МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«РЯЗАНСКИЙ ГОСУДАРСТВЕННЫЙ РАДИОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТИМЕНИ В.Ф. УТКИНА»

Кафедра Автоматизированных систем управления

ПРЕДДИПЛОМНАЯ ПРАКТИКА

по теме

«Разработка серверной части информационной системы центра исследования строительных конструкций»

Период прохождения:

21.04.2025 - 21.05.2025

Выполнил:

студент группы 135 Жильцов В.А.

Руководитель работы:

доцент кафедры АСУ, к.т.н.

Брянцев А.А.

Рязань 2025

Федеральное государственное бюджетное образовательное учреждение высшего образования «Рязанский государственный радиотехнический университет имени В.Ф. Уткина»

«Утверждаю»
Заведующий кафедрой
_____ Холопов С.И.
_____
(Ф.И.О, подпись)
«21» 04 2025 г.

ЗАДАНИЕ НА ПРЕДДИПЛОМНУЮ ПРАКТИКУ БАКАЛАВРА

Студент Жильцов Владимир Александрович гр. 135
(Ф.И.О. бакалавра)

Направление подготовки: 09.03.02 – Информационные системы и технологии

Направленность: Информационные системы и технологии

Трудоёмкость практики: 4 недели, 216 часов

Научный руководитель: Брянцев Андрей Анатольевич, к.т.н., доцент, доцент Рязанского государственного радиотехнического университета имени В.Ф. Уткина
(Ф.И.О., должность, учёное звание)

Тема практики: разработка серверной части информационной системы центра исследования строительных конструкций

Цель практики: реализация серверной части информационной системы центра исследования строительных конструкций

Задачи практики:

1) настройка подключения базы данных;
2) создание сущностных классов;
3) реализация серверной части информационной системы на языке C#.

Планируемые результаты: серверная часть информационной системы центра исследований строительных конструкций.

Студент _____ Жильцов В.А.
Научный руководитель _____ Брянцев А.А.
Руководитель от предприятия _____ Болонин И.В.

Оглавление

Введение

Преддипломная практика проходила в организации ООО “Центр исследования строительных конструкций и материалов”.

Основной вид деятельности организации - выполняет полный комплекс работ, начиная с обращения потенциального заказчика и создания заявки на обследование строительной конструкций, проведения самого обследования и составления отчета по проведенному обследованию.

Целью данной работы является реализация серверной части информационной системы центра исследования строительных конструкций.

Для достижения цели требуется решить следующие задачи:

1. настройка подключения базы данных;
2. создание сущностных классов;
3. реализация серверной части информационной системы на языке C#.

1. Анализ предметной области

Компании, занимающиеся обследованием строительных конструкций, играют важную роль в обеспечении безопасности и надежности зданий и сооружений [1]. Эти компании проводят всестороннюю оценку состояния строительных объектов, выявляют возможные дефекты и проблемы, а также разрабатывают рекомендации по их устранению. Благодаря своей деятельности, такие компании помогают предотвратить аварии и разрушения, обеспечивая тем самым безопасность людей и долговечность строений.

Основные аспекты центров исследования строительных конструкций и материалов:

Цели и задачи: компании, занимающиеся обследованием строительных конструкций, направлены на всестороннее изучение и оценку состояния зданий и сооружений. Задачи включают в себя выявление дефектов, оценку прочности и надежности конструкций, разработку рекомендаций по их устранению, а также обеспечение безопасности эксплуатации объектов.

Методы обследования: обследование строительных конструкций базируется на использовании современных технологий и инструментов. Применяются как визуальные методы осмотра, так и инструментальные методы диагностики, включая ультразвуковое, радиографическое и магнитное обследование. Комплексный подход позволяет детально изучить состояние конструкции, выявить скрытые дефекты и провести точную оценку.

Целью курсового проекта является инжиниринг базы данных информационной системы центра исследований строительных конструкций, которая обеспечит автоматизацию технических процессов, повысит уровень удобства для клиентов, упростит работу сотрудников и увеличит безопасность данных.

База данных будет использоваться:

1. Со стороны клиента, чтобы:

- клиент мог ознакомиться с предоставляемыми услугами;
- оформить заявку на обследование требуемой конструкции;
- добавлять требуемые ему услуги для обследования;
- просматривать отчет об обследовании;
- просматривать и утверждать договор.

2. Сотрудниками центра исследования для того, чтобы:

- обрабатывать заявки клиентов на обследование;
- контролировать прибыль с заказа, организации или исполнительных бригад;

- вести отчетность по активным заявкам.

3. Администратором системы, который:

- контролирует техническую работу системы;
- управляет пользователями и сотрудниками;
- Отменяет транзакции.

В системе установим 3 уровня доступа:

1) для клиента (ограниченный) – просмотр информации о услугах, подача заявок, просмотр отчетов по своим заявкам;

2) для сотрудника (расширенный) – доступ к заявкам клиентов, ведение учета заявок, добавление отчетов в базу;

3) для администратора – управление данными, настройка системы, контроль пользователей и безопасности.

2. Определение атрибутов сущностей и их ключей

На основании проведенного анализа предметную область «Центр исследования строительных конструкций» можно описать следующими сущностями: «Заказчик», «Сотрудник», «Пользователь», «Роль», «Должность», «Адрес», «Организация», «Объект обследования», «Статус заявки», «Заявка на обследование», «Бригада», «Каталог услуг», «Выбранные услуги», «Отчет об обследовании», «Договор». Результат представлен в таблице 1.

Таблица 1 – Список сущностей

№	Название	Значение
1	Заказчик	Информация о заказчиках
2	Сотрудник	Информация о сотрудниках
3	Пользователь	Информация о пользователях
4	Роль	Содержит информацию о возможной роли пользователя
5	Должность	Информация о должностях, которые могут иметь сотрудники
6	Адрес	Содержит информацию обо всех адресах объектов обследования
7	Организация	Содержит информацию обо всех организациях, для которых происходили обследования
8	Объект обследования	Содержит информацию обо всех объектах обследования
9	Статус заявки	Содержит информацию о возможные статусы заявки
10	Заявка на обследование	Информация о заявках на обследование
11	Бригада	Информация о бригадах, занимающихся обследованием
12	Каталог услуг	Информация о предоставляемых услугах
13	Выбранные услуги	Содержит информацию о выбранных услугах в каждой заявке
14	Отчет об обследовании	Содержит информацию об отчете
15	Договор	Информацию о договорах

Описание атрибутов для каждой сущности приведены в таблицах 2-16.

Таблица 2 – Атрибуты сущности «Заказчик»

Ключевое поле	Название	Тип данных	Допустимость NULL	Назначение
Pk (primary key)	Идентификатор заказчика	Числовой	Нет	Уникальный идентификатор
	ФИО	Строковый	Нет	Определяет ФИО заказчика
	Номер телефона	Строковый	Нет	Определяет номер телефона заказчика
	Адрес электронной почты	Строковый	Нет	Определяет адрес электронной почты заказчика

Таблица 3 – Атрибуты сущности «Сотрудник»

Ключевое поле	Название	Тип данных	Допустимость NULL	Назначение
Pk (primary key)	Идентификатор сотрудника	Числовой	Нет	Уникальный идентификатор
	ФИО	Строковый	Нет	Определяет ФИО сотрудника
	Номер телефона	Строковый	Нет	Определяет номер телефона сотрудника
	Адрес электронной почты	Строковый	Нет	Определяет адрес электронной почты сотрудника
Fk (foreign key)	Идентификатор должности	Числовой	Нет	Определяет занимаемую сотрудником должность
Fk (foreign key)	Идентификатор бригады	Числовой	Да	Определяет бригаду, в которой состоит сотрудник

Таблица 4 – Атрибуты сущности «Пользователь»

Ключевое поле	Название	Тип данных	Допустимость NULL	Назначение
Pk (primary key)	Идентификатор пользователя	Числовой	Нет	Уникальный идентификатор
	Электронная почта	Строковый	Нет	Определяет адрес электронной почты пользователя
	Зашифрованный пароль	Строковый	Нет	Определяет пароль пользователя
Fk (foreign key)	Идентификатор роли	Числовой	Нет	Определяет роль, которую имеет пользователь
Fk (foreign key)	Идентификатор заказчика	Числовой	Да	Определяет заказчика если он является пользователем
Fk (foreign key)	Идентификатор сотрудника	Числовой	Да	Определяет сотрудника если он является пользователем

Таблица 5 – Атрибуты сущности «Роль»

Ключевое поле	Название	Тип данных	Допустимость NULL	Назначение
Pk (primary key)	Идентификатор роли	Числовой	Нет	Уникальный идентификатор
	Наименование роли	Строковый	Нет	Определяет наименование роли

Таблица 6 – Атрибуты сущности «Должность»

Ключевое поле	Название	Тип данных	Допустимость NULL	Назначение
Pk (primary key)	Идентификатор должности	Числовой	Нет	Уникальный идентификатор
	Наименование должности	Строковый	Нет	Определяет наименование должности

Таблица 7 – Атрибуты сущности «Адрес»

Ключевое поле	Название	Тип данных	Допустимость NULL	Назначение
Pk (primary key)	Идентификатор адреса	Числовой	Нет	Уникальный идентификатор
	Наименование города	Строковый	Нет	Определяет наименование города, в котором находится объект обследования
	Наименование улицы	Строковый	Нет	Определяет наименование улицы, на которой находится объект обследования
	Номер	Строковый	Нет	Определяет номер дома или строения на определенной улице

Таблица 8 – Атрибуты сущности «Организация»

Ключевое поле	Название	Тип данных	Допустимость NULL	Назначение
Pk (primary key)	Идентификатор организации	Числовой	Нет	Уникальный идентификатор
	Наименование организации	Строковый	Нет	Содержит наименование организации, для которой проводится обследование
	ИНН	Строковый	Нет	Идентификационный номер налогоплательщика

