error("Error fetching user data:", e);
} finally {
    setLoading(false);
}
    };

    fetchData();
}, [session]);

return (
    <Sidebar collapsible="icon" {...props}>
        <SidebarHeader>
            <TeamSwitcher teams={data.teams}/>
        </SidebarHeader>
        <SidebarContent>
            <NavTables tables={data.tables}/>
            <NavMain items={data.navMain}/>
        </SidebarContent>
        <SidebarFooter>
            <NavUser
                user={{
                    name: userData?.full_name || "Unknown",
                    email: userData?.username || "Unknown",
                    role: userData?.role || "user",
                }}
            />
        </SidebarFooter>
        <SidebarRail/>
    </Sidebar>
);
}

```

Company  
Enterprise

Tables

Clients

Orders

Employees

Tasks

Warehouses

Equipments

Payment

Users

More

Platform

Analytics

Documentation

Settings

DK

Daniil Klimov

admin

daniil@klimov.com

Dashboard > Log

Select column

Select column

	ID	Operation	Table Name	Date
<input checked="" type="checkbox"/>	1	INSERT	employees	2024-11-30T13:51:39.020079
<input checked="" type="checkbox"/>	2	INSERT	users	2024-11-30T16:04:02.507942
<input checked="" type="checkbox"/>	3	INSERT	employees	2024-11-30T17:47:25.245600
<input checked="" type="checkbox"/>	4	INSERT	employees	2024-11-30T17:47:25.245600
<input checked="" type="checkbox"/>	5	INSERT	employees	2024-11-30T17:47:25.245600
<input type="checkbox"/>	6	INSERT	employees	2024-11-30T17:47:25.245600
<input type="checkbox"/>	7	INSERT	employees	2024-11-30T17:47:25.245600
<input type="checkbox"/>	8	INSERT	employees	2024-11-30T17:47:25.245600
<input type="checkbox"/>	9	INSERT	employees	2024-11-30T17:47:25.245600
<input type="checkbox"/>	10	INSERT	clients	2024-11-30T17:47:25.319530

5 of 10 row(s) selected.

Columns

✓ Select

✓ Log\_id

✓ Operation

✓ Table\_name

Changed\_data

Changed\_by

✓ Timestamp

PreviousNext

Рисунок 37 – Возможность скрытия колонок

Company  
Enterprise

Tables

Clients

Orders

Employees

Tasks

Warehouses

Equipments

Payment

Users

More

Platform

Analytics

Documentation

Settings

DK

Daniil Klimov

admin

daniil@klimov.com

Dashboard > orders

Select column

Select column

+ Add New

Columns

	ID	Order Date	Order Status	Total Cost (\$)	Client ID	Actions
<input type="checkbox"/>	1	2024-11-10	в процессе	50000.00	1	Edit Delete
<input type="checkbox"/>	2	2024-11-11	завершен	15000.00	2	Edit Delete
<input type="checkbox"/>	3	2024-11-12	в процессе	30000.00	3	Edit Delete
<input type="checkbox"/>	4	2024-11-13	завершен	25000.00	4	Edit Delete
<input type="checkbox"/>	5	2024-11-14	в процессе	40000.00	5	Edit Delete
<input type="checkbox"/>	6	2024-11-15	отменен	10000.00	6	Edit Delete
<input type="checkbox"/>	7	2024-11-16	завершен	35000.00	7	Edit Delete
<input type="checkbox"/>	8	2024-11-10	в процессе	50000.00	1	Edit Delete
<input type="checkbox"/>	9	2024-11-11	завершен	15000.00	2	Edit Delete
<input type="checkbox"/>	10	2024-11-12	в процессе	30000.00	3	Edit Delete

0 of 10 row(s) selected.

PreviousNext

Рисунок 38 – Таблица заказов



Company  
Enterprise

Tables

Clients

Orders

Employees

Tasks

Warehouses

Equipments

Payment

Users

More

Platform

Analytics

Documentation

Settings

Dashboard > clients

Select columnSelect column

+ Add NewColumns

	ID	Full Name	Phone Number	Email Address	Physical Address	Actions
	1	Иванов Иван	89001234567	ivanov@mail.ru	ул. Пушкина, д. 10	Edit Delete
	2	Петров Петр	89009876543	petrov@mail.ru	ул. Лермонтова, д. 15	Edit Delete
	3	Сидорова Анна	89001112233	sidorova@mail.ru	ул. Чехова, д. 8	Edit Delete
	4	Кузнецов Олег	89002223344	kuznetsov@mail.ru	ул. Маяковского, д. 3	Edit Delete
	5	Смирнова Мария	89009998877	smirnova@mail.ru	ул. Садовая, д. 12	Edit Delete
	6	Федоров Алексей	89003334455	fedorov@mail.ru	ул. Кирова, д. 21	Edit Delete
	7	Михайлова Оксана	89004445566	mikhailova@mail.ru	ул. Тверская, д. 18	Edit Delete
	8	Иванов Иван	89001234567	ivanov@mail.ru	ул. Пушкина, д. 10	Edit Delete
