

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

Факультет безопасности информационных технологий

Направление подготовки: 11.03.03

Образовательная программа: Безопасность информационных технологий

Дисциплина:

«Информационная безопасность баз данных»

КУРСОВАЯ РАБОТА

на тему « Информационная система ремонтной мастерской »

Выполнил:

Рядовой Т.С., студент группы N3352, поток ИББД.N63 1.5

(подпись)

Проверил:

Таранов Сергей Владимирович

(отметка о выполнении)

(подпись)

(дата)

Санкт-Петербург
2025 г.

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

ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ (РАБОТУ)

Студент Рядовой Тимофей Сергеевич
(Фамилия, И., О.)
Факультет Безопасности информационных технологий
Группа N3352
Направление (специальность) 11.03.03 Конструирование и технология
электронных средств
Руководитель Таранов С. В., университет ИТМО, доцент (квалификационная
категория "ординарный доцент"), старший научный сотрудник, к.т.н.
(Фамилия, И.О., должность, ученое звание, степень)
Дисциплина Информационная безопасность баз данных

Наименование темы Информационная система ремонтной мастерской

Задание Спроектировать и разработать базу данных для информационной
системы ремонтной мастерской. Реализовать приложение
для взаимодействия с базой данных. Обеспечить защиту данных, включая мониторинг,
шифрование и разграничение доступа. Реализовать резервное копирование и восстановление
базы данных.

Краткие методические указания Использовать метод «сущность-связь»
для моделирования базы данных. Привести схему отношений к третьей нормальной форме
(3 НФ). Реализовать приложение на Python с использованием фреймворка FastAPI и ORM
SQLAlchemy. Обеспечить логирование изменений в базе данных.

Содержание пояснительной записки
1. Инфологическое моделирование базы данных.
2. Реализация базы данных в СУБД PostgreSQL.
3. Защита базы данных: мониторинг, шифрование, разграничение доступа.
4. Разработка приложения для взаимодействия с базой данных.
5. Резервирование и восстановление базы данных.
Рекомендуемая литература
1. Документация к PostgreSQL.
2. Документация к FastAPI.
3. Документация к SQLAlchemy.

Руководитель _____
Подпись, дата
Студент _____
Подпись, дата

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

ГРАФИК ВЫПОЛНЕНИЯ КУРСОВОГО ПРОЕКТА (РАБОТЫ)

Студент Рядовой Тимофей Сергеевич
(Фамилия, И.О.)
Факультет Безопасности информационных технологий
Группа N3352
Направление (специальность) 11.03.03 Конструирование и технология
электронных средств
Руководитель Таранов С. В., университет ИТМО, доцент (квалификационная
категория "ординарный доцент"), старший научный сотрудник, к.т.н.
(Фамилия, И.О., место работы, должность, ученое звание, степень)
Дисциплина Информационная безопасность баз данных
Наименование темы Информационная система ремонтной мастерской

№ п/п	Наименование этапа	Дата завершения		Оценка и подпись руководителя
		Планируемая	Фактическая	
1	Инфологическое моделирование базы данных	20.11.24	21.02.25	
2	Реализация базы данных в СУБД	21.11.24	22.02.25	
3	Защита базы данных	23.11.24	23.02.25	
4	Разработка приложения	27.11.24	26.02.25	
5	Резервирование и восстановление базы данных	28.11.24	28.02.25	

Руководитель _____
подпись, дата
Студент _____
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 Рядовой Тимофей Сергеевич
(Фамилия, И.О.)
Факультет Безопасности информационных технологий
Группа N3352
Направление (специальность) 11.03.03 Конструирование и технология
электронных средств
Руководитель Таранов С. В., университет ИТМО, доцент (квалификационная
категория "ординарный доцент"), старший научный сотрудник, к.т.н.
(Фамилия, И.О., место работы, должность, ученое звание, степень)
Дисциплина Информационная безопасность баз данных
Наименование темы Информационная система ремонтной мастерской

ХАРАКТЕРИСТИКА КУРСОВОГО ПРОЕКТА (РАБОТЫ)

- 1. Цель и задачи работы** ☐ Предложены студентом ☐ Сформулированы при участии студента ☐ Определены руководителем

Цель работы – проектирование и разработка информационной системы ремонтной
мастерской, включая базу данных, приложение для взаимодействия с ней, а также
обеспечение защиты данных.

Задачи:

1. Моделирование базы данных по методу «сущность-связь».
2. Реализация базы данных в СУБД PostgreSQL.
3. Обеспечение защиты данных: мониторинг, шифрование, разграничение доступа.
4. Разработка приложения на Python с использованием FastAPI и SQLAlchemy.
5. Реализация резервного копирования и восстановления базы данных.

2. Характер работы

- ☐ Расчет ☐ Конструирование
☒ Моделирование ☐ Другое,

3. Содержание работы

1. Инфологическое моделирование базы данных
2. Реализация базы данных в СУБД.
3. Защита данных.
4. Разработка приложения
5. Резервирование и восстановление базы данных.

4. Выводы

В результате работы была разработана информационная система ремонтной мастерской,
включающая базу данных, приложение для взаимодействия с ней, а также обеспечена защита
данных.

Студент _____
(подпись)

Руководитель _____
(подпись)

«__» _____ 20__ г.

СОДЕРЖАНИЕ

Содержание	5
Введение.....	7
1 Инфологическое моделирование баз данных по методу «сущность-связь»	8
1.1 Цель.....	8
1.2 Ход работы.....	8
1.2.1 Этап №1 - Системный анализ информационной системы	8
1.2.2 Этап №2 - Выделение сущностей и построение ER-диаграмм.....	11
1.2.3 Этап №3 - Преобразование ER-диаграммы	12
1.2.4 Этап №4 - Приведение отношений БД к 3НФ	12
1.2.5 Этап №5 – Моделирование уровня представлений ИС ремонтной мастерской	13
1.3 Вывод	13
2 Реализация базы данных в СУБД	15
2.1 Цель.....	15
2.2 Ход работы.....	15
2.2.1 Выбор СУБД	15
2.2.2 Создание БД и заполнение данными	16
2.2.3 Индексация таблиц	18
2.2.4 Установка взаимосвязей между таблицами	18
2.2.5 Заполнение данными	19
2.2.6 Создание представлений	22
2.3 Вывод	24
3 Защита данных.....	25
3.1 Цель.....	25
3.2 Ход работы.....	25
3.2.1 Задача №1 – мониторинг БД	25
3.2.2 Задача №2 – шифрование данных	27
3.2.3 Задача №3 – разграничение доступа к БД	29
3.3 Вывод	31
4 Разработка приложения.....	32
4.1 Цель.....	32
4.2 Ход работы.....	32
4.2.1 Обоснование выбора среды программирования и фреймворка.....	32

4.2.2	Структура	33
4.3	Взаимодействие с веб-приложением и базой данных.....	37
4.4	Методы защиты данных	41
4.5	Вывод	41
5	Резервирование и восстановление базы данных	42
5.1	Цель.....	42
5.2	Ход работы.....	42
5.2.1	Создание резервной копии БД	42
5.2.2	Внесение случайных изменений в таблицы.....	44
5.2.3	Создание контрольной точки и откат изменений	44
5.2.4	Анализ откаченных изменений	45
5.3	Вывод	45
	Заключение.....	46
	Список источников	47
	ПРИЛОЖЕНИЕ А.....	48

ВВЕДЕНИЕ

Информационные системы играют ключевую роль в современном мире, обеспечивая эффективное управление данными, поддержку бизнес-процессов и повышение уровня безопасности данных. Серия лабораторных работ направлена на изучение методов проектирования и разработки баз данных, включая процессы моделирования, создания структуры базы данных, индексирования, шифрования и резервирования данных. Каждая работа посвящена конкретным аспектам создания и эксплуатации реляционных баз данных, что позволит студентам получить практические навыки, необходимые для успешного построения и администрирования информационных систем.

1 ИНФОЛОГИЧЕСКОЕ МОДЕЛИРОВАНИЕ БАЗ ДАННЫХ ПО МЕТОДУ «СУЩНОСТЬ-СВЯЗЬ»

1.1 Цель

Цель работы: изучение способов семантического представления баз данных (БД), получение навыков инфологического проектирования с использованием нотации «сущность-связь».

1.2 Ход работы

В качестве информационной системы, для которой будет составлена БД, была выбрана «Ремонтная мастерская».

1.2.1 Этап №1 - Системный анализ информационной системы

Разрабатываемая база данных будет являться частью информационной системы ремонтной мастерской, которая обеспечивает автоматизацию следующих процессов:

- Управление заказами;
- Управление клиентами;
- Учёт используемых материалов и инвентаризация;
- Учёт услуг и расчёт стоимости;
- Управление оплатой и финансовый учёт;
- Мониторинг и отчетность.

Задачи, решаемые с помощью разрабатываемой базы данных:

- Автоматизация административных процессов: регистрация, обновление и отслеживание статуса заказов без необходимости ручной обработки данных;
- Оптимизация складского учета: поддержание актуальной информации по запасам материалов и автоматическое списание при их использовании;

- Улучшение сервиса и персонализация: хранение истории клиентов и их заказов для улучшения качества обслуживания;
- Контроль финансов: учёт поступлений и платежей по каждому заказу, анализ финансовых результатов;
- Отчётность и аналитика: генерация данных для анализа производительности, финансовых показателей и загруженности мастерской.

Источники данных:

- Поставщики материалов;
- Операторы мастерской (администраторы);
- Сотрудники мастерской (мастера и кладовщики);
- Клиенты;
- Системы оплаты.

Формат данных:

- Форма ввода: основные данные (заказы, клиенты, услуги, материалы) поступают в текстовом и числовом формате через формы интерфейса ИС;
- Файлы: от поставщиков и платёжных систем информация может поступать в виде файлов CSV, Excel или XML;
- API: при интеграции с платёжными системами и другими внешними сервисами данные поступают через API в JSON или XML формате.

Частота обновления: данные обновляются в реальном времени по мере поступления, с возможностью периодического резервного копирования.

Потребители информации:

- Операторы (менеджеры) мастерской;
- Сотрудники мастерской (мастера и кладовщики).

Формат представления информации:

- Для операторов: интерфейс с формами ввода, таблицами, списками, карточками клиентов и заказов, отчёты в формате PDF или Excel;
- Для сотрудников: списки задач, инструкции по ремонту, таблицы материалов, мобильные уведомления, отчёты о выполненной работе в формате JSON или XML.

Ограничения на сущности и связи:

- Клиенты:
 - Уникальность идентификатора клиента;
 - Уникальность телефона и/или адреса электронной почты;
 - Контактные данные (телефон или почта) должны быть заполнены.
- Заказы:
 - Поля «дата создания» и «дата завершения» не могут быть пустыми;
 - Сумма заказа не может быть отрицательной;
 - Статус заказа должен быть ограничен перечнем фиксированных значений;
 - После завершения заказа его статус не может быть изменён;
 - Запрещено удалять записи о завершённых заказах.
- Сотрудники:
 - Уникальность идентификатора сотрудника;
 - Должность сотрудника должна быть из ограниченного списка;
 - Мастер не может работать одновременно над несколькими заказами.
- Материалы:

- Уникальность идентификатора материала;
- Количество на складе и цена за единицу не могут быть отрицательными.
- Услуги:
 - Название услуги должно быть уникальным;
 - Базовая стоимость услуги должна быть положительным числом;
 - Услуга не может быть включена в заказ, если она отключена в списке активных услуг.

1.2.2 Этап №2 - Выделение сущностей и построение ER-диаграмм

На основе проведенного анализа ИС, для которой разрабатывается БД, можно выделить следующие сущности:

- Клиент (атрибуты: ID клиента, ФИО, телефон, email);
- Заказ (атрибуты: ID заказа, дата создания, дата завершения, статус, сумма);
- Сотрудник (атрибуты: ID сотрудника, ФИО, должность, контактный телефон);
- Материал (атрибуты: ID материала, название, количество на складе, цена за единицу);
- Услуга (атрибуты: ID услуги, название, базовая стоимость);
- Оплата (атрибуты: ID оплаты, сумма, дата оплаты, статус);
- Отчёт (атрибуты: ID отчёта, дата создания, тип отчёта, данные).