Таблица 9 – Атрибуты сущности «Объект обследования»

Ключевое поле	Название	Тип данных	Допустимость NULL	Назначение
Pk (primary key)	Идентификатор объекта	Числовой	Нет	Уникальный идентификатор
Fk (foreign key)	Идентификатор адреса	Числовой	Нет	Определяет по какому адресу находится объект обследования
Fk (foreign key)	Идентификатор организации	Числовой	Нет	Определяет к какой организации относится объект обследования
	Площадь объекта	Числовой	Нет	Определяет какую площадь занимает объект обследования

Таблица 10 – Атрибуты сущности «Статус заявки»

Ключевое поле	Название	Тип данных	Допустимость NULL	Назначение
Pk (primary key)	Идентификатор статуса заявки	Числовой	Нет	Уникальный идентификатор
	Вид статуса	Числовой	Нет	Определяет вид статуса, который возможен в заявке

Таблица 11 – Атрибуты сущности «Заявка на обследование»

Ключевое поле	Название	Тип данных	Допустимость NULL	Назначение
Pk (primary key)	Идентификатор заявки	Числовой	Нет	Уникальный идентификатор
Fk (foreign key)	Идентификатор заказчика	Числовой	Нет	Определяет какой заказчик оставил заявку на обследование

Продолжение таблицы 11

Fk (foreign key)	Идентификатор бригады	Числовой	Нет	Определяет какая бригада была назначена на обследование
	Дата принятия заявки	Дата	Нет	Определяет, когда дату, когда была принята заявка
Fk (foreign key)	Идентификатор статуса заявки	Числовой	Нет	Определяет этап, на котором находится заявка
	Дата начала обследования	Дата	Да	Определяет дату, когда начнется обследование
	Дата окончания обследования	Дата	Да	Определяет дату, когда закончится обследование

Таблица 12 – Атрибуты сущности «Бригада»

Ключевое поле	Название	Тип данных	Допустимость NULL	Назначение
Pk (primary key)	Идентификатор бригады	Числовой	Нет	Уникальный идентификатор
	Наименование бригады	Строковый	Нет	Определяет наименование бригады

Таблица 13 – Атрибуты сущности «Каталог услуг»

Ключевое поле	Название	Тип данных	Допустимость NULL	Назначение
Pk (primary key)	Идентификатор услуги	Числовой	Нет	Уникальный идентификатор
	Наименование услуги	Строковый	Нет	Определяет наименование услуги

Продолжение таблицы 13

	Стоимость услуги	Числовой	Нет	Определяет стоимость оказания услуги за единицу объема выполненной работы
	Единица измерения	Строковый	Нет	Определяет единицу измерения услуги
	Описание услуги	Строковый	Нет	Содержит информацию об услуги

Таблица 14 – Атрибуты сущности «Выбранные услуги»

Ключевое поле	Название	Тип данных	Допустимость NULL	Назначение
Pk (primary key)	Идентификатор выбранной услуги	Числовой	Нет	Уникальный идентификатор
Fk (foreign key)	Идентификатор услуги	Числовой	Нет	Определяет выбранную услугу
Fk (foreign key)	Идентификатор заявки	Числовой	Нет	Определяет заявку, к которой относится выбранная услуга
	Объем	Числовой	Нет	Определяет объем, который требуется выполнить
	Стоимость за услугу	Числовой	Нет	Определяет общую стоимость за оказание выбранной услуги и указанный объем

Таблица 15 – Атрибуты сущности «Отчет об обследовании»

Ключевое поле	Название	Тип данных	Допустимость NULL	Назначение
Pk (primary key)	Идентификатор отчета	Числовой	Нет	Уникальный идентификатор
Fk (foreign key)	Идентификатор заявки	Числовой	Нет	Определяет заявку, к которой относится отчет
Fk (foreign key)	Идентификатор сотрудника	Числовой	Нет	Определяет сотрудника, который сформировал отчет
	Отчет	Строковый	Нет	Содержит путь к файлу отчета

Таблица 16 – Атрибуты сущности «Договор»

Ключевое поле	Название	Тип данных	Допустимость NULL	Назначение
Pk (primary key)	Идентификатор договора	Числовой	Нет	Уникальный идентификатор
Fk (foreign key)	Идентификатор заявки	Числовой	Нет	Определяет заявку, которая относится к договору
Fk (foreign key)	Идентификатор отчета	Числовой	Нет	Определяет отчет, который относится к договору
Fk (foreign key)	Идентификатор сотрудника	Числовой	Нет	Определяет сотрудника, который сформировал договор
	Дата создания договора	Дата	Нет	Определяет дату создания договора
	Подтверждение	Логический	Нет	Определяет статус договора
	Стоимость оказания услуг	Числовой	Нет	Определяет сумму, которую необходимо оплатить заказчику за оказанные услуги

2.1 Выявление связей между сущностями

На основании описания предметной области и списка атрибутов из таблиц 2-13 описываются классы сущностей и их свойства, устанавливаются существующие связи между ними и приводится обоснование типов связей (1:1, 1:N и т. д.). Результат представлен в Таблице 17.

Таблица 17 – Список связей

№	Сущности, участвующие в связи	Тип связи	Обоснование
1	Бригада - Сотрудник	1:N	Один сотрудник может состоять в одной бригаде, но в бригаде может состоять несколько сотрудников
2	Должность - Сотрудник	1:N	Один сотрудник может иметь только одну должность, но несколько сотрудников могут иметь одинаковую должность
3	Адрес – Объект обследования	1:N	На одном адресе может быть несколько объектов обследования, но у одного объекта обследования может быть только один адрес
4	Заказчик - Пользователь	1:1	У одного заказчика может быть только один пользователь, у одного пользователя может быть только один заказчик
5	Сотрудник - Пользователь	1:1	У одного Сотрудника может быть только один пользователь, у одного пользователя может быть только один Сотрудник
6	Роль - Пользователь	1:N	У одного Пользователя может быть только одна роль, но у одной роли может быть несколько пользователей
7	Организация – Объект обследования	1:N	У одной организации может быть несколько объектов обследования, но у одного объекта обследования может быть только одна организация

Продолжение таблицы 17

8	Заказчик – Заявка на обследование	1:N	У одного заказчика может быть несколько заявок на обследование, но у заявки на обследование может быть только один заказчик
9	Бригада – Заявка на обследование	1:N	У одной бригады может быть несколько заявок на обследование, но у заявки на обследование может быть только одна бригада
10	Статус заявки на обследование - Заявка на обследование	1:N	С одним статусом заявки на обследование может быть несколько заявок на обследование, но у заявки на обследование может быть только один статус
11	Объект – Заявка на обследование	1:N	У одного объекта обследования может быть несколько заявок на обследование, но у заявки на обследование может быть только один объект обследования
12	Каталог услуг – Выбранные услуги	1:N	У одной услуги из каталога, может быть, несколько выбранных услуг, но у одной выбранной услуги может быть только одна услуга из каталога
13	Заявка на обследование – Выбранные услуги	1:N	У одной заявки на обследование, может быть, несколько выбранных услуг, но у одной выбранной услуги может быть только одна заявка на объект
14	Заявка на обследование – Отчет об обследовании	1:1	У одной заявки, может быть, только один отчет об обследовании и у отчета об обследовании может быть только одна заявка на обследование

Продолжение таблицы 17

15	Заявка на обследование - Договор	1:1	У одной заявки, может быть, только один договор и у договора может быть только одна заявка на обследование
16	Отчет об обследовании - договор	1:1	У одного отчета об обследовании, может быть, только один договор и у договора может быть только один отчет об обследовании
17	Сотрудник - Отчет об обследовании	1:N	Один сотрудник может сделать несколько отчетов об обследовании, но у одного отчета об обследовании может быть только один сотрудник
18	Сотрудник - договор	1:N	Один сотрудник может сделать несколько договоров, но у одного договора может быть только один сотрудник

3. Разработка глобальных моделей

3.1 Разработка глобальной логической модели

Глобальная логическая модель – представляет из себя слияние всех локальных логических моделей. Она описывает взаимосвязи между сущностями и их атрибутами, а также устанавливает правила работы с данными на концептуальном уровне. Глобальная логическая модель используется для обеспечения согласованности данных в масштабах всей информационной системы и формирования основы для дальнейшей реализации базы данных [2].

Разработанная глобальная логическая модель показана на рисунке 1.