	9	Петров Петр	89009876543	petrov@mail.ru	ул. Лермонтова, д. 15	Edit Delete
	10	Сидорова Анна	89001112233	sidorova@mail.ru	ул. Чехова, д. 8	Edit Delete

0 of 10 row(s) selected.

PreviousNext

DK

Daniil Klimov

admin

daniil@klimov.com

Рисунок 39 – Таблица клиентов до изменения клиента

Company  
Enterprise

Tables

Clients

Orders

Employees

Tasks

Warehouses

Equipments

Payment

Users

More

Platform

Analytics

Documentation

Settings

Dashboard > clients

Select columnSelect column

+ Add NewColumns

	ID	Full Name	Phone Number	Email Address	Physical Address	Responsible Employee	Actions
	1	Иванов Иван			ул. Пушкина, д. 10	1	Edit Delete
	2	Петров Петр			ул. Лермонтова, д. 15	2	Edit Delete
	3	Сидорова Анна			ул. Чехова, д. 8	3	Edit Delete
	4	Кузнецов Олег			ул. Маяковского, д. 3	4	Edit Delete
	5	Смирнова Мария			ул. Садовая, д. 12	1	Edit Delete
	6	Федоров Алексей			ул. Кирова, д. 21	2	Edit Delete
	7	Михайлова Оксана			ул. Тверская, д. 18	3	Edit Delete
	8	Иванов Иван			ул. Пушкина, д. 10	1	Edit Delete
	9	Петров Петр			ул. Лермонтова, д. 15	2	Edit Delete
	10	Сидорова Анна			ул. Чехова, д. 8	3	Edit Delete

0 of 10 row(s) selected.

PreviousNext

DK

Daniil Klimov

admin

daniil@klimov.com

Edit clients

ID

3

Full Name

Иванова (Сидорова) Анна

Phone Number

89001112233

Email Address

sidorova@mail.ru

Physical Address

ул. Чехова, д. 8

Responsible Employee

3

Cancel

Save Changes

Рисунок 40 – Окно изменения клиента

Company  
Enterprise

Tables

Clients

Orders

Employees

Tasks

Warehouses

Equipments

Payment

Users

More

Platform

Analytics

Documentation

Settings

Dashboard > clients

Select columnSelect column

+ Add NewColumns

ID	Full Name	Phone Number	Email Address	Physical Address	Responsible Employee	Actions
1	Иванов Иван	89001234567	ivanov@mail.ru	ул. Пушкина, д. 10	1	Edit Delete
2	Петров Петр	89009876543	petrov@mail.ru	ул. Лермонтова, д. 15	2	Edit Delete
3	Иванова (Сидорова) Анна	89001112233	sidorova@mail.ru	ул. Чехова, д. 8	3	Edit Delete
4	Кузнецов Олег	89002223344	kuznetsov@mail.ru	ул. Маяковского, д. 3	4	Edit Delete
5	Смирнова Мария	89009998877	smirnova@mail.ru	ул. Садовая, д. 12	1	Edit Delete
6	Федоров Алексей	89003334455	fedorov@mail.ru	ул. Кирова, д. 21	2	Edit Delete
7	Михайлова Оксана	89004445566	mikhailova@mail.ru	ул. Тверская, д. 18	3	Edit Delete
8	Иванов Иван	89001234567	ivanov@mail.ru	ул. Пушкина, д. 10	1	Edit Delete
9	Петров Петр	89009876543	petrov@mail.ru	ул. Лермонтова, д. 15	2	Edit Delete
10	Сидорова Анна	89001112233	sidorova@mail.ru	ул. Чехова, д. 8	3	Edit Delete

0 of 10 row(s) selected.

PreviousNext

DK

Daniil Klimov

admin

daniil@klimov.com

Рисунок 41 – Таблица клиентов после подтверждения изменений

Company  
Enterprise

Tables

Clients

Orders

Employees

Tasks

Warehouses

Equipments

Payment

Users

More

Platform

Analytics

Documentation

Settings

Dashboard > employees

ТестFilter by: Full Name

+ Add NewColumns

ID	Full Name	Job Title	Contact Number	Work Email	Actions
8	Тестов Иван	Тестирующий	+71234567890	test@compa.ny	Edit Delete

0 of 1 row(s) selected.

PreviousNext

DK

Daniil Klimov

admin

daniil@klimov.com

Рисунок 42 – Демонстрация возможности поиска по нужной колонке

**Add New warehouses**

**Warehouse Location**

Amazon Boulevard, 1

**Responsible Employee**

1

Cancel Add Record

Рисунок 43 – Окно добавления записи в таблицу

Company Enterprise

Dashboard > warehouses

Select column Select column

Delete 1 Item(s) + Add New Columns

ID	Warehouse Location	Actions
11	Склад на ул. Советская	Edit Delete
12	Склад на ул. Гагарина	Edit Delete
13	Склад на ул. Жукова	Edit Delete
14	Склад на ул. Пушкина	Edit Delete
15	Склад на ул. Дружбы	Edit Delete
16	Склад на ул. Победы	Edit Delete
17	Amazon Boulevard, 1	Edit Delete

1 of 17 row(s) selected

Previous Next

Daniil Klimov admin daniil@klimov.com

Рисунок 44 – После ввода данных новая запись отобразилась в конце списка, на второй странице

## 5 РЕЗЕРВНОЕ КОПИРОВАНИЕ И ВОССТАНОВЛЕНИЕ

Для выполнения резервного копирования создадим bash-скрипт:

### Листинг 34 – Скрипт backup\_script.sh