Связи между сущностями:

- Клиент - Заказ (1:M): каждый клиент может иметь несколько заказов, но каждый заказ принадлежит только одному клиенту;
- Заказ - Сотрудник (M:1): каждый заказ выполняется одним сотрудником, но сотрудник может выполнять несколько заказов;

- Заказ - Материал (М:М): в заказе может использоваться несколько материалов, и один материал может использоваться в нескольких заказах;
- Заказ —Услуга (М:М): в заказе может быть несколько услуг, и одна услуга может быть включена в несколько заказов;
- Заказ - Оплата (1:М): каждый заказ может иметь несколько оплат, но каждая оплата относится только к одному заказу;
- Отчёт -Заказ (1:М): каждый отчёт может содержать данные по нескольким заказам, но каждый заказ может быть включён только в один отчёт.

1.2.3 Этап №3 - Преобразование ER-диаграммы

На основе выделенных сущностей и связей между ними, ER-диаграмма преобразуется в схему отношений. Каждая сущность становится таблицей, а связи между сущностями отражаются через внешние ключи.

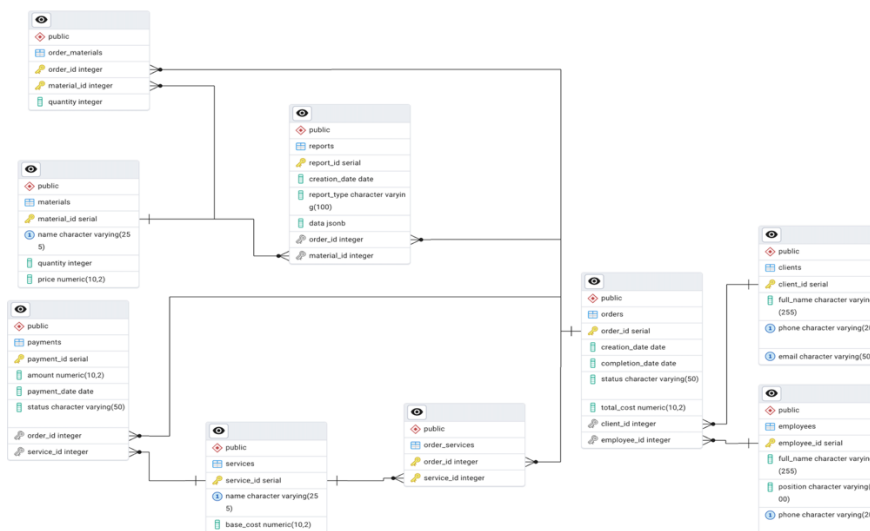


Рисунок 1 – ER-диаграмма базы данных

1.2.4 Этап №4 - Приведение отношений БД к 3НФ

Все таблицы приведены к третьей нормальной форме (3НФ), что обеспечивает отсутствие избыточности данных и транзитивных зависимостей.

1.2.5 Этап №5 – Моделирование уровня представлений ИС ремонтной мастерской

Потребитель «Операторы мастерской»:

- Представление 1. «Текущие заказы»:
 - ID заказа, статус, ФИО клиента, дата создания, сумма.
- Представление 2. «Отчёты по материалам»:
 - Название материала, количество на складе, цена за единицу, использовано в заказах.

Потребитель «Сотрудники мастерской»:

- Представление 1. «Задания для мастера»:
 - ID заказа, ФИО клиента, список услуг, список материалов.
- Представление 2. «Отчёты о выполненной работе»:
 - ID заказа, ФИО сотрудника, дата завершения, использованные материалы.

1.3 Вывод

Была спроектирована и разработана база данных для информационной системы ремонтной мастерской.

В рамках данной работы были последовательно выполнены следующие шаги:

1. Проведен анализ и определены сущности, атрибуты и связи между ними, что позволило построить ER-диаграмму базы данных;
2. Сформированы отношения, отражающие связи и ключевые зависимости, после чего модель была нормализована до третьей нормальной формы (3НФ);

3. Для обеспечения эффективного доступа к данным различных категорий пользователей (менеджеров и финансового отдела) были созданы представления.

2 РЕАЛИЗАЦИЯ БАЗЫ ДАННЫХ В СУБД

2.1 Цель

Цель работы: получение навыков по работе с современными системами управления базами данных (СУБД).

2.2 Ход работы

Задание:

1. Выбрать систему управления базами данных (СУБД) и обосновать свой выбор.
2. Создать БД в выбранной СУБД на основе итоговой разработанной схемы отношений из ЛР 1. Заполнить созданную БД информацией.
3. Индексировать таблицы. Добавить индексы для атрибутов, по которым происходит объединение таблиц, а также атрибуты, по которым выполняется поиск/фильтрация данных.
4. Установить взаимосвязи между таблицами.
5. Создать представления, составленные в пункте 5 лабораторной работы №1.

2.2.1 Выбор СУБД

В качестве инструментария для создания БД выбрана СУБД PostgreSQL с графическим интерфейсом pgAdmin4.

Обоснование выбора:

- PostgreSQL является одной из наиболее популярных и мощных СУБД с открытым исходным кодом;
- Поддержка сложных запросов, транзакций и индексов;
- Возможность работы с большими объемами данных;
- Широкая поддержка сообществом и обширная документация.

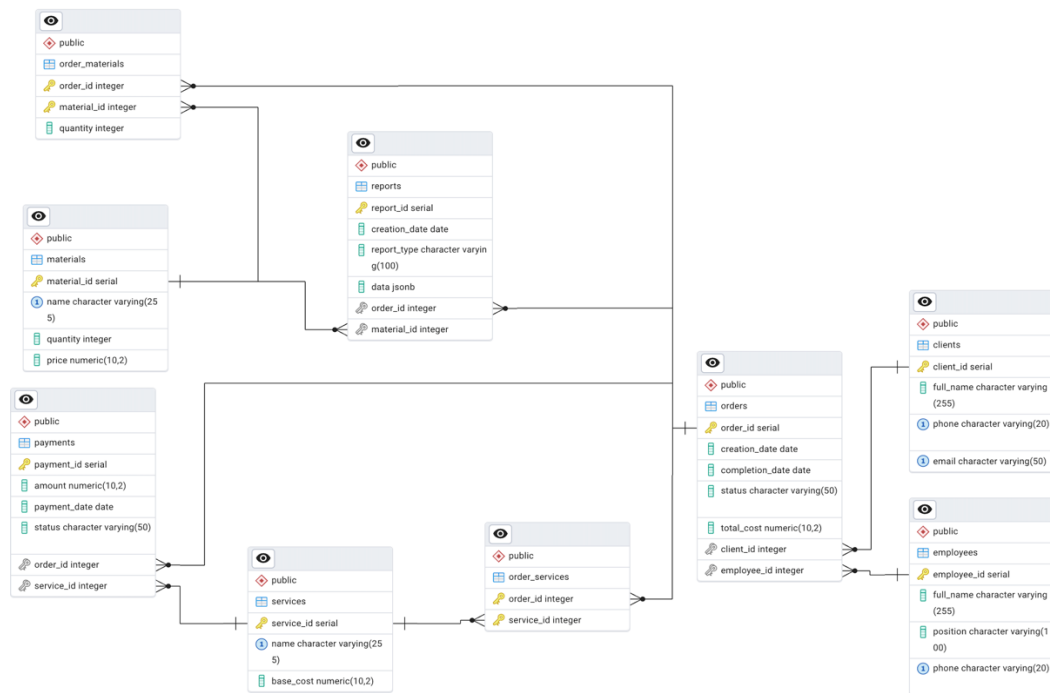


Рисунок 2 – ER-диаграмма базы данных

2.2.2 Создание БД и заполнение данными

Листинг 1 – Создание базы данных

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

Листинг 2 – Создание таблиц

```
-- Таблица "Клиенты"
CREATE TABLE IF NOT EXISTS public.clients (
    client_id SERIAL PRIMARY KEY,
    full_name VARCHAR(255) NOT NULL,
    phone VARCHAR(20) UNIQUE NOT NULL,
    email VARCHAR(50) UNIQUE
);

-- Таблица "Заказы"
CREATE TABLE IF NOT EXISTS public.orders (
    order_id SERIAL PRIMARY KEY,
    creation_date DATE NOT NULL,
    completion_date DATE,
    status VARCHAR(50) CHECK (status IN ('Новый', 'В процессе',
'Завершен', 'Отменен')) NOT NULL,
```



```

        total_cost NUMERIC(10, 2) CHECK (total_cost >= 0),
        client_id INTEGER REFERENCES clients(client_id) ON DELETE
CASCADE
);

-- Таблица "Сотрудники"
CREATE TABLE IF NOT EXISTS public.employees (
    employee_id SERIAL PRIMARY KEY,
    full_name VARCHAR(255) NOT NULL,
    position VARCHAR(100) NOT NULL,
    phone VARCHAR(20) UNIQUE NOT NULL
);

-- Таблица "Материалы"
CREATE TABLE IF NOT EXISTS public.materials (
    material_id SERIAL PRIMARY KEY,
    name VARCHAR(255) UNIQUE NOT NULL,
    quantity INTEGER CHECK (quantity >= 0),
    price NUMERIC(10, 2) CHECK (price >= 0)
);

-- Таблица "Услуги"
CREATE TABLE IF NOT EXISTS public.services (
    service_id SERIAL PRIMARY KEY,
    name VARCHAR(255) UNIQUE NOT NULL,
    base_cost NUMERIC(10, 2) CHECK (base_cost >= 0)
);

-- Таблица "Оплаты"
CREATE TABLE IF NOT EXISTS public.payments (
    payment_id SERIAL PRIMARY KEY,
    amount NUMERIC(10, 2) CHECK (amount >= 0),
    payment_date DATE NOT NULL,
    status VARCHAR(50) CHECK (status IN ('Оплачено', 'Частично
оплачено', 'Не оплачено')) NOT NULL,
    order_id INTEGER REFERENCES orders(order_id) ON DELETE
CASCADE
);

-- Таблица "Отчеты"
CREATE TABLE IF NOT EXISTS public.reports (
    report_id SERIAL PRIMARY KEY,
    creation_date DATE NOT NULL,
    report_type VARCHAR(100) NOT NULL,
    data JSONB,
    order_id INTEGER REFERENCES orders(order_id) ON DELETE
CASCADE
);

-- Таблица "Материалы заказа"
CREATE TABLE IF NOT EXISTS public.order_materials (
    order_id INTEGER REFERENCES public.orders(order_id) ON
DELETE CASCADE,

```

```

        material_id INTEGER REFERENCES public.materials(material_id)
ON DELETE CASCADE,
        quantity INTEGER NOT NULL CHECK (quantity > 0),
        PRIMARY KEY (order_id, material_id)
);

-- Таблица "Услуги заказа"
CREATE TABLE IF NOT EXISTS public.order_services (
        order_id INT REFERENCES public.orders(order_id) ON DELETE
CASCADE,
        service_id INT REFERENCES public.services(service_id) ON
DELETE CASCADE,
        PRIMARY KEY (order_id, service_id)
);

```

2.2.3 Индексация таблиц

Листинг 3 – Индексация таблиц

```

-- Индексы для таблицы "Заказы"
CREATE INDEX idx_orders_client_id ON public.orders(client_id);

-- Индексы для таблицы "Оплаты"
CREATE INDEX idx_payments_order_id ON public.payments(order_id);

-- Индексы для таблицы "Отчеты"
CREATE INDEX idx_reports_order_id ON public.reports(order_id);

```

2.2.4 Установка взаимосвязей между таблицами

Листинг 4 – Установка взаимосвязей

```

-- reports
ALTER TABLE public.reports
ADD COLUMN material_id INT;

ALTER TABLE public.reports
ADD CONSTRAINT fk_material FOREIGN KEY (material_id)
REFERENCES public.materials(material_id)
ON DELETE SET NULL;

-- payments
ALTER TABLE public.payments
ADD COLUMN service_id INT;

ALTER TABLE public.payments
ADD CONSTRAINT fk_service FOREIGN KEY (service_id)
REFERENCES public.services(service_id)
ON DELETE SET NULL;

-- orders
ALTER TABLE public.orders
ADD COLUMN employee_id INT;