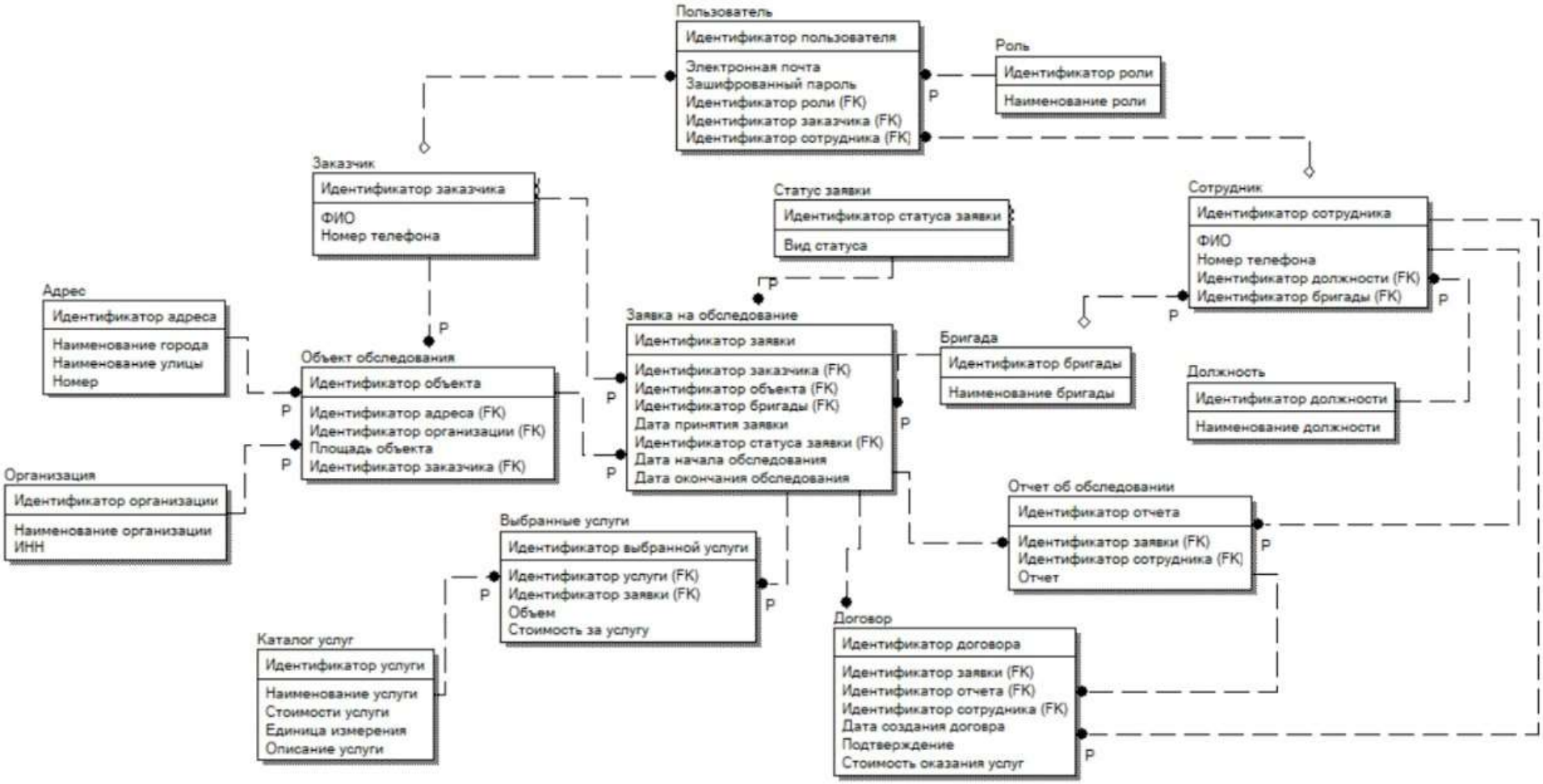


Рисунок 1 – Глобальная логическая модель

3.2 Разработка глобальной физической модели данных

Физическая модель данных описывает структуру и расположение данных на физическом уровне их хранения. Она определяет реализацию баз данных в конкретной системе управления базами данных (СУБД), включая таблицы, их колонки, индексы, ограничения целостности и способы доступа к данным. Физическая модель разрабатывается с учетом технических требований, таких как производительность, объем хранимой информации, целостность данных и их безопасность [3].

В результате разработки получена следующая физическая модель для СУБД PostgreSQL. Разработанная глобальная физическая модель показана на рисунке 2.

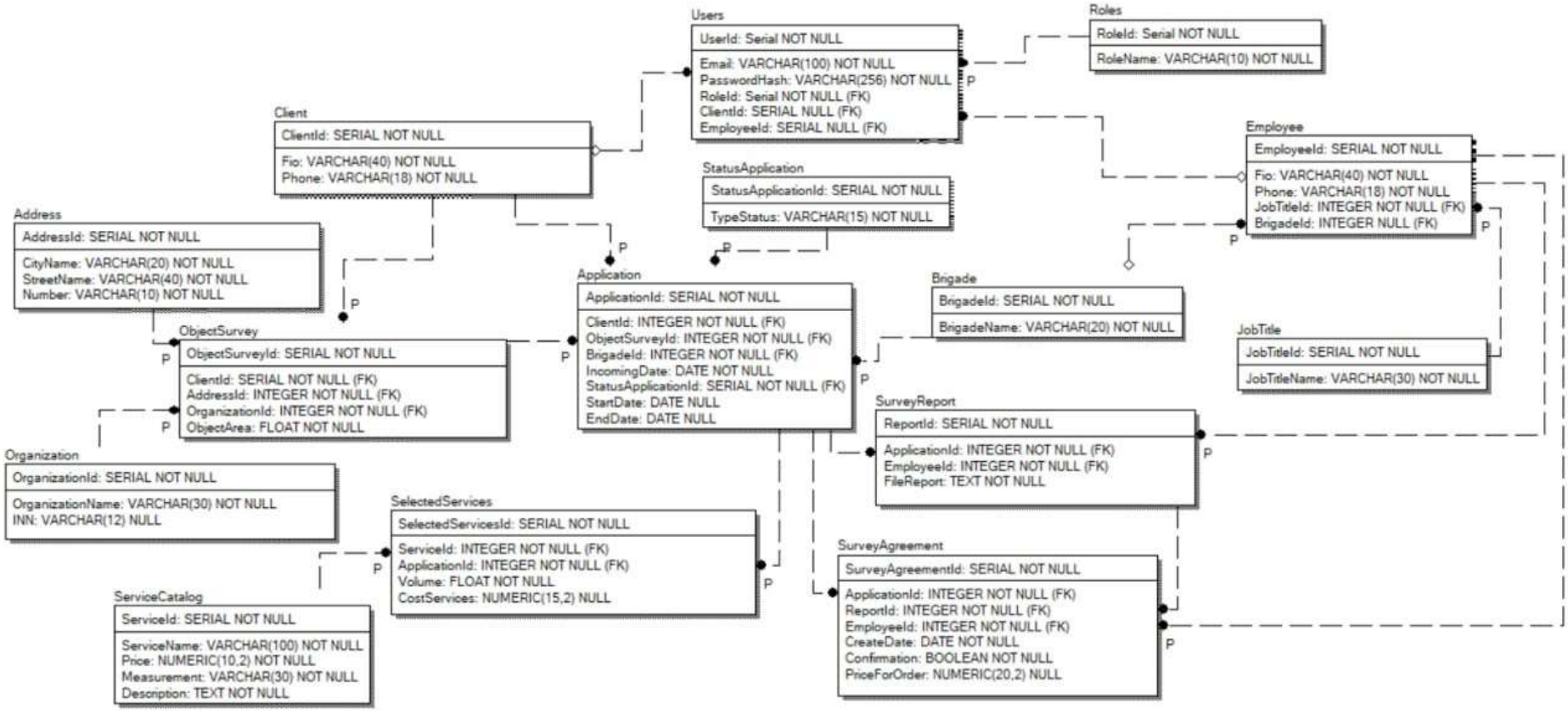


Рисунок 2 – глобальная физическая модель

4. Разработка базы данных

4.1 Проектирование таблиц [4]

Исходя из п.3.2 физической модели данных необходимо создать следующие таблицы:

1) заказчик:

- ClientId (SERIAL, PK) – уникальный идентификатор заказчика (первичный ключ);
- Fio (VARCHAR (40), NOT NULL) – фамилия, имя и отчество заказчика. Длина ограничена 40 символами;
- Phone (VARCHAR (18), UNIQUE, NOT NULL) – контактный номер телефона заказчика. Значение должно быть уникальным;
- Email (VARCHAR (100), UNIQUE, NOT NULL) – контактная почта заказчика. Значение должно быть уникальным

2) сотрудник:

- EmployeeId (SERIAL, PK) – уникальный идентификатор сотрудника (первичный ключ);
- Fio (VARCHAR (40), NOT NULL) – фамилия, имя и отчество сотрудника. Длина ограничена 40 символами;
- Phone (VARCHAR (18), UNIQUE, NOT NULL) – контактный номер телефона сотрудника. Значение должно быть уникальным;
- Email (VARCHAR (100), UNIQUE, NOT NULL) – контактная почта сотрудника. Значение должно быть уникальным
- JobTitleId (INTEGER, NOT NULL, FK → JobTitle(JobTitleId), ON DELETE SET NULL) – идентификатор должности сотрудника (внешний ключ, ссылается на JobTitle);
- BrigadeId (INTEGER, FK → Brigade(BrigadeId), ON DELETE SET NULL) – идентификатор бригады в которой состоит сотрудник (внешний ключ, ссылается на Brigade);

3) должность:

- JobTitleId (SERIAL, PK) - уникальный идентификатор должности (первичный ключ);
- JobTitleName (VARCHAR (30), UNIQUE, NOT NULL) – наименование должности. Длинна ограничена 30 символами;

4) пользователь:

- UserId (SERIAL, PK) – уникальный идентификатор пользователя (первичный ключ);
- Email (VARCHAR (100), NOT NULL, UNIQUE) – адрес электронной почты пользователя. Длина ограничена 100 символами;
- PasswordHash (VARCHAR (256), NOT NULL) – зашифрованный пароль пользователя. Длина ограничена 256 символами;
- RoleId (INT, NOT NULL, FK → Roles(RoleId) ON DELETE RESTRICT) – идентификатор роли пользователя (внешний ключ, ссылается на Roles);
- ClientId (INT, FK → Client(ClientId) ON DELETE RESTRICT) – идентификатор заказчика(внешний ключ, ссылается на Client);
- EmployeeId (INT, FK → Employee(EmployeeId) ON DELETE RESTRICT) – идентификатор сотрудника (внешний ключ, ссылается на Employee);

5) роль:

- RoleId (SERIAL, PK) – уникальный идентификатор роли;
- RoleName (VARCHAR(10)) – название роли. Длина ограничена 10 символами;

6) адрес:

- AddressId (SERIAL, PK) - уникальный идентификатор адреса (первичный ключ);
- CityName (VARCHAR (20), NOT NULL) – название города. Длинна ограничена 20 символами;