```
#!/bin/bash

# Настройки
DB_NAME="company_services_db"
DB_USER="postgres"
BACKUP_DIR="/backup"
TIMESTAMP=$(date +%Y%m%d_%H%M%S)
BACKUP_FILE="$BACKUP_DIR/$DB_NAME_$TIMESTAMP.sql"

mkdir -p "$BACKUP_DIR"

pg_dump -U "$DB_USER" -F c -f "$BACKUP_FILE" "$DB_NAME"

if [ $? -eq 0 ]; then
    echo "Резервная копия успешно создана: $BACKUP_FILE"
else
    echo "Ошибка создания резервной копии"
    exit 1
fi
```

Теперь нужно сделать настройку автоматического выполнения. Для этого добавим его в cron.

### Листинг 35 – Команды для cron

```
crontab -e

0 3 * * * /path/to/backup_script.sh

crontab -l
```

После настройки этого скрипта мы сможем в любой момент вернуться к последней сохраненной версии базы данных.

### Листинг 36 – Команда для восстановления

```
pg_restore -U postgres -d company_service_db /backup/latest_backup.sql
```

## **ЗАКЛЮЧЕНИЕ**

В ходе выполнения курсовой работы была достигнута поставленная цель — разработан многопользовательский веб-интерфейс для управления базой данных информационной системы компании, предоставляющей услуги коммуникаций и видеонаблюдения.

В процессе работы были выполнены следующие задачи:

- спроектирована и смоделирована структура базы данных, соответствующая потребностям компании;
- реализована база данных в выбранной системе управления базами данных (СУБД);
- настроена базовая система безопасности для защиты данных, включая ограничение доступа и разграничение прав пользователей;
- разработан и протестирован сервис для взаимодействия с базой данных через веб-интерфейс;
- изучены и реализованы основные методы резервного копирования базы данных для обеспечения ее сохранности и надежности.

Результаты работы подтвердили возможность применения разработанного решения для повышения эффективности взаимодействия сотрудников с базами данных компании. Веб-интерфейс обеспечивает удобство и доступность управления информацией, а внедренные меры безопасности и возможности резервного копирования повышают защиту данных и их сохранность.

Таким образом, поставленные задачи выполнены, цель работы достигнута. Разработанное решение может быть использовано в реальной работе компании с возможностью дальнейшей доработки и масштабирования под растущие потребности.