```

```
ALTER TABLE public.orders
ADD CONSTRAINT fk_employee FOREIGN KEY (employee_id)
REFERENCES public.employees(employee_id)
ON DELETE SET NULL;
```

2.2.5 Заполнение данными

Листинг 5 – Заполнение таблиц данными

```
-- Клиенты (10 записей)
INSERT INTO public.clients (full_name, phone, email) VALUES
('Иванов Иван Иванович', '+79111234567', 'ivanov@mail.ru'),
('Петров Петр Петрович', '+79119876543', 'petrov@gmail.com'),
('Сидорова Анна Сергеевна', '+79112345678',
'sidorova@yandex.ru'),
('Васильев Василий Васильевич', '+79119871234',
'vasiliev@mail.ru'),
('Кузнецова Ольга Николаевна', '+79117654321',
'kuznetsova@bk.ru'),
('Смирнов Алексей Владимирович', '+79114567890',
'smirnov@outlook.com'),
('Андреев Андрей Андреевич', '+79115678901',
'andreev@gmail.com'),
('Морозова Марина Павловна', '+79113456789',
'morozova@yandex.ru'),
('Григорьев Григорий Григорьевич', '+79112349876',
'grigoriev@mail.ru'),
('Федорова Дарья Викторовна', '+79119875432', 'fedorova@bk.ru');

-- Сотрудники (10 записей)
INSERT INTO public.employees (full_name, position, phone) VALUES
('Смирнов Алексей Владимирович', 'Мастер', '+79111234569'),
('Кузнецова Ольга Ивановна', 'Кладовщик', '+79111234570'),
('Васильев Дмитрий Сергеевич', 'Администратор', '+79111234571'),
('Морозов Павел Игоревич', 'Инженер', '+79111234572'),
('Тимофеев Артем Викторович', 'Сервисный специалист',
'+79111234573'),
('Борисова Наталья Петровна', 'Бухгалтер', '+79111234574'),
('Капустин Виктор Александрович', 'Мастер', '+79111234575'),
('Семенова Екатерина Алексеевна', 'Кладовщик', '+79111234576'),
('Егоров Николай Олегович', 'Инженер', '+79111234577'),
('Фролова Ирина Владимировна', 'Оператор', '+79111234578');

-- Заказы (10 записей)
INSERT INTO public.orders (creation_date, completion_date,
status, total_cost, client_id, employee_id) VALUES
('2024-01-01', '2024-01-10', 'Завершен', 5000.00, 1, 1),
('2024-01-05', NULL, 'В процессе', 3000.00, 2, 2),
('2024-01-10', NULL, 'Новый', 2000.00, 3, 3),
('2024-02-01', NULL, 'Новый', 4000.00, 4, 4),
('2024-02-05', '2024-02-15', 'Завершен', 3500.00, 5, 5),
('2024-02-10', NULL, 'В процессе', 2500.00, 6, 6),
```

```

('2024-02-15', NULL, 'Новый', 1500.00, 7, 7),
('2024-02-20', NULL, 'Новый', 1800.00, 8, 8),
('2024-02-25', '2024-03-01', 'Завершен', 5000.00, 9, 9),
('2024-03-01', NULL, 'В процессе', 2200.00, 10, 10);

-- Материалы (10 записей)
INSERT INTO public.materials (name, quantity, price) VALUES
('Винты', 100, 10.00),
('Гайки', 200, 5.00),
('Шурупы', 150, 8.00),
('Провода', 300, 15.00),
('Платы', 50, 500.00),
('Разъемы', 120, 20.00),
('Клей', 75, 25.00),
('Термопаста', 100, 50.00),
('Чипы', 30, 700.00),
('Конденсаторы', 500, 2.00);

-- Услуги (10 записей)
INSERT INTO public.services (name, base_cost) VALUES
('Ремонт компьютера', 1000.00),
('Замена жесткого диска', 500.00),
('Установка программного обеспечения', 300.00),
('Чистка ноутбука', 700.00),
('Диагностика', 400.00),
('Замена термопасты', 600.00),
('Настройка сети', 800.00),
('Ремонт блока питания', 1200.00),
('Замена экрана', 2500.00),
('Перепайка компонентов', 1800.00);

-- Оплаты (10 записей)
INSERT INTO public.payments (amount, payment_date, status,
order_id, service_id) VALUES
(5000.00, '2024-01-10', 'Оплачено', 1, 1),
(1500.00, '2024-01-06', 'Частично оплачено', 2, 2),
(3500.00, '2024-02-15', 'Оплачено', 5, 3),
(1000.00, '2024-02-16', 'Частично оплачено', 6, 4),
(1800.00, '2024-02-26', 'Оплачено', 9, 5),
(1200.00, '2024-03-01', 'Оплачено', 10, 6),
(3000.00, '2024-02-21', 'Оплачено', 8, 7),
(700.00, '2024-02-22', 'Частично оплачено', 7, 8),
(2500.00, '2024-02-28', 'Оплачено', 4, 9),
(600.00, '2024-02-27', 'Частично оплачено', 3, 10);

-- Отчеты (10 записей)
INSERT INTO public.reports (creation_date, report_type, data,
order_id, material_id) VALUES
('2024-01-10', 'Финансовый отчет', '{"total_cost": 5000, "paid":
5000}', 1, 1),
('2024-01-06', 'Отчет по материалам', '{"materials_used":
["Винты", "Гайки"]}', 2, 2),

```

```

('2024-02-15', 'Финансовый отчет', '{"total_cost": 3500, "paid": 3500}', 5, 3),
('2024-02-16', 'Отчет по материалам', '{"materials_used": ["Шурупы"]}', 6, 4),
('2024-02-20', 'Финансовый отчет', '{"total_cost": 1800, "paid": 1800}', 8, 5),
('2024-02-22', 'Отчет по материалам', '{"materials_used": ["Провода", "Клей"]}', 7, 6),
('2024-02-25', 'Финансовый отчет', '{"total_cost": 5000, "paid": 5000}', 9, 7),
('2024-02-27', 'Отчет по материалам', '{"materials_used": ["Термопаста"]}', 3, 8),
('2024-02-28', 'Финансовый отчет', '{"total_cost": 2500, "paid": 2500}', 4, 9),
('2024-03-01', 'Финансовый отчет', '{"total_cost": 2200, "paid": 2200}', 10, 10);

```

-- Связь заказов и материалов

```

INSERT INTO public.order_materials (order_id, material_id, quantity) VALUES
(1, 1, 10),
(1, 2, 20),
(2, 3, 15),
(2, 4, 10),
(3, 5, 2),
(4, 6, 5),
(5, 7, 3),
(6, 8, 4),
(7, 9, 1),
(8, 10, 50),
(9, 1, 10),
(9, 3, 12),
(10, 2, 5),
(10, 5, 3);

```

-- Связь заказов и услуг

```

INSERT INTO public.order_services (order_id, service_id) VALUES
(1, 1),
(1, 2),
(2, 3),
(2, 4),
(3, 5),
(4, 6),
(5, 7),
(6, 8),
(7, 9),
(8, 10),
(9, 1),
(9, 3),
(10, 2),
(10, 4);

```

2.2.6 Создание представлений

2.2.6.1 Представление №1 – Текущие и новые заказы

Листинг 6 – Представление №1

```
CREATE VIEW public.current_orders AS
SELECT
    o.order_id,
    o.status,
    c.full_name AS client_name,
    o.creation_date,
    o.total_cost
FROM public.orders o
JOIN public.clients c ON o.client_id = c.client_id
WHERE o.status NOT IN ('Завершен', 'Отменен'); -- only active
orders
```

```
repair_workshop=# select * from current_orders;
 order_id | status | client_name | creation_date | total_cost
-----+-----+-----+-----+-----
      2 | В процессе | Петров Петр Петрович | 2024-01-05 | 3000.00
      3 | Новый | Сидорова Анна Сергеевна | 2024-01-10 | 2000.00
      4 | Новый | Васильев Василий Васильевич | 2024-02-01 | 4000.00
      6 | В процессе | Смирнов Алексей Владимирович | 2024-02-10 | 2500.00
      7 | Новый | Андреев Андрей Андреевич | 2024-02-15 | 1500.00
      8 | Новый | Морозова Марина Павловна | 2024-02-20 | 1800.00
     10 | В процессе | Федорова Дарья Викторовна | 2024-03-01 | 2200.00
(7 строк)
```

Рисунок 3 – Представление №1

2.2.6.2 Представление №2 - Отчет по материалам

Листинг 7 – Представление №2

```
CREATE VIEW public.material_reports AS
SELECT
    m.name AS material_name,
    m.material_id AS material_id,
    m.quantity AS stock_quantity,
    m.price AS unit_price,
    COALESCE(SUM(om.quantity), 0) AS used_in_orders
FROM public.materials m
LEFT JOIN public.order_materials om ON m.material_id =
om.material_id
GROUP BY m.material_id;
```

```
repair_workshop=# select * from material_reports;
```

material_name	material_id	stock_quantity	unit_price	used_in_orders
Чипы	9	30	700.00	1
Шурупы	3	150	8.00	27
Платы	5	50	500.00	5
Провода	4	300	15.00	10
Конденсаторы	10	500	2.00	50
Разъемы	6	120	20.00	5
Гайки	2	200	5.00	25
Клей	7	75	25.00	3
Винты	1	100	10.00	20
Термопаста	8	100	50.00	4

(10 строк)

Рисунок 4 – Представление №2

2.2.6.3 Представление №3 – Задания для мастера

Листинг 8 – Представление №3

```
CREATE VIEW public.master_tasks AS
SELECT
    o.order_id,
    c.full_name AS client_name,
    STRING_AGG(DISTINCT s.name, ', ') AS service_list,
    STRING_AGG(DISTINCT m.name, ', ') AS material_list
FROM public.orders o
JOIN public.clients c ON o.client_id = c.client_id
LEFT JOIN public.order_services os ON o.order_id = os.order_id
LEFT JOIN public.services s ON os.service_id = s.service_id
LEFT JOIN public.order_materials om ON o.order_id = om.order_id
LEFT JOIN public.materials m ON om.material_id = m.material_id
GROUP BY o.order_id, c.full_name;
```

```
repair_workshop=# select * from master_tasks;
```

order_id	client_name	service_list	material_list
1	Иванов Иван Иванович	Замена жесткого диска, Ремонт компьютера	Винты, Гайки
2	Петров Петр Петрович	Установка программного обеспечения, Чистка ноутбука	Провода, Шурупы
3	Сидорова Анна Сергеевна	Диагностика	Платы
4	Васильев Василий Васильевич	Замена термопасты	Разъемы
5	Кузнецова Ольга Николаевна	Настройка сети	Клей
6	Смирнов Алексей Владимирович	Ремонт блока питания	Термопаста
7	Андреев Андрей Андреевич	Замена экрана	Чипы
8	Морозова Марина Павловна	Перепайка компонентов	Конденсаторы
9	Григорьев Григорий Григорьевич	Ремонт компьютера, Установка программного обеспечения	Винты, Шурупы
10	Федорова Дарья Викторовна	Замена жесткого диска, Чистка ноутбука	Гайки, Платы

(10 строк)

Рисунок 5 – Представление №3

2.2.6.4 Представление №4 – Отчет о выполненных заказах

Листинг 9 – Представление №4

```
CREATE VIEW public.completed_work_reports AS
SELECT
    o.order_id,
    e.full_name AS employee_name,
    o.completion_date,
    STRING_AGG(DISTINCT m.name, ', ') AS used_materials
FROM public.orders o
JOIN public.employees e ON o.employee_id = e.employee_id
LEFT JOIN public.order_materials om ON o.order_id = om.order_id
LEFT JOIN public.materials m ON om.material_id = m.material_id
WHERE o.status = 'Завершен'
GROUP BY o.order_id, e.full_name, o.completion_date;
```

```
repair_workshop=# select * from completed_work_reports;
 order_id |          employee_name          | completion_date | used_materials
-----+-----+-----+-----
        1 | Смирнов Алексей Владимирович | 2024-01-10      | Винты, Гайки
        5 | Тимофеев Артем Викторович    | 2024-02-15      | Клей
        9 | Егоров Николай Олегович       | 2024-03-01      | Винты, Шурупы
(3 строки)
```

Рисунок 6 – Представление №4

2.3 Вывод

В ходе выполнения данной работы была создана база данных для ремонтной мастерской, заполнена тестовыми данными, проиндексированы ключевые атрибуты и созданы представления для удобства работы с данными.