- StreetName (VARCHAR (40), NOT NULL) – название улицы. Длинна ограничена 20 символами;
- Number (VARCHAR (10), NOT NULL) – номер строения. Длинна ограничена 10 символами;

7) организация:

- OrganizationId (SERIAL, PK) – уникальный идентификатор организации (первичный ключ);
- OrganizationName (VARCHAR (30), NOT NULL) – наименования организации. Длинна ограничена 30 символами;
- INN (VARCHAR (18), UNIQUE, NOT NULL) – идентификационный номер налогоплательщика. Длинна ограничена 18 символами;

8) объект обследования:

- ObjectSurveyId (SERIAL, PK) – уникальный идентификатор объекта обследования (первичный ключ);
- ClientId (INTRGER, NOT NULL, FK→ Client(ClientId), ON DELETE SET NULL)
- AddressId (INTEGER, NOT NULL, FK → Address(AddressId), ON DELETE SET NULL) – уникальный идентификатор адреса на котором находится обследуемый объект обследования (внешний ключ, ссылается на Address);
- OrganizationId (INTEGER, NOT NULL, FK → Organization (OrganizationId), ON DELETE SET NULL) – уникальный идентификатор организации которой принадлежит объект обследования (внешний ключ, ссылается на Organization);
- ObjectArea (FLOAT, NOT NULL) – площадь которую занимает объект;

9) статус заявки на обследование:

- StatusApplicationId (SERIAL, PK) – уникальный идентификатор статуса заявки на обследование (первичный ключ);

- TypeStatus (varchar(15), NOTNULL) – вид статуса заявки на обследование. Длинна ограничена 15 символами;

10) заявка на обследование:

- AplicationId (SERIAL, PK) – уникальный идентификатор заявки на обследование (первичный ключ);
- ClientId (INTEGER, NOT NULL, FK → Client(ClientId), ON DELETE SET NULL) – идентификатор клиента который подал заявку на обследование (внешний ключ, ссылается на Client);
- ObjectSurveyId (INTEGER, NOT NULL, FK → ObjectSurvey (ObjectSurveyId), ON DELETE SET NULL) – идентификатор объекта обследования который необходимо обследовать клиенту (внешний ключ, ссылается на ObjectSurvey);
- BrigadeId (INTEGER, FK → Brigade(BrigadeId), ON DELETE SET NULL) – идентификатор бригады которая будет выполнять обследование(внешний ключ, ссылается на Brigade);
- IncomingDate (Date), NOT NULL) – дата принятия заказа;
- StatusApplicationId (INTEGER, FK → StatusApplication (StatusApplicationId), ON DELETE SET NULL) – идентификатор статуса заявки показывающий в каком состоянии находится заявка;
- StrateDate (Date, CHECK (IncomingDate< StrateDate)) – дата начала обследования;
- EndDate (Date, CHECK (StrateDate < EndDate)) – дата окончания обследования;

11) бригада:

- BrigadeId (SERIAL, PK) – уникальный идентификатор бригады (первичный ключ);
- BrigadeName (VARCHAR (20), NOT NULL) – наименование бригады. Длинна ограничена 20 символами;

12) каталог услуг:

- ServiceId (SERIAL, PK) – уникальный идентификатор услуги (первичный ключ);
- ServiceName (VARCHAR (100), NOT NULL) – наименование услуги. Длинна ограничена 100 символами;
- Price (NUMERIC (10,2), NOT NULL) – цена на оказанию услуги за единицу измерения;
- Measurement (VARCHAR (30), NOT NULL) – мера в которой измеряется услуги. Длинна ограничена 30 символами;
- Description (TEXT, NOT NULL) – описание услуги;

13) выбранные услуги:

- SelectedServicesId (SERIAL, PK) – уникальный идентификатор выбранной услуги (первичный ключ);
- ServiceId (INTEGER, NOT NULL, FK → Service(ServiceId), ON DELETE SET NULL, ON UPDATE CASCADE) – идентификатор услуги которая требуется клиенту(внешний ключ, ссылается на Service);
- AplicationId (INTEGER, NOT NULL, FK → Aplication (AplicationId), ON DELETE CASCADE) – идентификатор заявки на обследование к которой относится выбранная услуга (внешний ключ, ссылается на Aplication);
- Volume (FLOAT, NOT NULL) – объем выполняемой работы;
- CostServices (NUMERIC (15,2)) – цена оказания услуги за указанный объем работы;

14) отчет об обследовании:

- ReportId (SERIAL, PK) – уникальный идентификатор отчета об обследовании (первичный ключ);
- AplicationId (INTEGER, NOT NULL, FK → Aplication (AplicationId), ON DELETE RESTRICT) – идентификатор заявки на обследование к которой относится отчет об обследовании (внешний ключ, ссылается на Aplication);

- EmployeeId (INTEGER, NOT NULL, FK → Employee (EmployeeId), ON DELETE SET NULL) – идентификатор сотрудника который сделал отчет об обследовании (внешний ключ, ссылается на Employee);
- FileReport (TEXT, NOT NULL) – путь по которому лежит отчет;

15) договор:

- SurveyAgreementId (SERIAL, PK) - уникальный идентификатор договора (первичный ключ);
- AplicationId (INTEGER, NOT NULL, FK → Aplication (AplicationId), ON DELETE RESTRICT) – идентификатор заявки на обследование к которой составляется договор (внешний ключ, ссылается на Aplication);
- ReportId (INTEGER, NOT NULL, FK → SurveyReport (ReportId), ON DELETE SET NULL) – идентификатор отчета об обследовании (внешний ключ, ссылается на SurveyReport);
- EmployeeId (INTEGER, NOT NULL, FK → Employee (EmployeeId), ON DELETE SET NULL) – идентификатор сотрудника (внешний ключ, ссылается на Employee);
- CreateDate (DATE, NOT NULL) – дата создания;
- Confirmation (BOOLEAN, NOT NULL) – флаг подтверждения клиента;
- PriceForOrder (NUMERIC (20,2) – цена за всю проделанную работу.

Схема базы данных после создания таблиц (приложение А) отражена на рисунке 3.

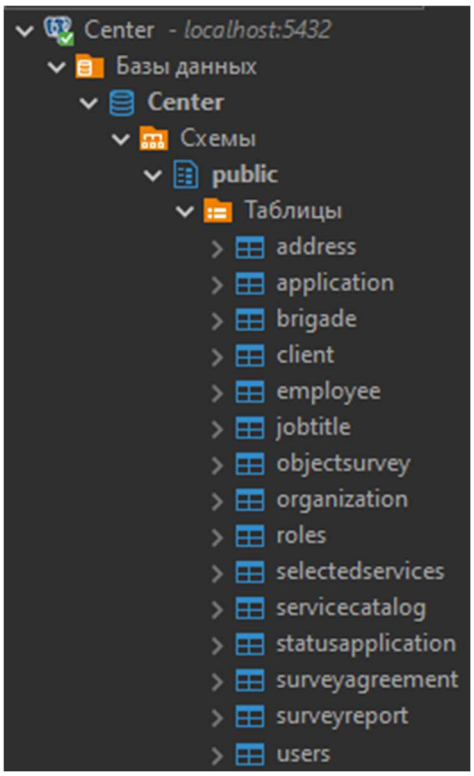


Рисунок 3 – Схема БД после создания таблиц

Созданные и заполненные данными таблицы представлены на рисунках 4-18.

	addressid	cityname	streetname	number
1	1	г.Рязань	ул. Интернациональнгая	д.1
2	2	г.Рязань	ул. Прижелезнодорожная	д.52
3	3	г.Рязань	Московское шоссе	д.5А
4	4	г.Рязань	ул. Бирюзова	д.28А
5	5	г.Рязань	ул. Интернациональнгая	д.23
6	6	г.Рязань	шоссе Солотчинское	д.11

Рисунок 4 – Таблица Address

	applicationid	clientid	objectsurveyid	brigadeid	incomingdate	statusapplicationid	startedate	enddate
1	1	1	1	1	2025-01-10	3	2025-01-11	2025-01-15
2	2	2	2	2	2025-01-15	3	2025-01-16	2025-01-18
3	3	3	3	1	2025-02-01	3	2025-02-02	2025-02-05
4	4	4	4	2	2025-02-10	3	2025-02-11	2025-02-13
5	5	4	5	1	2025-02-20	3	2025-02-21	2025-02-25
6	6	5	6	2	2025-03-01	2	2025-04-07	[NULL]
7	7	1	1	1	2025-03-10	2	2025-04-09	[NULL]
8	8	2	2	2	2025-03-20	1	[NULL]	[NULL]
9	9	1	1	2	2025-05-17	1	[NULL]	[NULL]

Рисунок 5 – Таблица Application

	brigadeid	brigadename
1	1	Бригада №1
2	2	Бригада №2

Рисунок 6 – Таблица Brigade

	clientid	fio	phone
1	1	Кузьмин Сергей Иванович	8(905)584-98-94
2	2	Евсеев Адам Романович	8(910)952-05-13
3	3	Пахомова Алина Михайловна	8(950)005-31-93
4	4	Мальцев Константин Артёмович	8(942)507-57-05
5	5	Романова Валентина Александровна	8(952)174-27-75

Рисунок 7 – Таблица Client

1	1	Ермолов Леон Маркович	8(910)293-00-97	1	[NULL]
2	2	Симонов Николай Егорович	8(965)867-22-69	2	1
3	3	Рябов Никита Макарович	8(971)301-14-78	2	2
4	4	Григорьев Илья Кириллович	8(944)991-09-27	3	1
5	5	Лобанов Виктор Евгеньевич	8(956)295-85-89	3	2
6	6	Уткина Лея Дмитриевна	8(985)874-96-44	4	[NULL]

Рисунок 8 – Таблица Employee

	jobtitleid	jobtitlename
1	1	Директор
2	2	Главный инженер проектировщик
3	3	Инженер проектировщик
4	4	Менеджер

Рисунок 9 – Таблица JobTitle

	objectsurveyid	clientid	addressid	organizationid	objectarea
1	1	1	1	1	15 000
2	2	2	2	2	160 000
3	3	3	3	3	36 500
4	4	4	4	4	1 500
5	5	4	5	4	1 000
6	6	5	5	5	27 000

Рисунок 10 – Таблица ObjectSurvey

	organizationid	organizationname	inn
1	1	Строительная компания	0420520521
2	2	Русская кожа	0602065848
3	3	тц. Барс	6227009810
4	4	Прайм	6234203335
5	5	Круиз	6234085000

Рисунок 11 – Таблица Organization

	roleid	rolename
1	1	Client
2	2	Employee
3	3	Admin

Рисунок 12 – Таблица Roles

	selectedservicesid	serviceid	applicationid	volume	costservices
1	1	1	1	1	10 000
2	2	2	1	15 000	15 000 000
3	3	1	2	1	10 000
4	4	3	2	5 000	1 000 000
5	5	1	3	1	10 000
6	6	4	3	15 000	15 000 000
7	7	1	4	1	10 000
8	8	5	4	1 500	262 500
9	9	1	5	1	10 000
10	10	2	5	1 000	1 000 000
11	11	1	6	1	10 000
12	12	3	6	27 000	5 400 000
13	13	1	7	1	10 000
14	14	4	7	15 000	15 000 000
15	15	1	8	1	10 000
16	16	5	8	20 000	350 000
17	17	1	9	1	10 000

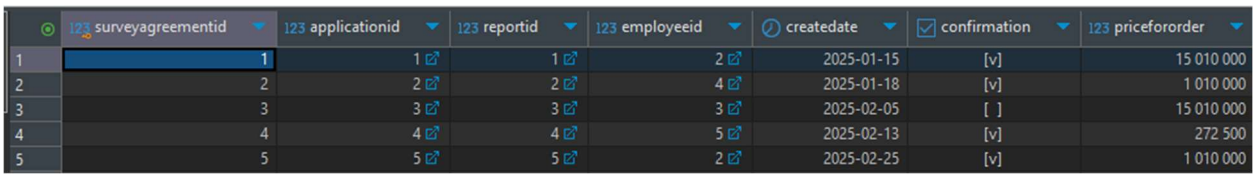
Рисунок 13 – Таблица SelectedServices

	serviceid	servicename	price	measurement	description
1	1	Составление отчета	10 000	ШТ	Наши специалисты составят отчет, в котором будут указаны про
2	2	Обследование технического состояния несущих и ограждающих конст	350	М2	Проводится детальное обследование несущих и ограждающих к
3	3	Поиск и обследование технического состояния инженерных коммуни	400	М2	Выполняется поиск и диагностика состояния внутренних комму
4	4	Обследование технического состояния инженерных систем	450	М2	Оценивается общее техническое состояние инженерных систем
5	5	Обмерные работы	200	М2	Проводятся точные обмеры помещения или здания с составлен

Рисунок 14 – Таблица ServiceCatalog

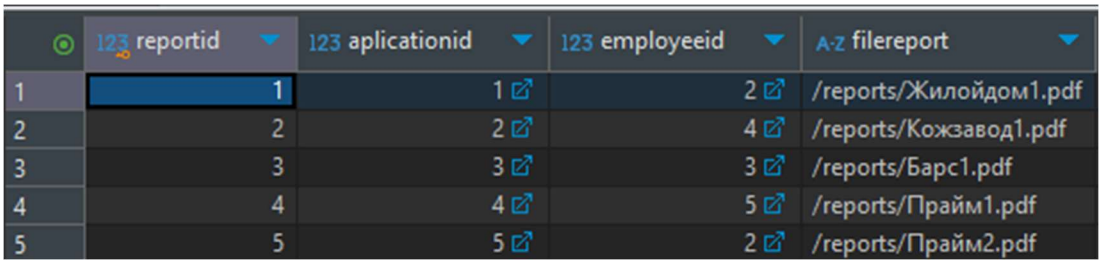
	statusapplicationid	typestatus
1	1	Принята
2	2	Выполняется
3	3	Закончена

Рисунок 15 – Таблица StatusApplication



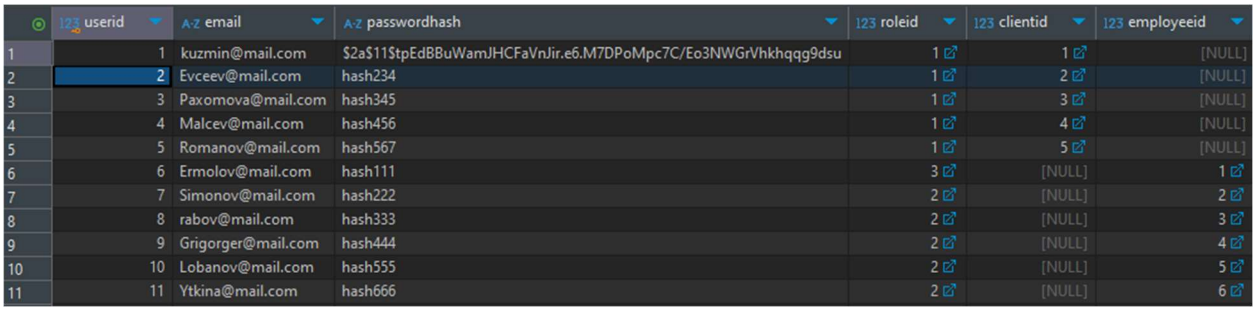
	surveyagreementid	applicationid	reportid	employeeid	createdate	confirmation	pricefororder
1	1	1	1	2	2025-01-15	[v]	15 010 000
2	2	2	2	4	2025-01-18	[v]	1 010 000
3	3	3	3	3	2025-02-05	[ ]	15 010 000
4	4	4	4	5	2025-02-13	[v]	272 500
5	5	5	5	2	2025-02-25	[v]	1 010 000

Рисунок 16 – Таблица SurveyAgreement



	reportid	aplicationid	employeeid	filereport
1	1	1	2	/reports/Жилойдом1.pdf
2	2	2	4	/reports/Кожзавод1.pdf
3	3	3	3	/reports/Барс1.pdf
4	4	4	5	/reports/Прайм1.pdf
5	5	5	2	/reports/Прайм2.pdf

Рисунок 17 – Таблица SurveyReport



	userid	email	passwordhash	roleid	clientid	employeeid
1	1	kuzmin@mail.com	$2a$11$tpEdBBuWamJHCFaVnJir.e6.M7DPoMpc7C/Eo3NWGrVhkhqqg9dsu	1	1	[NULL]
2	2	Evceev@mail.com	hash234	1	2	[NULL]
3	3	Paxomova@mail.com	hash345	1	3	[NULL]
4	4	Malcev@mail.com	hash456	1	4	[NULL]
5	5	Romanov@mail.com	hash567	1	5	[NULL]
6	6	Ermolov@mail.com	hash111	3	[NULL]	1
7	7	Simonov@mail.com	hash222	2	[NULL]	2
8	8	rabov@mail.com	hash333	2	[NULL]	3
9	9	Grigorger@mail.com	hash444	2	[NULL]	4
10	10	Lobanov@mail.com	hash555	2	[NULL]	5
11	11	Ytkina@mail.com	hash666	2	[NULL]	6

Рисунок 18 – Таблица Users

4.2 Проектирование функций

4.2.1 Проектирование хранимых функций

Функции формируют набор строк с указанными столбцами из одной или нескольких таблиц. Они дают возможность комбинировать данные из разных запросов и выводить их в виде единого результата [5].

Разработаем следующие хранимые функции (Приложение А):

1) функция позволяет анализировать все заявки клиента. Она возвращает таблицу с данными об адресе объекта обследования, организации, наименование исполнительной бригады, дате подачи заявки, статусе заявки, дате начала и окончания обследования и дате формирования договора. Результаты сортируются по дате подачи заявки что позволяет отслеживать клиенту поданные и завершенные заявки. Эта функция полезна для анализа времени исполнения заявки на различных объектах обследования клиента;

2) функция анализирует принесенную прибыль от каждой организации за определенный временной период. Она возвращает таблицу с данными об наименовании организации и её ИНН, и сумму стоимости оказания услуг всех договоров, заключенных с данной организации. Эта функция полезна для оценки прибыли, принесенных от организаций;

3) функция анализирует принесенную прибыль от каждой исполнительной бригады за определенный временной период. Она возвращает таблицу с данными об наименовании исполнительной бригады и сумму стоимости оказания услуг всех договоров исполнителями которых является данная бригада. Эта функция полезна для оценки эффективности работы сотрудников, состоящих в исполнительных бригадах и принятия управленческих решений на основе их производительности;

4.2.2 Проектирование хранимых процедур

Процедуры применяются для выполнения сложных алгоритмов обработки данных. В отличие от селективных, они могут либо не возвращать строки вообще, либо выдавать результат в виде одной строки через выходные параметры [5].

Разработаем следующие хранимые процедуры (Приложение А):

1) процедура выполняет несколько шагов для добавления нового заказчика:

- добавляется новый заказчик в таблицу Clients. Вставляются данные об клиенте. Возвращается идентификатор клиента в переменную reg_clientid;
- добавляется новый пользователь в таблицу Users. Вставляются данные об пользователе;
- если на каком-либо этапе возникает ошибка, все изменения откатываются, и выдается исключение с сообщением об ошибки.