3 ЗАЩИТА ДАННЫХ

3.1 Цель

Цель работы: получение навыков создания примитивных систем мониторинга, разграничения доступа и шифрования средствами СУБД

3.2 Ход работы

3.2.1 Задача №1 – мониторинг БД

Задачи:

- Создать таблицу-лог для записи изменений в БД;
- Создать триггеры для основных таблиц, которые будут фиксировать изменения (вставка, обновление, удаление) и записывать их в таблицу-лог;
- Продемонстрировать работу системы логирования.

3.2.1.1 Создание лог-таблицы

Листинг 10 – Создание таблицы

```
CREATE TABLE public.main_log (  
    log_item_id SERIAL PRIMARY KEY,  
    table_name VARCHAR(50) NOT NULL,  
    operation_type VARCHAR(30) NOT NULL,  
    operation_date TIMESTAMP,  
    user_operator VARCHAR(30) NOT NULL,  
    changed_data JSONB  
);
```

3.2.1.2 Создание функции и триггеров

Листинг 11 – Создание функции

```
CREATE OR REPLACE FUNCTION logging() RETURNS TRIGGER AS $$  
BEGIN  
    INSERT INTO public.main_log (table_name, operation_type,  
        operation_date, user_operator, changed_data)  
        VALUES (  
            TG_TABLE_NAME,  
            TG_OP,  
            NOW(),  
            current_user,  
            row_to_json(CASE WHEN TG_OP = 'DELETE' THEN OLD ELSE  
NEW END)
```

```

);
RETURN CASE WHEN TG_OP = 'DELETE' THEN OLD ELSE NEW END;
END;
$$ LANGUAGE plpgsql;

```

Листинг 12 – Создание триггеров

```

-- Триггер для таблицы "Заказы"
CREATE TRIGGER logging_orders
AFTER INSERT OR UPDATE OR DELETE ON public.orders
FOR EACH ROW EXECUTE FUNCTION logging();

-- Триггер для таблицы "Клиенты"
CREATE TRIGGER logging_clients
AFTER INSERT OR UPDATE OR DELETE ON public.clients
FOR EACH ROW EXECUTE FUNCTION logging();

-- Триггер для таблицы "Сотрудники"
CREATE TRIGGER logging_employees
AFTER INSERT OR UPDATE OR DELETE ON public.employees
FOR EACH ROW EXECUTE FUNCTION logging();

-- Вставка данных в таблицу "Заказы"
INSERT INTO public.orders (creation_date, completion_date,
status, total_cost, client_id, employee_id)
VALUES ('2024-02-02', NULL, 'Новый', 2500.00, 1, 1);

-- Обновление данных в таблице "Клиенты"
UPDATE public.clients SET email = 'new_email@mail.ru' WHERE
client_id = 1;

-- Удаление данных из таблицы "Сотрудники"
DELETE FROM public.employees WHERE employee_id = 1;

-- Проверка таблицы-лога
SELECT * FROM public.main_log;

```

3.2.1.3 Пример работы

Видим, что также обновились записи в таблице orders, где был удаленный employee_id.

Query

Query History

1

SELECT * FROM public.main_log;

2

Data Output

Messages

Notifications

SQL

	log_item_id [PK] integer	table_name character varying (50)	operation_type character varying (30)	operation_date timestamp without time zone	user_operator character varying (30)	changed_data jsonb
1	1	orders	INSERT	2025-03-05 19:48:44.413301	postgres	{ "status": "Новый", "order_id": 11, "client_id": 1, "total_cost": 2500.00 }
2	2	clients	UPDATE	2025-03-05 19:48:50.899209	postgres	{ "email": "new_email@mail.ru", "phone": "+79111234567", "client_id": 1 }
3	3	employees	DELETE	2025-03-05 19:50:13.882674	postgres	{ "phone": "+79111234569", "position": "Мастер", "full_name": "Смирнов" }
4	4	orders	UPDATE	2025-03-05 19:50:13.882674	postgres	{ "status": "Завершен", "order_id": 1, "client_id": 1, "total_cost": 5000.00 }
5	5	orders	UPDATE	2025-03-05 19:50:13.882674	postgres	{ "status": "Новый", "order_id": 11, "client_id": 1, "total_cost": 2500.00 }

Рисунок 7 – Пример заполнения логов после срабатывания триггеров

	order_id [PK] integer	creation_date date	completion_date date	status character varying (50)	total_cost numeric (10,2)	client_id integer	employee_id integer
1	2	2024-01-05	[null]	В процессе	3000.00	2	2
2	3	2024-01-10	[null]	Новый	2000.00	3	3
3	4	2024-02-01	[null]	Новый	4000.00	4	4
4	5	2024-02-05	2024-02-15	Завершен	3500.00	5	5
5	6	2024-02-10	[null]	В процессе	2500.00	6	6
6	7	2024-02-15	[null]	Новый	1500.00	7	7
7	8	2024-02-20	[null]	Новый	1800.00	8	8
8	9	2024-02-25	2024-03-01	Завершен	5000.00	9	9
9	10	2024-03-01	[null]	В процессе	2200.00	10	10
10	1	2024-01-01	2024-01-10	Завершен	5000.00	1	[null]
11	11	2024-02-02	[null]	Новый	2500.00	1	[null]

Рисунок 8 – Обновление таблицы orders после удаление employee_id

3.2.2 Задача №2 – шифрование данных

Задачи по шифрованию данных:

- Создать таблицу с секретными данными;
- Зашифровать данные в таблице с использованием симметричного алгоритма шифрования (например, AES-256);
- Продемонстрировать, что без знания ключа шифрования данные недоступны.

3.2.2.1 Создание таблицы и шифрование данных

Листинг 13 – Создание таблицы и шифрование

```
CREATE TABLE public.secret_data (
    id SERIAL PRIMARY KEY,
```

```

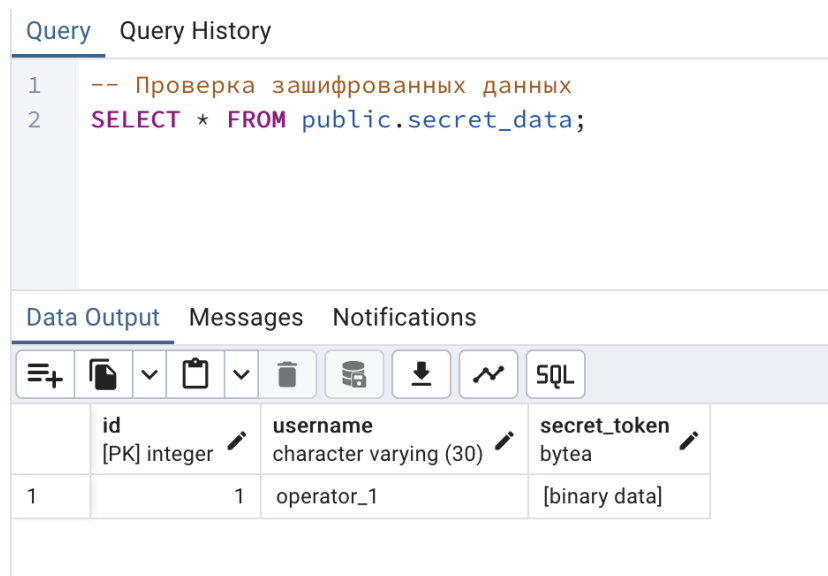
        username VARCHAR(30) NOT NULL,
        secret_token BYTEA NOT NULL
    );

-- Установка расширения pgcrypto
CREATE EXTENSION IF NOT EXISTS pgcrypto;

-- Вставка зашифрованных данных
INSERT INTO public.secret_data (username, secret_token)
VALUES (
    'operator_1', pgp_sym_encrypt('token_',
    '9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f
    00a08')
);

```

3.2.2.2 Пример работы



Query Query History

```

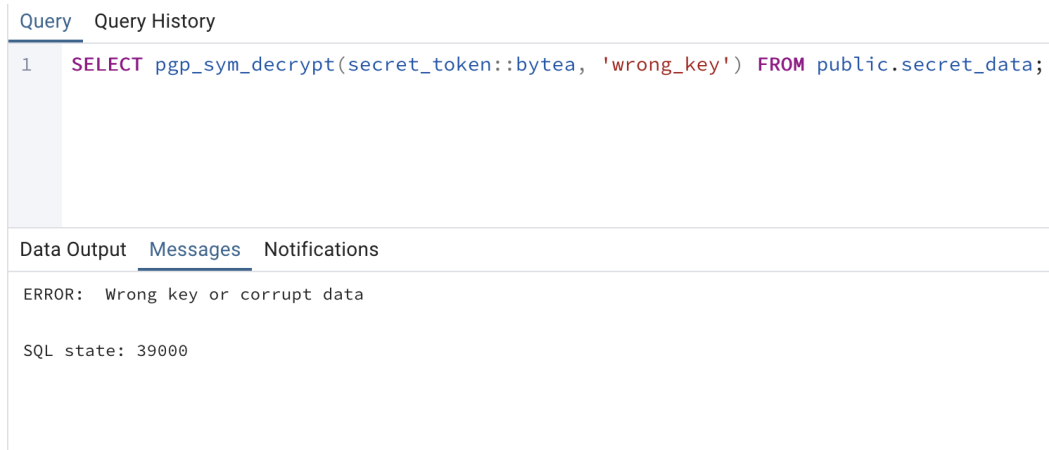
1 -- Проверка зашифрованных данных
2 SELECT * FROM public.secret_data;

```

Data Output Messages Notifications

	id [PK] integer	username character varying (30)	secret_token bytea
1	1	operator_1	[binary data]

Рисунок 9 – Проверка данных



Query Query History

```

1 SELECT pgp_sym_decrypt(secret_token::bytea, 'wrong_key') FROM public.secret_data;

```

Data Output Messages Notifications

ERROR: Wrong key or corrupt data

SQL state: 39000

Рисунок 10 – Попытка расшифровки данных без ключа

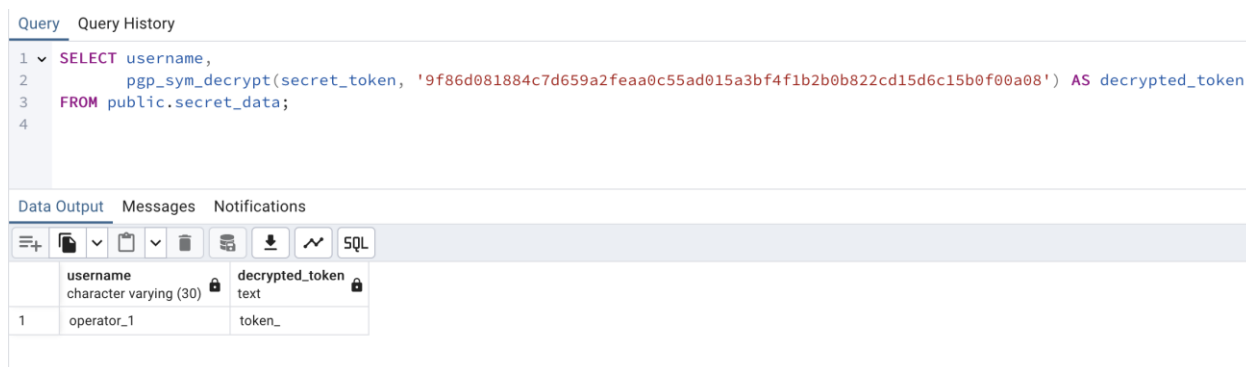


Рисунок 11 – Успешная попытка расшифровки

3.2.3 Задача №3 – разграничение доступа к БД

Задачи по разграничению доступа в БД:

- Создать роли для различных классов пользователей (например, операторы, мастера);
- Настроить привилегии для каждой роли в соответствии с принципом минимальных привилегий;
- Продемонстрировать работу системы разграничения доступа.