Данная функция позволяет добавить нового клиента;

2) процедура выполняет несколько шагов для добавления новой заявки:

- клиент вводит свой номер телефона наименование и ИНН организации а также адрес объекта обследования и его площадь, если номер телефона не найден выдается ошибка, если организация не найдена тогда она добавляется в таблицу Organization и возвращается её уникальный идентификатор, который сохраняется в переменной new_organization, если адрес не найден тогда он добавляется в таблицу Address и возвращается её уникальный идентификатор, который сохраняется в переменной new_address, после этих проверок проверяется имеется ли объект обследования в таблице ObjectSurvey, если нет то он добавляется используя идентификаторы клиента, добавленной организации и добавленного адреса, а также площади объекта обследования;
- добавляется новая заявка в таблице Application вставляется уникальный идентификатор клиента и уникальный идентификатор объекта обследования;
- если на каком-либо этапе возникает ошибка, все изменения откатываются, и выдается исключение с сообщением об ошибки.

Данная функция позволяет добавить новую заявку, а также новую организацию, адрес, объект обследования и выбранные услуги;

3) процедура выполняет несколько шагов для добавления новой выбранной услуги:

- заказчик вводит свой номер телефона, дату подачи заявки, ИНН организации, адрес объекта обследования, наименование выбранной услуги и объем требуемой работы;
- если на каком-либо этапе возникает ошибка, все изменения откатываются, и выдается исключение с сообщением об ошибки.

Данная функция позволяет добавить выбранную услугу.

4.3 Проектирование триггеров [5]

Разработаны следующие триггеры (Приложение А).

4.3.1 Проектирование триггеров бизнес-требований

1) при добавлении или обновлении выбранной услуги необходимо автоматически рассчитывать стоимость её оказания, исходя из установленной стоимости за единицу объема и указанного объёма;

2) при создании договора подсчитывать сумму оказанных услуг;

3) при создании новой заявки автоматически выбирать исполнительную бригаду.

4.3.2 Проектирование триггеров бизнес-ограничений

1) отчет не может быть добавлен если заявки не является законченной;

2) договор не может быть добавлен если заявки не является законченной и отчет об обследовании не был добавлен;

3) нельзя установить дату формирования договора на более ранний срок чем дату проведения обследования.

4.3.3 Проектирование триггера ограничений целостности

Нельзя удалить отчет если связанный договор подтвердил клиент.

4.3.4 Проектирование триггера ограничения действия

1) нельзя добавить отчет об обследовании если сотрудник не состоит в бригаде, которая исполняла заявку;

2) нельзя удалить договор если его подтвердил клиент.

4.3.5 Проектирование триггера DDL

DDL-триггер будет создан как дополнительный уровень защиты базы данных, предотвращающие удаление таблиц всеми пользователями, кроме postgres.

4.3.6 Проектирование триггера БД

Триггер базы данных будет создан как дополнительный уровень защиты базы данных, записывающий в таблицу userlog имя пользователя и время его подключения к БД.

4.4 Разработка реализации политики безопасности

Администратор – сотрудник, который контролирует работу системы. Роль должна обладать правами:

- полные права (CREATE, SELECT, INSERT, UPDATE, DELETE) на все таблицы;
- доступ ко всем функциям и триггерам;
- возможность создавать и назначать права другим пользователям.

Сотрудник – человек, который осуществляет управление заявками, договорами и отчетами, а также должен иметь возможность просматривать клиентов, выбранные услуги, услуги, организации, адреса, объекты обследования и бригады. Роль должна обладать правами:

- разрешение на работу с заявками, отчетами, договорами (SELECT, INSERT, UPDATE в Application, SurveyReport, SurveyAgreement);
- просмотр клиентов, выбранных услуг, услуг, организаций, адресы, объекты обследования и бригады (SELECT в Client, SelectedServices, ServiceCatalog, Organization, Address, ObjectSurvey, Brigade);
- использование хранимой функции для анализа принесенной прибыли от каждой исполнительной бригады;
- использование хранимой функции для анализа принесенной прибыли от каждой организации.

Заказчик – человек, осуществляющий формирование заявки, добавление организаций, адресов, объектов обследования, а также иметь возможность просматривать услуги, выбранные услуги, которые относятся к его заявкам, заявки оставленные им, отчеты относящиеся к его заявкам и договора, относящиеся к его заявкам. Роль должна обладать правами:

- разрешение на работу с адресами и организациями (SELECT, INSERT, UPDATE в Organization, Address);
- разрешение на добавление новых заявок и выбранных услуг (INSERT в Application, SelectedServices);

- разрешение на просмотр каталога услуг, заявок, договоров и отчетов об исследовании (SELECT в ServiceCatalog, Application, SurveyAgreement, SurveyReport);
- использования хранимой функции для анализа заявок заказчика;
- использование хранимой процедуры для добавления нового клиента;
- использование хранимой процедуры для создания новой заявки;
- использование хранимой процедуры для добавления выбранной услуги.

5. Разработка серверной части

Для реализации серверной части информационной системы был создан проект в среде Visual Studio. Для обеспечения функций позволяющих работать с базой данных используется Entity Framework[6], представляющий собой специальную объектно-ориентированную технологию на базе фреймворка .NET для работы с данными.

При создании проекта были установлены необходимые пакеты. Данные пакеты представлены на рисунке 19.

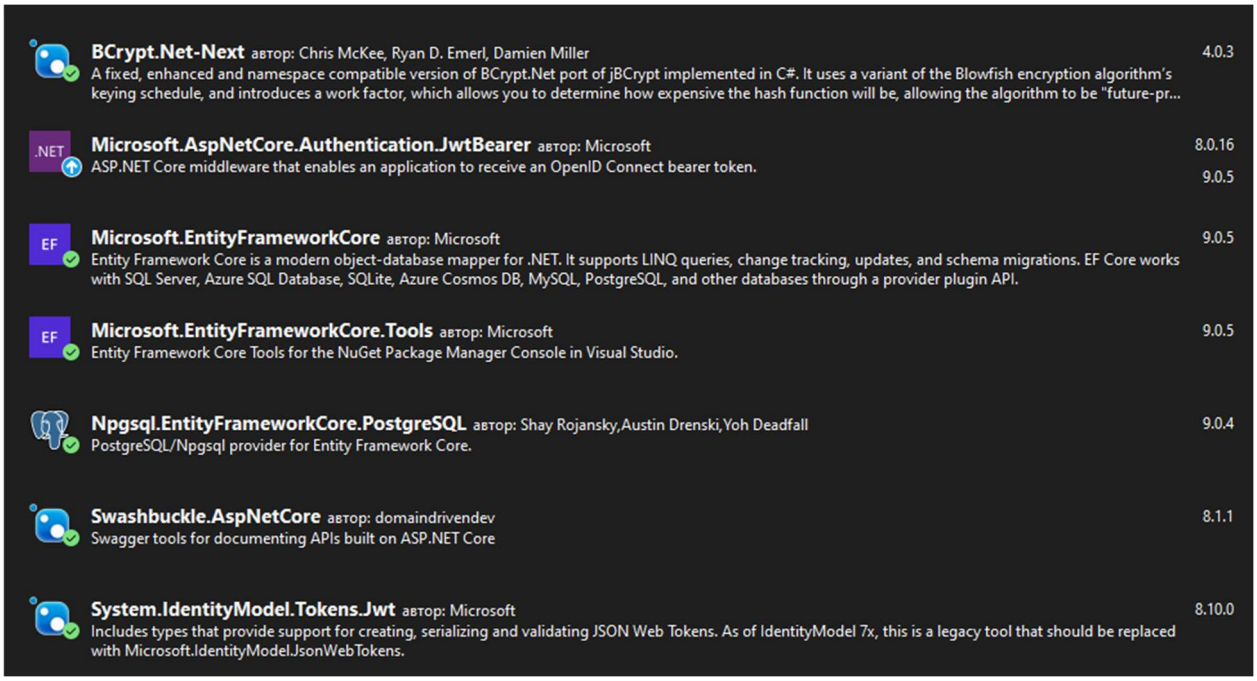


Рисунок 19 – Установленные пакет

5.1 Настройка подключения к базе данных

Для подключения разработанной базы данных к проекту был разработан appsetings.json, который хранит в себе строку подключения к базе данных (листинг 1).

Листинг 1 – Подключение к базе данных

```
{
  "ConnectionStrings": {
    "DefaultConnection":
"Host=localhost;Port=5432;Database=Center;Username=postgres;Password=2111"
}
```

Для определения структуры базы данных был разработан класс CenterContext. Созданный контекст данных отображен в приложении Б.

5.2 Создание сущностных классов

На основе разработанной физической модели данных и созданной базы данных необходимо реализовать класс каждой сущности, в которых будут хранится описание атрибутов сущностей а таже связи с другими сущностями. Всего разрабатываемых моделей 15.

Класс модели (сущности) – это класс, который используется для сопоставления с таблицей базы данных в Entity Framework Core. Сопоставление происходит по наименованию таблицы и модели, EF Core получает информацию, какую таблицу использовать для столбцов и сопоставлений конфигурации [6].

Для созданных моделей была создана папка Models в которой хранятся все модели сущностей с соответствующими именами и атрибутами, полученными в результате разработки физической модели и базы данных. Созданные классы, хранящиеся в папке Models отображены на рисунке 20.

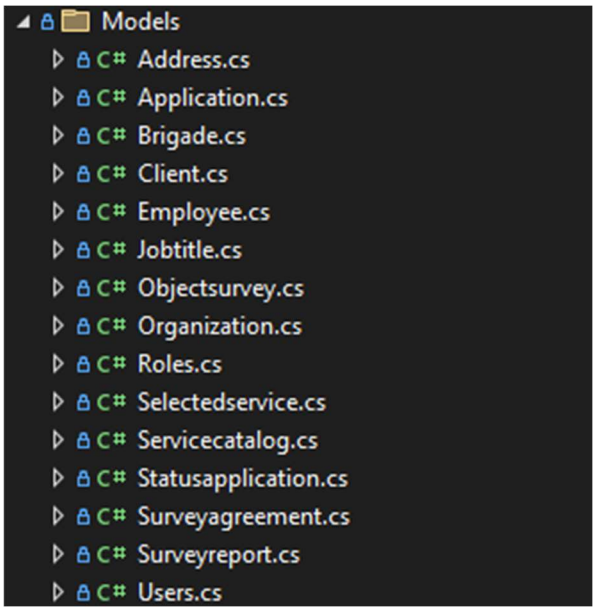


Рисунок 20 – Созданные классы

Разработка сущностного класса Client представлена в листинге 2.