3.2.3.1 Создание ролей

Листинг 14 – Создание ролей

```

-- Роль для операторов
CREATE ROLE operator_role WITH
    NOLOGIN
    NOSUPERUSER
    NOCREATEDB
    NOCREATEROLE
    NOREPLICATION
    INHERIT;

-- Роль для мастеров
CREATE ROLE master_role WITH
    NOLOGIN
    NOSUPERUSER
    NOCREATEDB
    NOCREATEROLE
    NOREPLICATION
    INHERIT;

```

3.2.3.2 Настройка привилегий

Листинг 15 – Настройка привилегий

```
-- Привилегии для операторов
GRANT SELECT, INSERT, UPDATE ON public.orders TO
operator_role;
GRANT SELECT ON public.clients TO operator_role;

-- Привилегии для мастеров
GRANT SELECT, INSERT, UPDATE ON public.employees TO
master_role;
GRANT SELECT ON public.materials TO master_role;

-- Пользователь для оператора
CREATE ROLE operator_user WITH LOGIN PASSWORD
'operator_pass';
GRANT operator_role TO operator_user;

-- Пользователь для мастера
CREATE ROLE master_user WITH LOGIN PASSWORD 'master_pass';
GRANT master_role TO master_user;
```

3.2.3.3 Пример работы

Query

Query History

1

SET ROLE operator_user;

2

SELECT * FROM public.orders;

Data Output

Messages

Notifications

SQL

	order_id [PK] integer	creation_date date	completion_date date	status character varying (50)	total_cost numeric (10,2)	client_id integer	employee_id integer
1	2	2024-01-05	[null]	В процессе	3000.00	2	2
2	3	2024-01-10	[null]	Новый	2000.00	3	3
3	4	2024-02-01	[null]	Новый	4000.00	4	4
4	5	2024-02-05	2024-02-15	Завершен	3500.00	5	5
5	6	2024-02-10	[null]	В процессе	2500.00	6	6
6	7	2024-02-15	[null]	Новый	1500.00	7	7
7	8	2024-02-20	[null]	Новый	1800.00	8	8
8	9	2024-02-25	2024-03-01	Завершен	5000.00	9	9
9	10	2024-03-01	[null]	В процессе	2200.00	10	10
10	1	2024-01-01	2024-01-10	Завершен	5000.00	1	[null]
11	11	2024-02-02	[null]	Новый	2500.00	1	[null]

Рисунок 12 – Успешная попытка доступа



Рисунок 13 – Неуспешная попытка доступа

3.3 Вывод

В ходе выполнения работы были реализованы системы мониторинга, шифрования данных и разграничения доступа в базе данных. Эти механизмы обеспечивают безопасность данных и контроль доступа к информации в рамках СУБД

4 РАЗРАБОТКА ПРИЛОЖЕНИЯ

4.1 Цель

Цель работы: получение навыков использования серверных языков программирования, фреймворков для работы с БД, ORM-систем.

4.2 Ход работы

Задание:

1. На основе БД, созданной в предыдущих лабораторных работах, разработать сервис, который взаимодействует с разработанной БД.
2. Описать, каким образом происходит взаимодействие с базой данных, с помощью каких фреймворков или языков. Продемонстрировать примеры выборки, вставки, удаления данных из БД.
3. Описать основные функции сервиса, его назначение и структуру. Оценить методы защиты данных, реализованные в сервисе.

4.2.1 Обоснование выбора среды программирования и фреймворка

Для реализации сервиса взаимодействия с базой данных выбран язык программирования Python и фреймворк FastAPI. Все сервисы упакованы в Docker контейнеры для упрощения и безопасности разработки. Запуск производится посредством команды `docker-compose`. Фронтенд написан с использованием `html` и `JavaScript`.

Обоснование выбора:

- Python — простой и мощный язык, широко используемый для разработки веб-приложений и работы с базами данных;
- FastAPI — набирающий все большую популярность фреймворк для создания веб-приложений, который позволяет быстро разрабатывать API и веб-интерфейсы;

- Для работы с базой данных используется ORM SQLAlchemy, который упрощает взаимодействие с БД и позволяет избежать написания сложных SQL-запросов.

4.2.2 Структура

Всего 3 микросервиса:

- db: база данных PostgreSQL;
- api: основная логика бэкенда и фронтенда;
- nginx: веб-сервер для проксирования запросов.

4.2.2.1 Микросервис db

Вся инициализация базы данных из предыдущих лабораторных работ разбита на файлы с расширением sql. Также добавлена таблица users, где хранятся пользователи и их хэшированные пароли. Для достижения этого использовалась хэш-функция bcrypt.

Листинг 16 – Таблица пользователей

```
class User(Base):
    __tablename__ = "users"
    id = Column(Integer, primary_key=True, index=True)
    email = Column(String, unique=True, index=True)
    hashed_password = Column(String)
```

Листинг 17 – Создание хэша пароля в файле utils.py

```
from passlib.context import CryptContext #type: ignore

# Hashing algorithm for passwords
pwd_context = CryptContext(schemes=["bcrypt"],
    deprecated="auto")

# Function to hash passwords
def get_password_hash(password: str) -> str:
    return pwd_context.hash(password)

# Function to verify a hashed password
def verify_password(plain_password: str, hashed_password: str) -
    > bool:
    return pwd_context.verify(plain_password, hashed_password)
```

```

> psql -h localhost -p 5433 -U postgres -d repair_workshop
Пароль пользователя postgres:
psql (16.4 (Homebrew), сервер 15.12 (Debian 15.12-1.pgdg120+1))
Введите "help", чтобы получить справку.

repair_workshop=# \dt
                Список отношений
 Схема |          Имя          | Тип   | Владелец
-----+-----+-----+-----
public | clients               | таблица | postgres
public | employees              | таблица | postgres
public | main_log               | таблица | postgres
public | materials              | таблица | postgres
public | order_materials        | таблица | postgres
public | order_services         | таблица | postgres
public | orders                 | таблица | postgres
public | payments               | таблица | postgres
public | reports                | таблица | postgres
public | secret_data            | таблица | postgres
public | services                | таблица | postgres
public | users                  | таблица | postgres
(12 строк)

repair_workshop=#

```

Рисунок 14 – Доступ к базе данных

4.2.2.2 Микросервис api

Вся основная логика отражена в Приложении А. JWT формируется на основе почты пользователя.

Настроено:

- Вход и аутентификация с помощью JWT;
- Логирование с помощью logger;
- Взаимодействие с базой данных с помощью SQLAlchemy;
- Схемы и модели для базы данных.

Листинг 18 – Часть кода файла models.py

```

class User(Base):
    __tablename__ = "users"
    id = Column(Integer, primary_key=True, index=True)
    email = Column(String, unique=True, index=True)
    hashed_password = Column(String)

class Client(Base):
    __tablename__ = "clients"
    client_id = Column(Integer, primary_key=True,
index=True)

```

```

    full_name = Column(String, nullable=False)
    phone = Column(String, nullable=False)
    email = Column(String, nullable=True)

class Order(Base):
    __tablename__ = "orders"
    order_id = Column(Integer, primary_key=True, index=True)
    creation_date = Column(Date, nullable=False)
    completion_date = Column(Date, nullable=True)
    status = Column(String, nullable=False)
    total_cost = Column(DECIMAL, nullable=True)
    client_id = Column(Integer,
ForeignKey("clients.client_id"), nullable=False)
    employee_id = Column(Integer,
ForeignKey("employees.employee_id"), nullable=True)

    client = relationship("Client")
    employee = relationship("Employee")

```

Листинг 19 – Часть кода schemas.py

```

class ClientResponse(BaseModel):
    client_id: int
    full_name: str
    phone: str
    email: Optional[str] = None

    class Config:
        orm_mode = True
        from_attributes=True

class OrderResponse(BaseModel):
    order_id: int
    creation_date: date
    completion_date: Optional[date] = None
    status: str
    total_cost: Optional[Decimal] = None
    client_id: int

    class Config:
        orm_mode = True
        from_attributes=True

```

Листинг 20 – Dockerfile микросервиса api

```

FROM python:3.12

WORKDIR /api/

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

```

```
COPY . .
```

```
EXPOSE 8000
```

```
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port",  
"8000"]
```

4.2.2.3 Микросервис nginx

Настроено проксирование внутри контейнера и логирование.

Листинг 21 – default.conf

```
server {  
    listen 80;  
  
    location / {  
        proxy_pass http://api:8000;  
        proxy_set_header Host $host;  
        proxy_set_header X-Real-IP $remote_addr;  
    }  
  
    access_log /var/log/nginx/access.log;  
    error_log /var/log/nginx/error.log;  
}
```

4.2.2.4 docker-compose

Настроено:

- Запуск сервиса api строго после запуска контейнера база данных;
- Мэппинг томов для всех микросервисов для моментального контроля изменений;
- Мэппинг портов;
- Первоначальная инициализация базы данных в папке db.

Листинг 22 – docker-compose.yml

```
services:  
  db:  
    image: postgres:15  
    container_name: postgres-ISDB-lab4  
    environment:  
      POSTGRES_USER: ${POSTGRES_USER}  
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}  
      POSTGRES_DB: ${POSTGRES_DB}
```

```

ports:
  - "5433:5432"
volumes:
  - db_data:/var/lib/postgresql/data
  - ./db:/docker-entrypoint-initdb.d
healthcheck:
  test: ["CMD-SHELL", "pg_isready -U ${POSTGRES_USER} -d
${POSTGRES_DB}"]
  interval: 1s
  timeout: 5s
  retries: 10

api:
  build: ./api
  container_name: fastapi-ISBD-lab4
  volumes:
    - ./api:/api
  environment:
    SERVER_ID: SERVER-1
    DATABASE_URL:
postgresql://${POSTGRES_USER}:${POSTGRES_PASSWORD}@db:5432/${
POSTGRES_DB}
  depends_on:
    db:
      condition: service_healthy

nginx:
  image: nginx:latest
  container_name: nginx-ISDB-lab4
  ports:
    - "80:80"
  volumes:
    - ./nginx:/etc/nginx/conf.d
  depends_on:
    - api

volumes:
  db_data:

```

4.3 Взаимодействие с веб-приложением и базой данных

Запуск производится с помощью команды `docker-compose up`.

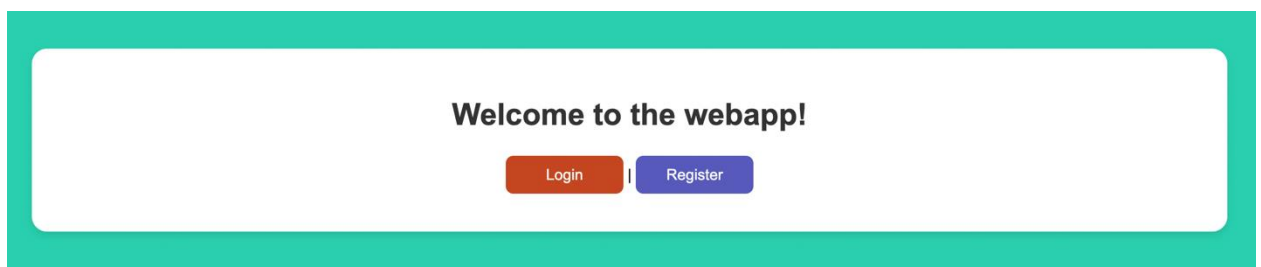
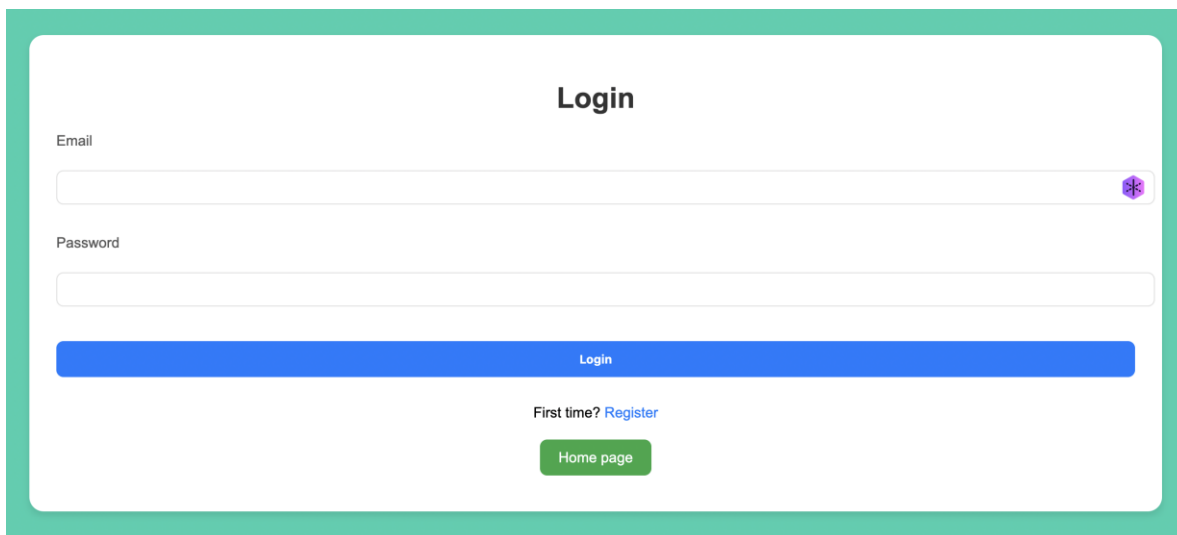
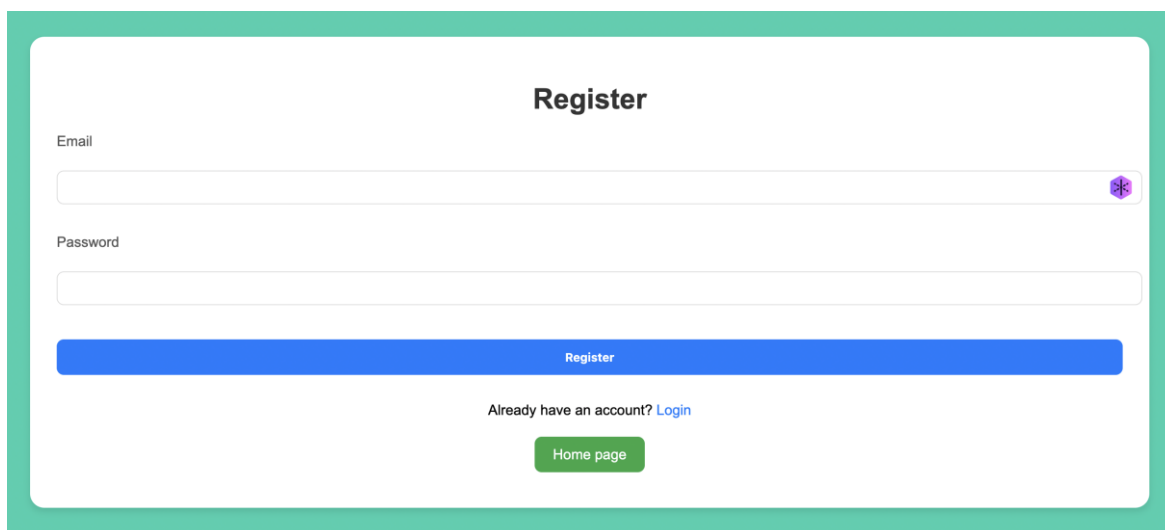


Рисунок 15 – Приветственная страница



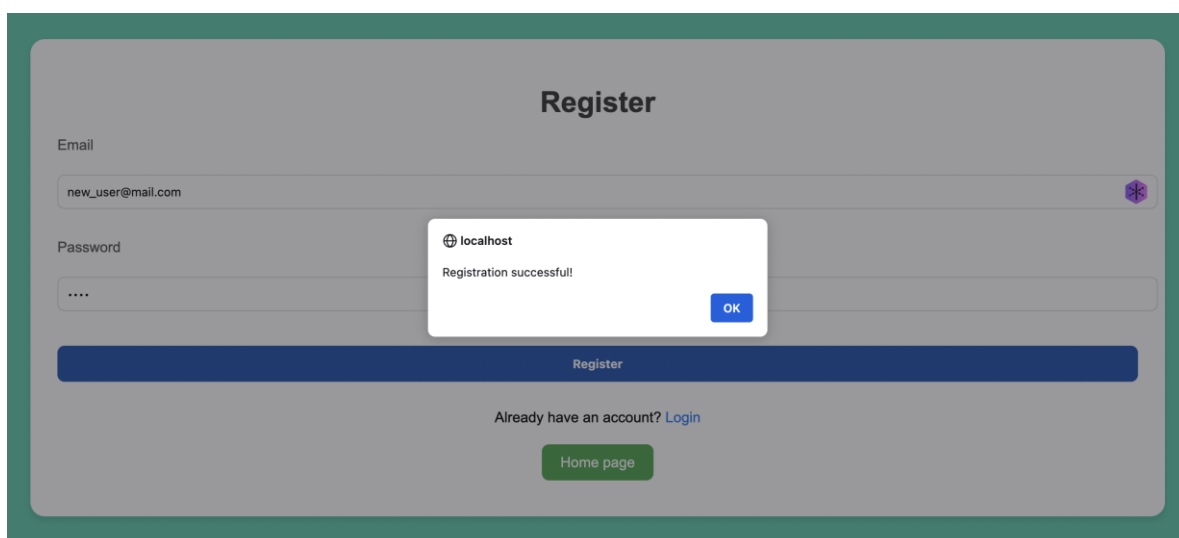
The image shows a login page with a white background and a green border. At the top, the word "Login" is centered in bold. Below it, there are two input fields: "Email" and "Password". The "Email" field has a purple icon on the right. Below the input fields is a blue button labeled "Login". At the bottom, there is a link "First time? Register" and a green button labeled "Home page".

Рисунок 16 – Страница входа



The image shows a register page with a white background and a green border. At the top, the word "Register" is centered in bold. Below it, there are two input fields: "Email" and "Password". The "Email" field has a purple icon on the right. Below the input fields is a blue button labeled "Register". At the bottom, there is a link "Already have an account? Login" and a green button labeled "Home page".

Рисунок 17 – Страница регистрации



The image shows the register page with a grey background and a dark green border. A modal window is displayed in the center, titled "localhost" with a sub-header "Registration successful!". The modal has an "OK" button. The background form is dimmed and shows the "Register" title, "Email" field (containing "new_user@mail.com"), "Password" field (containing "...."), a dark blue "Register" button, a link "Already have an account? Login", and a green "Home page" button.

Рисунок 18 – Окно после успешной регистрации

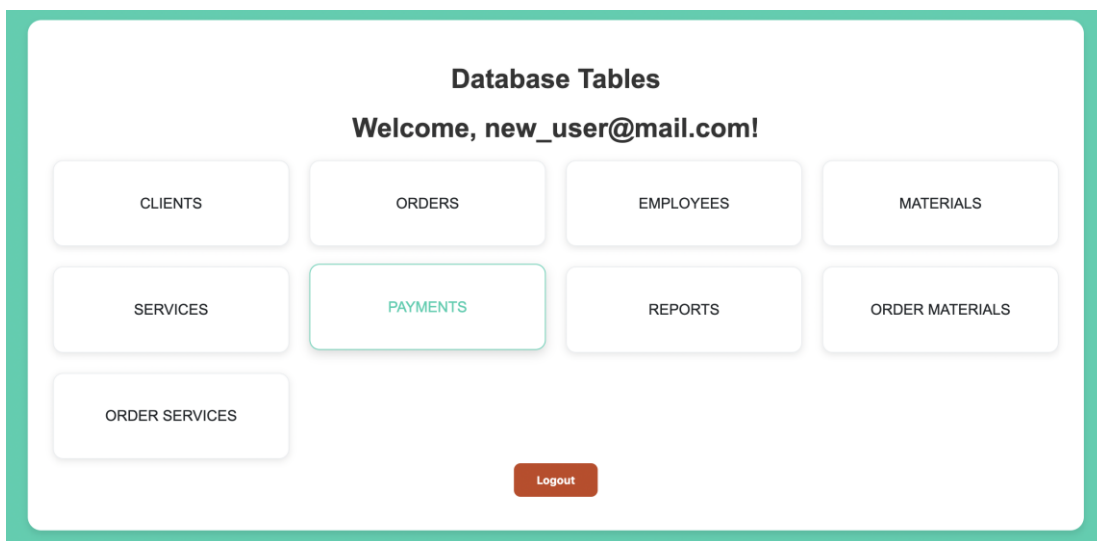


Рисунок 19 – Домашняя страница пользователя

Table: services

service_id	name	base_cost	Actions
1	Ремонт компьютера	1000.00	Edit Delete
2	Замена жесткого диска	500.00	Edit Delete
3	Установка программного обеспечения	300.00	Edit Delete
4	Чистка ноутбука	700.00	Edit Delete
5	Диагностика	400.00	Edit Delete