Листинг 2 – Класс Client

```
using System;
using System.Collections.Generic;

namespace WebApplication1.Models;

public partial class Client
{
    public int Clientid { get; set; }
    public string Fio { get; set; } = null!;
    public string Phone { get; set; } = null!;
    public virtual ICollection<Objectsurvey> Objectsurveys { get; set; } = new
List<Objectsurvey>();
    public virtual ICollection<Application> Applications { get; set; } = new
List<Application>();
    public virtual ICollection<Users> Userses { get; set; } = new List<Users>();
}
```

Подобным образом были разработаны остальные сущностные классы. Созданные сущностные классы отображены в приложении Б.

Для ограничения и структурирования передаваемых данных между серверной и клиентской частью приложения в проекте были дополнительно реализованы классы DTO.

Данные классы создаются отдельно от моделей и имею только те атрибуты, с которыми необходимо взаимодействовать через API. Это позволяет снизить объемы передаваемых данных и обеспечивает независимость клиентской части от структуры базы данных.

Для классов DTO создана отдельная папка DTOS в которой хранятся все классы DTO. Созданные классы, хранящиеся в папке DTOS отображены на рисунке 21.

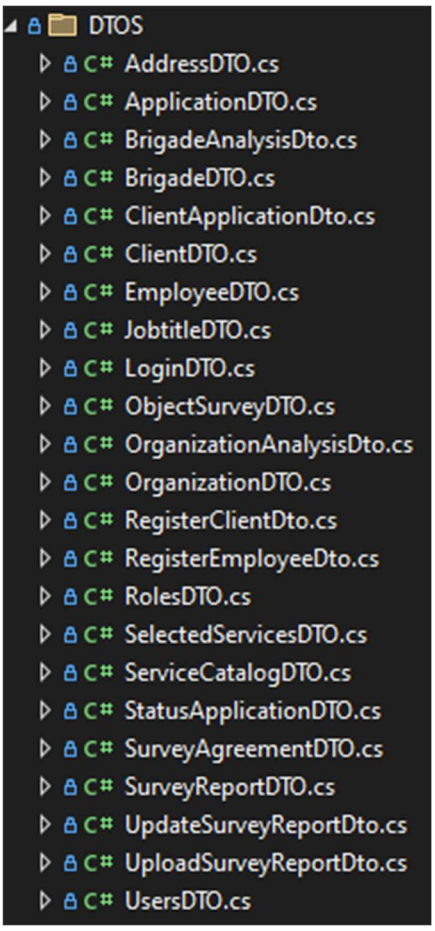


Рисунок 21 – Созданные DTO-классы

Разработка DTO-класса для сущности Client представлена в листинге 3.

Листинг 3 – Класс ClientDto

```
namespace WebApplication1.DTOS
{
    public class ClientCreateDto
    {
        public string Fio { get; set; }
        public string Phone { get; set; }
    }

    public class ClientDto : ClientCreateDto
    {
        public int ClientId { get; set; }
    }

}
```

Подобным образом были разработаны остальные DTO-классы. Созданные DTO-классы отображены в приложении Б.

Для реализации добавления и обновления были сделаны отдельные DTO-класс (листинг 4 - 5)

Листинг 4 – UploadSurveyReportDto

```
using Microsoft.AspNetCore.Mvc;
```

```
namespace WebApplication1.DTOS
{
    public class UploadSurveyReportDto
    {
        [FromForm]
        public int ApplicationId { get; set; }
        [FromForm]
        public int EmployeeId { get; set; }
        [FromForm]
        public IFormFile File { get; set; }
    }
}
```

Листинг 5 – UpdateSurveyReportDto

```
using Microsoft.AspNetCore.Mvc;

namespace WebApplication1.DTOS
{
    public class UpdateSurveyReportDto
    {
        [FromForm]
        public int ReportId { get; set; }
        [FromForm]
        public int ApplicationId { get; set; }
        [FromForm]
        public int EmployeeId { get; set; }
        [FromForm]
        public IFormFile? File { get; set; }
    }

}
```

Также реализации регистрации и авторизации были использованы JWT [7]. Для этого были созданы DTO-классы регистрации заказчиков и сотрудников, а также DTO-класс авторизации (листинг 6 - 8).

Листинг 6 – Класс RegisterClientDto

```
namespace WebApplication1.DTOS
{
    public class RegisterClientDto
    {
        public string Email { get; set; }
        public string Password { get; set; }
        public string Fio { get; set; }
        public string Phone { get; set; }
    }

}
```

Листинг 7 – Класс RegisterEmployreeDto

```
namespace WebApplication1.DTOS
{
    public class RegisterEmployeeDto
    {
        public string Email { get; set; }
        public string Password { get; set; }
        public string Fio { get; set; }
        public string Phone { get; set; }
```

```
        public int JobTitleId { get; set; }
        public int? BrigadeId { get; set; }
    }

}
```

Листинг 8 – Класс LoginDto

```
namespace WebApplication1.DTOS
{
    public class LoginDto
    {
        public string Email { get; set; }
        public string Password { get; set; }
    }

}
```

5.3 Реализация серверной части информационной системы на языке C#

На основе спроектированных классов необходимо с помощью средств Entity Framework разработать контроллеры для каждого класса, выполняющие CRUD[16] операции по отношению к данным в базе данных. Контроллеры добавляются в папку Controllers. Созданные контроллеры в папке Controllers отображены на рисунке 22.

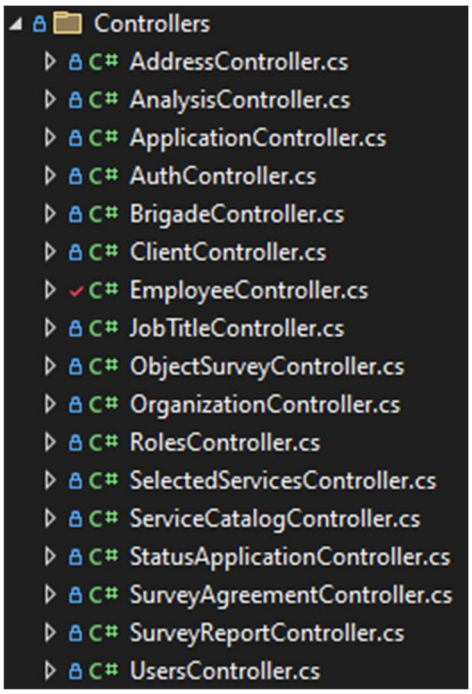


Рисунок 22 - Созданные контроллеры

Реализованный контроллер для класса Client (листинг 9).

Листинг 9 – ClientController

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System;
```

```
using WebApplication1;
using WebApplication1.DTOS;
using WebApplication1.Models;
[ApiController]
[Route("api/[controller]")]
public class ClientController : ControllerBase
{
    private readonly CenterContext _context;
    public ClientController(CenterContext context)
    {
        _context = context;
    }
    // GET: api/Client
    [HttpGet]
    [Authorize(Roles = "Admin,Employee")]
    public async Task<ActionResult<IEnumerable<ClientDto>>> GetClients()
    {
        var clients = await _context.Clients
            .Select(c => new ClientDto
            {
                ClientId = c.Clientid,
                Fio = c.Fio,
                Phone = c.Phone
            }).ToListAsync();

        return Ok(clients);
    }
    // GET: api/Client/5
    [HttpGet("{id}")]
    [Authorize(Roles = "Admin,Employee")]
    public async Task<ActionResult<ClientDto>> GetClient(int id)
    {
        var client = await _context.Clients.FindAsync(id);
        if (client == null) return NotFound();
        return Ok(new ClientDto
        {
            ClientId = client.Clientid,
            Fio = client.Fio,
            Phone = client.Phone
        });
    }
    // POST: api/Client
    [HttpPost]
    [Authorize(Roles = "Admin")]
    public async Task<ActionResult<ClientDto>> CreateClient(ClientCreateDto dto)
    {
        var client = new Client
        {
            Fio = dto.Fio,
            Phone = dto.Phone,
        };
        _context.Clients.Add(client);
        await _context.SaveChangesAsync();
        return CreatedAtAction(nameof(GetClient), new { id = client.Clientid }, new
ClientDto
        {
            ClientId = client.Clientid,
            Fio = client.Fio,
            Phone = client.Phone
        });
    }
    // PUT: api/Client/5
```

```
    [HttpPut("{id}")]
    [Authorize(Roles = "Admin")]
    public async Task<IActionResult> UpdateClient(int id, ClientCreateDto dto)
    {
        var client = await _context.Clients.FindAsync(id);
        if (client == null) return NotFound();
        client.Fio = dto.Fio;
        client.Phone = dto.Phone;
        await _context.SaveChangesAsync();
        return NoContent();
    }
    // DELETE: api/Client/5
    [HttpDelete("{id}")]
    [Authorize(Roles = "Admin")]
    public async Task<IActionResult> DeleteClient(int id)
    {
        var client = await _context.Clients.FindAsync(id);
        if (client == null) return NotFound();
        _context.Clients.Remove(client);
        await _context.SaveChangesAsync();
        return NoContent();
    }

}
```

Подобным образом для каждой сущности были реализованы контроллеры. Созданные контроллеры отображены в приложении Б.

В контроллере для сущности SurveyReport были разработаны методы реализующие загрузку и выгрузку отчета об обследовании. Данный контроллер представлен на листинге 10

Листинг 10 – SurveyReportController