Рисунок 20 – Таблица services

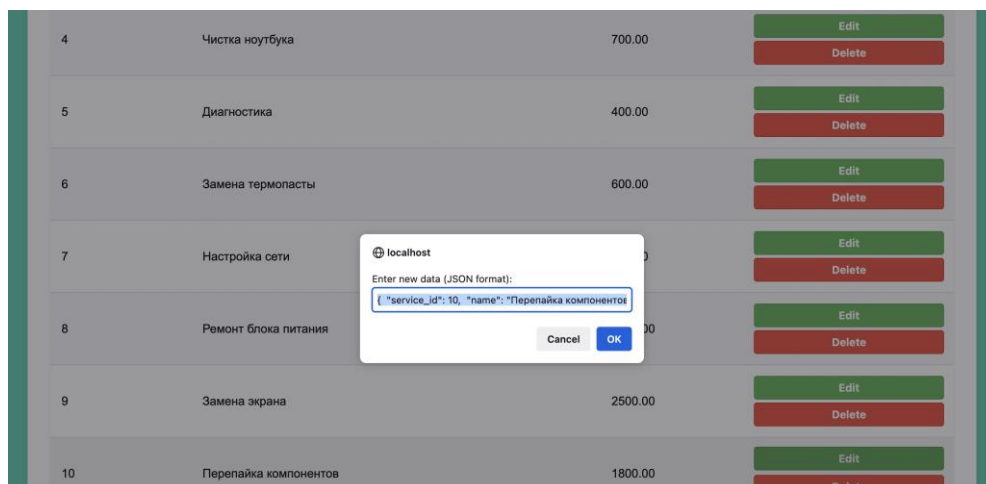


Рисунок 21 – Редактирование записи

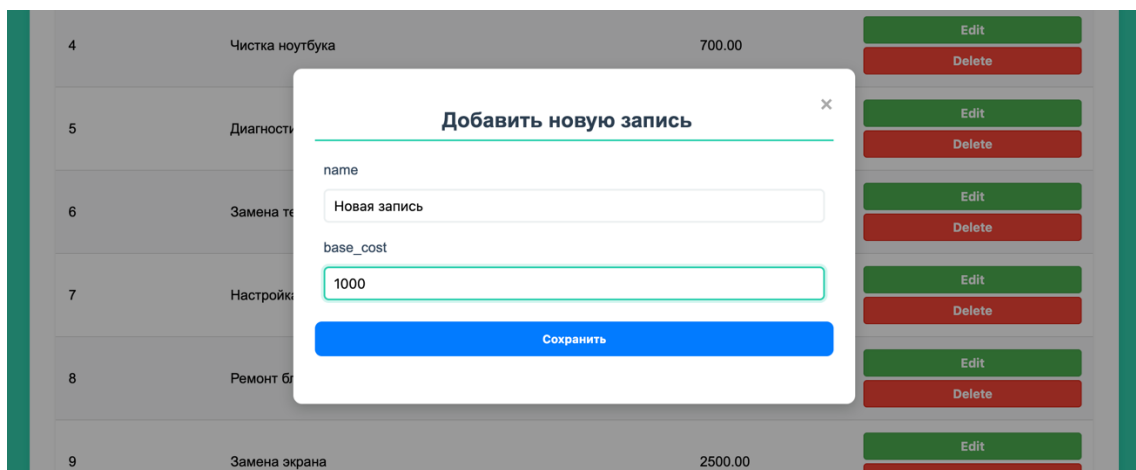


Рисунок 22 – Попытка добавления новой записи

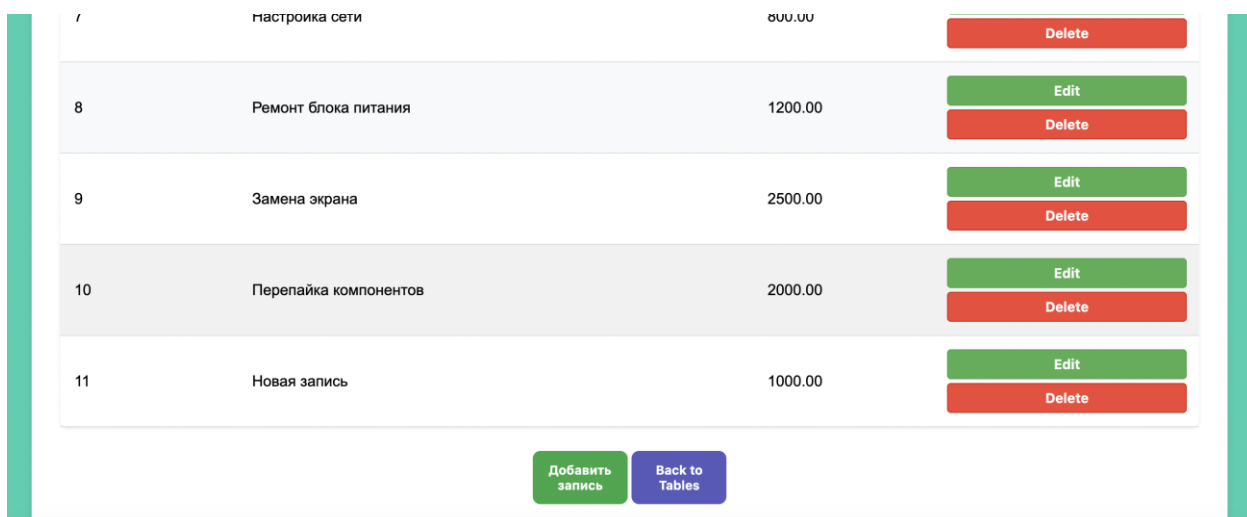


Рисунок 23 – Проверка добавления новой записи

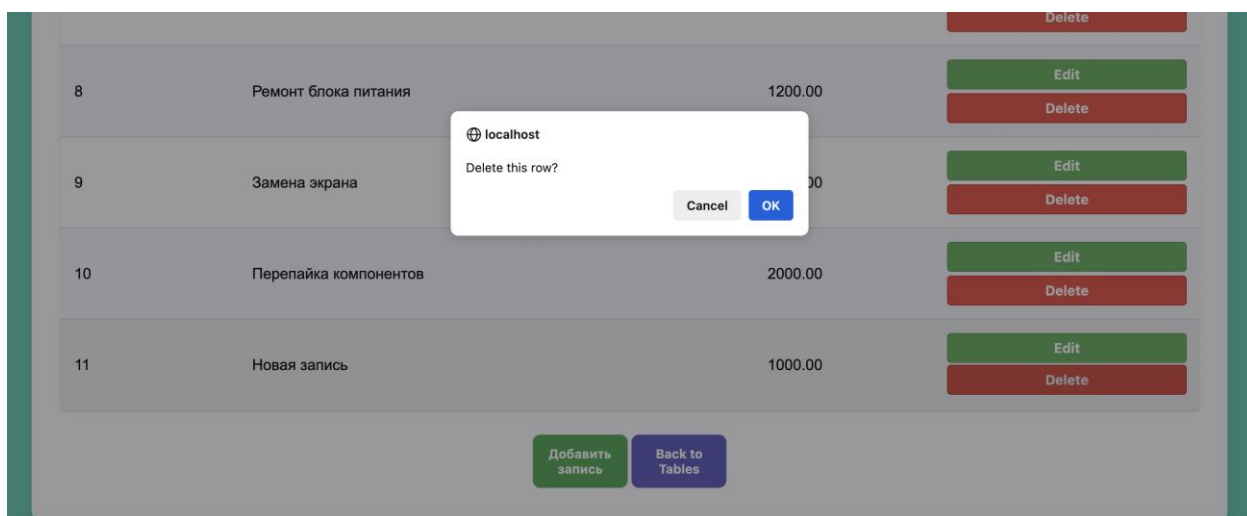


Рисунок 24 – Попытка удаление записи

			Delete
7	Настройка сети	800.00	Edit Delete
8	Ремонт блока питания	1200.00	Edit Delete
9	Замена экрана	2500.00	Edit Delete
10	Перепайка компонентов	2000.00	Edit Delete

Добавить запись Back to Tables

Рисунок 25 – Проверка удаления записи

4.4 Методы защиты данных

Аутентификация и авторизация:

- Используется JWT для доступа к веб-приложению. Также настроено управления сессиями пользователей.

Логирование действий:

- Все изменения в БД фиксируются в таблице-логе, что позволяет отслеживать действия пользователей.

Шифрование данных:

Для хранения паролей пользователей используется хэширование с помощью библиотеки bcrypt.

4.5 Вывод

В ходе выполнения лабораторной работы был разработан сервис для взаимодействия с базой данных ремонтной мастерской. Сервис поддерживает регистрацию и аутентификацию пользователей, просмотр, редактирование, удаление, добавление данных, а также логирование всех изменений в БД. Для обеспечения безопасности данных используются методы аутентификации на основе JWT, авторизации на основе JWT и шифрования.

5 РЕЗЕРВИРОВАНИЕ И ВОССТАНОВЛЕНИЕ БАЗЫ ДАННЫХ

5.1 Цель

Цель работы: получение навыков по резервированию и восстановлению базы данных.

5.2 Ход работы

Задание:

1. Создание резервной копии БД согласно выбранному расписанию.
2. Внесение случайных изменений в таблицы базы данных до момента создания контрольной точки.
3. Демонстрация процесса отката к последней контрольной точке и откат изменений, выполненных в пункте 2.
4. Анализ возможности просмотра изменений, которые были откачены, с помощью системы логирования СУБД или средств системы резервирования.

5.2.1 Создание резервной копии БД

Для создания резервной копии базы данных используется утилита `pg_dump`, которая входит в состав PostgreSQL.

Листинг 23 – Команда для создания резервной копии:

```
pg_dump -U postgres -F c -b -v -f backup.dump  
repair_workshop
```

Описание параметров:

- `-U postgres` — указание пользователя для подключения к БД.
- `-F c` — формат резервной копии (custom).
- `-b` — включение больших объектов в резервную копию.
- `-v` — вывод подробной информации о процессе.
- `-f backup.dump` — указание имени файла для резервной копии.

- `repair_workshop` — имя базы данных, для которой создается резервная копия.

```
➤ pg_dump -U postgres -F c -b -v -f backup.dump repair_workshop
pg_dump: последний системный OID: 16383
pg_dump: чтение расширений
pg_dump: выявление членов расширений
pg_dump: чтение схем
pg_dump: чтение пользовательских таблиц
pg_dump: чтение пользовательских функций
pg_dump: чтение пользовательских типов
pg_dump: чтение процедурных языков
pg_dump: чтение пользовательских агрегатных функций
pg_dump: чтение пользовательских операторов
pg_dump: чтение пользовательских методов доступа
pg_dump: чтение пользовательских классов операторов
pg_dump: чтение пользовательских семейств операторов
pg_dump: чтение пользовательских анализаторов текстового поиска
pg_dump: чтение пользовательских шаблонов текстового поиска
pg_dump: чтение пользовательских словарей текстового поиска
pg_dump: чтение пользовательских конфигураций текстового поиска
pg_dump: чтение пользовательских оболочек сторонних данных
pg_dump: чтение пользовательских сторонних серверов
pg_dump: чтение прав по умолчанию
pg_dump: чтение пользовательских правил сортировки
pg_dump: чтение пользовательских преобразований
pg_dump: чтение приведений типов
pg_dump: чтение преобразований
pg_dump: чтение информации о наследовании таблиц
pg_dump: чтение событийных триггеров
pg_dump: поиск таблиц расширений
pg_dump: поиск связей наследования
pg_dump: чтение информации о столбцах интересующих таблиц
pg_dump: поиск выражений по умолчанию для таблиц
pg_dump: поиск ограничений–проверок для таблиц
pg_dump: пометка наследованных столбцов в подтаблицах
pg_dump: чтение информации о секционировании
pg_dump: чтение индексов
pg_dump: пометка индексов в секционированных таблицах
pg_dump: чтение расширенной статистики
pg_dump: чтение ограничений
pg_dump: чтение триггеров
pg_dump: чтение правил перезаписи
pg_dump: чтение политик
pg_dump: чтение политик защиты на уровне строк
pg_dump: чтение публикаций
pg_dump: чтение информации о таблицах, включённых в публикации
pg_dump: чтение информации о схемах, включённых в публикации
pg_dump: чтение подписок
pg_dump: чтение больших объектов
pg_dump: чтение информации о зависимостях
pg_dump: сохранение кодировки (UTF8)
pg_dump: сохранение standard_conforming_strings (on)
pg_dump: сохранение search_path =
pg_dump: сохранение определения базы данных
pg_dump: выгрузка содержимого таблицы "public.clients"
pg_dump: выгрузка содержимого таблицы "public.employees"
pg_dump: выгрузка содержимого таблицы "public.main_log"
pg_dump: выгрузка содержимого таблицы "public.materials"
pg_dump: выгрузка содержимого таблицы "public.order_materials"
pg_dump: выгрузка содержимого таблицы "public.order_services"
pg_dump: выгрузка содержимого таблицы "public.orders"
pg_dump: выгрузка содержимого таблицы "public.payments"
pg_dump: выгрузка содержимого таблицы "public.reports"
pg_dump: выгрузка содержимого таблицы "public.secret_data"
pg_dump: выгрузка содержимого таблицы "public.services"
```

Рисунок 26 – Реализация команды `pg_dump`

5.2.2 Внесение случайных изменений в таблицы

Перед созданием контрольной точки внесем случайные изменения в таблицы базы данных.

Листинг 24 – Пример изменения данных в таблице "Заказы"

```
UPDATE orders SET status = 'Отменен' WHERE order_id = 1;
```

Листинг 25 – Пример удаления данных из таблицы "Клиенты"

```
DELETE FROM clients WHERE client_id = 2;
```

5.2.3 Создание контрольной точки и откат изменений

5.2.3.1 Создание контрольной точки

В PostgreSQL контрольная точка создается автоматически, но можно принудительно вызвать создание контрольной точки с помощью команды CHECKPOINT.

5.2.3.2 Восстановление из резервной копии

Для восстановления базы данных из резервной копии используется утилита pg_restore.

Листинг 26 – Команда для восстановления

```
pg_restore -U postgres -d repair_workshop -v backup.dump
```

Описание параметров:

- -U postgres — указание пользователя для подключения к БД;
- -d repair_workshop — имя базы данных, в которую восстанавливаются данные;
- -v — вывод подробной информации о процессе;
- backup.dump — имя файла резервной копии.

После восстановления базы данных изменения, внесенные в пункте 2, будут откачены.

5.2.4 Анализ откаченных изменений

Для анализа изменений, которые были откачены, можно использовать систему логирования, созданную в лабораторной работе №3.

Листинг 27 – Пример запроса к таблице-логу

```
SELECT * FROM main_log WHERE operation_type IN ('UPDATE',  
'DELETE');
```

Результат запроса:

- В таблице-логе будут зафиксированы все изменения, которые были внесены в базу данных до создания контрольной точки;

Это позволяет отследить, какие именно данные были изменены или удалены, и восстановить их вручную, если это необходимо

5.3 Вывод

В ходе выполнения данной работы были получены навыки по резервированию и восстановлению базы данных. Была создана резервная копия БД, внесены случайные изменения, продемонстрирован процесс отката к последней контрольной точке и проведен анализ откаченных изменений с помощью системы логирования.

ЗАКЛЮЧЕНИЕ

Выполненная серия лабораторных работ позволила студентам освоить основные принципы и методы проектирования, создания и защиты баз данных. Были рассмотрены различные аспекты работы с базами данных, начиная от моделирования и нормализации данных, заканчивая обеспечением безопасности и восстановлением данных после сбоев. Полученные навыки позволят эффективно применять современные технологии и инструменты в области разработки и администрирования информационных систем, что значительно повысит уровень профессиональной подготовки будущих специалистов в сфере информационных технологий.

СПИСОК ИСТОЧНИКОВ

1. Документация FastAPI [Электронный ресурс]. – URL: <https://fastapi.tiangolo.com/> (Дата обращения: 25.01.2025).
2. Документация Jinja [Электронный ресурс]. – URL: <https://jinja.palletsprojects.com/en/stable/> (Дата обращения: 25.01.2025).
3. Документация PostgreSQL – Создание ролей [Электронный ресурс]. – URL: <https://postgrespro.ru/docs/postgresql/9.6/sql-creatorole> (Дата обращения: 25.01.2025).
4. Третья нормальная форма [Электронный ресурс]. – URL: https://ru.hexlet.io/courses/rdb-basics/lessons/3nf/theory_unit (Дата обращения: 10.10.2024).
5. Нормализация отношений [Электронный ресурс]. – URL: <https://habr.com/ru/articles/254773/> (Дата обращения: 10.10.2024).
6. Резервное копирование в PostgreSQL [Электронный ресурс]. – URL: <https://selectel.ru/blog/postgresql-backup-tools/> (Дата обращения: 25.01.2025).
7. Методические указания по выполнению научно-исследовательской и выпускной квалификационной работы магистрантов [Электронный ресурс]. – URL: https://books.ifmo.ru/book/2728/metodicheskie_ukazaniya_po_vypolneniyu_na_uchnoissledovatel'skoy_i_vypusknoy_kvalifikacionnoy_raboty_magistrantov:_uchebno-metodicheskoe_posobie..htm. (Дата обращения: 25.01.2025).

ПРИЛОЖЕНИЕ А

Листинг 28 – Программный код main.py (микросервис api)

```
from fastapi import FastAPI, Depends, HTTPException, status, Request, Query
# type: ignore
from fastapi.security import OAuth2PasswordRequestForm # type: ignore
from fastapi.responses import HTMLResponse, RedirectResponse # type: ignore
from fastapi.staticfiles import StaticFiles # type: ignore
from fastapi.templating import Jinja2Templates # type: ignore
from sqlalchemy.orm import Session, joinedload # type: ignore
from sqlalchemy.exc import SQLAlchemyError # type: ignore
from sqlalchemy import text # type: ignore
from fastapi.middleware.cors import CORSMiddleware # type: ignore
from typing import List, Optional, Dict, Any
from decimal import Decimal
import logging

from database import engine, Base, get_db
from schemas import UserCreate, Token, UserResponse
from utils import verify_password, get_password_hash
from auth import create_access_token, get_current_user
from models import User, Client, Order, Employee, Material, Service, Payment,
Report, OrderMaterial, OrderService
from schemas import ClientResponse, OrderResponse, EmployeeResponse,
MaterialResponse, ServiceResponse, PaymentResponse, ReportResponse,
OrderMaterialResponse, OrderServiceResponse, PaginatedResponse
import models

# create app
app = FastAPI()
app.mount("/static", StaticFiles(directory="static"), name="static")
templates = Jinja2Templates(directory="templates")

# create database tables
Base.metadata.create_all(bind=engine)

# create logger
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# ===== CONST
# =====
PRIMARY_KEYS = {
    "clients": "client_id",
    "orders": "order_id",
    "employees": "employee_id",
    "materials": "material_id",
    "services": "service_id",
    "payments": "payment_id",
    "reports": "report_id",
    "order_materials": ["order_id", "material_id"],
    "order_services": ["order_id", "service_id"]
}
```



```

MODELS_MAPPING = {
    "clients": Client,
    "orders": Order,
    "employees": Employee,
    "materials": Material,
    "services": Service,
    "payments": Payment,
    "reports": Report,
    "order_materials": OrderMaterial,
    "order_services": OrderService,
}

SCHEMAS_MAPPING = {
    "clients": ClientResponse,
    "orders": OrderResponse,
    "employees": EmployeeResponse,
    "materials": MaterialResponse,
    "services": ServiceResponse,
    "payments": PaymentResponse,
    "reports": ReportResponse,
    "order_materials": OrderMaterialResponse,
    "order_services": OrderServiceResponse,
}

# ===== LOGIN LOGIC
=====
@app.get("/login", response_class=HTMLResponse)
async def login_page(request: Request):
    return templates.TemplateResponse(request=request, name="login.html",
context={"request": request})

@app.post("/login/", response_model=Token)
def login(form_data: OAuth2PasswordRequestForm = Depends(), db: Session =
Depends(get_db)):
    user = db.query(User).filter(User.email == form_data.username).first()

    if not user or not verify_password(form_data.password,
user.hashed_password):
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Incorrect email or password",
            headers={"WWW-Authenticate": "Bearer"},
        )

    # generate JWT token for authentication
    access_token = create_access_token(data={"sub": user.email})
    return {"access_token": access_token, "token_type": "bearer"}

# ===== REGISTRATION LOGIC
=====
@app.get("/register", response_class=HTMLResponse)
async def register_page(request: Request):
    return templates.TemplateResponse(request=request, name="register.html",
context={"request": request})

@app.post("/register/", response_model=Token)
def register(user: UserCreate, db: Session = Depends(get_db)):
    # check if user already exists
    db_user = db.query(User).filter(User.email == user.email).first()
    if db_user:
        raise HTTPException(status_code=400, detail="Email already
registered")

```

```

# hash the password and create the user
hashed_password = get_password_hash(user.password)
new_user = User(email=user.email, hashed_password=hashed_password)
db.add(new_user)
db.commit()
db.refresh(new_user)

# generate JWT token
access_token = create_access_token(data={"sub": new_user.email})
return {"access_token": access_token, "token_type": "bearer"}

# ===== TABLES LOGIC
=====
@app.get("/tables", response_class=HTMLResponse)
async def tables_page(request: Request):
    return templates.TemplateResponse(request=request, name="tables.html",
context={"request": request})

@app.get("/tables/{table_name}", response_class=HTMLResponse)
async def table_page(request: Request, table_name: str):
    return templates.TemplateResponse("table.html", {"request": request,
"table_name": table_name})

# ===== API TABLES LOGIC
=====
@app.get("/api/tables")
def get_available_tables():
    return {
        "tables": [
            "clients", "orders", "employees", "materials",
            "services", "payments", "reports",
            "order_materials", "order_services"
        ]
    }

@app.get("/api/tables/{table_name}", response_model=PaginatedResponse)
def get_table_data(table_name: str, db: Session = Depends(get_db)):
    try:
        model = MODELS_MAPPING.get(table_name)
        if not model:
            raise HTTPException(status_code=404, detail="Table not found")
        if table_name == "clients":
            data = db.query(model).order_by(model.client_id).all()
        elif table_name == "orders":
            data = db.query(model).order_by(model.order_id).all()
        elif table_name == "employees":
            data = db.query(model).order_by(model.employee_id).all()
        elif table_name == "materials":
            data = db.query(model).order_by(model.material_id).all()
        elif table_name == "services":
            data = db.query(model).order_by(model.service_id).all()
        elif table_name == "payments":
            data = db.query(model).order_by(model.payment_id).all()
        else:
            data = db.query(model).all()
        total_count = db.query(model).count()
        schema = SCHEMAS_MAPPING[table_name]
        serialized_data = [schema.from_orm(item) for item in data]

        return {"data": serialized_data, "totalCount": total_count}

    except AttributeError:

```

```

        raise HTTPException(status_code=404, detail="Table not found")
    except Exception as e:
        logger.error(f"exception: {str(e)}")
        raise HTTPException(status_code=400, detail=str(e))

# ===== CREATE & UPDATE & DELETE =====
@app.post("/api/tables/{table_name}")
def add_row(table_name: str, data: dict, db: Session = Depends(get_db)):
    try:
        model = MODELS_MAPPING.get(table_name)
        if not model:
            raise HTTPException(status_code=404, detail="Table not found")

        new_row = model(**data)
        db.add(new_row)
        db.commit()
        db.refresh(new_row)
        return {"message": "Row added successfully"}
    except SQLAlchemyError as e:
        db.rollback()
        raise HTTPException(status_code=400, detail=str(e._message))
    except Exception as e:
        raise HTTPException(status_code=500, detail="Internal server error")

@app.patch("/api/tables/{table_name}/{row_id}")
def update_row(table_name: str, row_id: str, data: dict, db: Session =
Depends(get_db)):
    try:
        model = MODELS_MAPPING.get(table_name)
        if not model:
            raise HTTPException(status_code=404, detail="Table not found")

        pk = PRIMARY_KEYS.get(table_name)

        # Обработка составных ключей
        if isinstance(pk, list):
            key_parts = row_id.split('-')
            if len(key_parts) != len(pk):
                raise HTTPException(status_code=400, detail="Invalid
composite key")

            filters = [getattr(model, pk[i]) == key_parts[i] for i in
range(len(pk))]
        else:
            filters = [getattr(model, pk) == row_id]

        row = db.query(model).filter(*filters).first()

        if not row:
            raise HTTPException(status_code=404, detail="Row not found")

        for key, value in data.items():
            if hasattr(model, key):
                setattr(row, key, value)
            else:
                raise HTTPException(status_code=400, detail=f"Invalid field:
{key}")

        db.commit()
        return {"message": "Row updated successfully"}

    except SQLAlchemyError as e:

```

```

        db.rollback()
        raise HTTPException(status_code=400, detail=str(e))
    except Exception as e:
        raise HTTPException(status_code=500, detail="Internal server error")

@app.delete("/api/tables/{table_name}/{row_id}")
def delete_row(table_name: str, row_id: int, db: Session = Depends(get_db)):
    try:
        model = MODELS_MAPPING.get(table_name)
        if not model:
            raise HTTPException(status_code=404, detail="Table not found")

        pk = PRIMARY_KEYS.get(table_name)
        if not pk:
            raise HTTPException(status_code=400, detail="Invalid table")

        row = db.query(model).filter(getattr(model, pk) == row_id).first()
        if not row:
            raise HTTPException(status_code=404, detail="Row not found")

        db.delete(row)
        db.commit()
        return {"message": "Row deleted successfully"}
    except SQLAlchemyError as e:
        db.rollback()
        raise HTTPException(status_code=400, detail=str(e._message))
    except Exception as e:
        raise HTTPException(status_code=500, detail="Internal server error")

# ===== ANOTHER LOGIC
# =====
@app.get("/home", response_class=HTMLResponse)
async def read_root(request: Request):
    # return {"message": "Welcome to the FastAPI Auth Demo"}
    return templates.TemplateResponse(request=request, name="home.html",
    context={"request": request})

@app.get("/")
def redirect_to_home():
    # redirect
    return RedirectResponse(url="/home", status_code=301)

# Protected route to check JWT token validity
@app.get("/me", response_model=UserResponse)
def access_cabinet(current_user: User = Depends(get_current_user)):
    return {"message": f"Welcome to your cabinet, {current_user.email}!",
    "user": current_user}

```