```
using Microsoft.AspNetCore.Mvc;
using WebApplication1.DTOS;
using WebApplication1.Models;
using WebApplication1;
using Microsoft.EntityFrameworkCore;
using Microsoft.AspNetCore.Authorization;

[ApiController]
[Route("api/[controller]")]
public class SurveyReportController : ControllerBase
{
    private readonly CenterContext _context;
    private readonly IWebHostEnvironment _env;
    public SurveyReportController(CenterContext context, IWebHostEnvironment env)
    {
        _context = context;
        _env = env;
    }
    // GET: api/SurveyReport
    [HttpGet]
    [Authorize(Roles = "Admin,Employee")]
    public async Task<ActionResult<IEnumerable<SurveyReportDto>>> GetSurveyReports()
    {
        var reports = await _context.Surveyreports
            .Select(r => new SurveyReportDto
```

```
            {
                ReportId = r.Reportid,
                AplicationId = r.Aplicationid,
                EmployeeId = r.Employeeid,
                FileReport = r.Filereport
            }).ToListAsync();

        return Ok(reports);
    }
    // GET: api/SurveyReport/5
    [HttpGet("{id}")]
    [Authorize(Roles = "Admin,Employee")]
    public async Task<ActionResult<SurveyReportDto>> GetSurveyReport(int id)
    {
        var report = await _context.Surveyreports.FindAsync(id);
        if (report == null) return NotFound();
        return Ok(new SurveyReportDto
        {
            ReportId = report.Reportid,
            AplicationId = report.Aplicationid,
            EmployeeId = report.Employeeid,
            FileReport = report.Filereport
        });
    }
    // POST: api/SurveyReport
    [HttpPost("upload")]
    [Consumes("multipart/form-data")]
    [Authorize(Roles = "Admin,Employee")]
    public async Task<IActionResult> Upload([FromForm] UploadSurveyReportDto dto)
    {
        if (dto.File == null || dto.File.Length == 0)
            return BadRequest(new { error = "Файл не загружен" });
        var fileName = Guid.NewGuid().ToString() +
Path.GetExtension(dto.File.FileName);
        var report = new Surveyreport
        {
            Aplicationid = dto.ApplicationId,
            Employeeid = dto.EmployeeId,
            Filereport = fileName
        };
        _context.Surveyreports.Add(report);
        await _context.SaveChangesAsync(); // Сначала проверяем, что всё ок в БД
        // Только если всё хорошо — сохраняем файл
        var uploadsFolder = Path.Combine(_env.ContentRootPath, "UploadedReports");
        if (!Directory.Exists(uploadsFolder))
            Directory.CreateDirectory(uploadsFolder);
        var filePath = Path.Combine(uploadsFolder, fileName);
        using (var stream = new FileStream(filePath, FileMode.Create))
        {
            await dto.File.CopyToAsync(stream);
        }
        return Ok(new { report.Reportid });
    }
    // PUT: api/SurveyReport
    [HttpPut("update")]
    [Consumes("multipart/form-data")]
    [Authorize(Roles = "Admin,Employee")]
    public async Task<IActionResult> Update([FromForm] UpdateSurveyReportDto dto)
    {
        var report = await _context.Surveyreports.FindAsync(dto.ReportId);
        if (report == null) return NotFound("Отчёт не найден");
        report.Aplicationid = dto.ApplicationId;
```

```
        report.Employeeid = dto.EmployeeId;
        if (dto.File != null && dto.File.Length > 0)
        {
            // Удаляем старый файл
            var uploadsFolder = Path.Combine(_env.ContentRootPath,
"UploadedReports");
            var oldFilePath = Path.Combine(uploadsFolder, report.Filereport);
            if (System.IO.File.Exists(oldFilePath))
            {
                System.IO.File.Delete(oldFilePath);
            }
            // Загружаем новый файл
            var newFileName = Guid.NewGuid().ToString() +
Path.GetExtension(dto.File.FileName);
            var newFilePath = Path.Combine(uploadsFolder, newFileName);
            using (var stream = new FileStream(newFilePath, FileMode.Create))
            {
                await dto.File.CopyToAsync(stream);
            }
            report.Filereport = newFileName;
        }
        await _context.SaveChangesAsync();
        return Ok("Отчёт успешно обновлён");
    }
    // GET: api/SurveyReport/Download/5
    [HttpGet("download/{id}")]
    [Authorize(Roles = "Admin,Employee")]
    public async Task<IActionResult> DownloadSurveyReport(int id)
    {
        var report = await _context.Surveyreports.FindAsync(id);
        if (report == null) return NotFound();
        var uploadsFolder = Path.Combine(_env.ContentRootPath, "UploadedReports");
        var filePath = Path.Combine(uploadsFolder, report.Filereport);
        if (!System.IO.File.Exists(filePath))
            return NotFound("Файл не найден");
        var memory = new MemoryStream();
        using (var stream = new FileStream(filePath, FileMode.Open))
        {
            await stream.CopyToAsync(memory);
        }
        memory.Position = 0;
        return File(memory, "application/pdf", Path.GetFileName(filePath));
    }
    // DELETE: api/SurveyReport
    [HttpDelete("{id}")]
    [Authorize(Roles = "Admin")]
    public async Task<IActionResult> DeleteSurveyReport(int id)
    {
        var report = await _context.Surveyreports.FindAsync(id);
        if (report == null) return NotFound();
        var uploadsFolder = Path.Combine(_env.ContentRootPath, "UploadedReports");
        var filePath = Path.Combine(uploadsFolder, report.Filereport);
        if (System.IO.File.Exists(filePath))
            System.IO.File.Delete(filePath);
        _context.Surveyreports.Remove(report);
        await _context.SaveChangesAsync();
        return NoContent();
    }
}
```

Для реализации авторизации и регистрации с помощью JWT был разработан специальный сервис которой при авторизации генерирует JWT. Данный сервис представлен на листинге 11.

Листинг 11 – JwtService

```
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;
using Microsoft.Extensions.Configuration;
using Microsoft.IdentityModel.Tokens;
using WebApplication1.Models;
public class JwtService
{
    private readonly IConfiguration _configuration;

    public JwtService(IConfiguration configuration)
    {
        _configuration = configuration;
    }
    public string GenerateToken(Users user)
    {
        var claims = new List<Claim>
    {
        new Claim(ClaimTypes.NameIdentifier, user.Userid.ToString()),
        new Claim(ClaimTypes.Name, user.Email),
        new Claim(ClaimTypes.Role, user.Roles.Rolename)
    };
        if (user.Clientid.HasValue)
            claims.Add(new Claim("ClientId", user.Clientid.Value.ToString()));
        if (user.Employeeid.HasValue)
            claims.Add(new Claim("EmployeeId", user.Employeeid.Value.ToString()));
        var key = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_configuration["Jwt:Key"]));
        var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);
        var expires =
DateTime.Now.AddMinutes(Convert.ToDouble(_configuration["Jwt:ExpiresMinutes"]));
        var token = new JwtSecurityToken(
            issuer: _configuration["Jwt:Issuer"],
            audience: _configuration["Jwt:Audience"],
            claims: claims,
            expires: expires,
            signingCredentials: creds
        );
        return new JwtSecurityTokenHandler().WriteToken(token);
    }

}
```

На основе спроектированных DTO-классов регистрации и авторизации был разработан контроллер, который осуществляет регистрацию пользователей, хеширование паролей, проверку наличия учетных записей, а также авторизацию пользователей в системе (листинг 12).

Листинг 12 – AuthController

```
using Microsoft.AspNetCore.Mvc;
```

```
using Microsoft.EntityFrameworkCore;
using Microsoft.AspNetCore.Identity;
using Microsoft.IdentityModel.Tokens;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;
using WebApplication1.DTOS;
using WebApplication1.Models;
using BCrypt.Net;
using Microsoft.AspNetCore.Authorization;
namespace WebApplication1.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class AuthController : ControllerBase
    {
        private readonly CenterContext _context;
        private readonly IConfiguration _configuration;
        private readonly PasswordHasher<Users> _passwordHasher;
        public AuthController(CenterContext context, IConfiguration configuration)
        {
            _context = context;
            _configuration = configuration;
            _passwordHasher = new PasswordHasher<Users>();
        }
        private async Task<int> GetRoleId(string roleName)
        {
            var role = await _context.Roleses.FirstOrDefaultAsync(r => r.Rolename ==
roleName);
            if (role == null)
                throw new Exception($"Роль '{roleName}' не найдена");
            return role.Roleid;
        }
        [HttpPost("register/client")]
    public async Task<IActionResult> RegisterClient(RegisterClientDto dto)
    {
        if (await _context.Userses.AnyAsync(u => u.Email == dto.Email))
            return BadRequest("Пользователь с таким Email уже существует");
        var client = new Client
        {
            Fio = dto.Fio,
            Phone = dto.Phone
        };
        _context.Clients.Add(client);
        await _context.SaveChangesAsync();
        var user = new Users
        {
            Email = dto.Email,
            Passwordhash = BCrypt.Net.BCrypt.HashPassword(dto.Password),
            Clientid = client.Clientid,
            Roleid = await GetRoleId("Client")
        };
        _context.Userses.Add(user);
        await _context.SaveChangesAsync();
        return Ok("Клиент успешно зарегистрирован");
    }
    [HttpPost("register/employee")]
        [Authorize(Roles = "Admin")]
        public async Task<IActionResult> RegisterEmployee(RegisterEmployeeDto dto)
    {
        if (await _context.Userses.AnyAsync(u => u.Email == dto.Email))
            return BadRequest("Пользователь с таким Email уже существует");
```

```
        var employee = new Employee
        {
            Fio = dto.Fio,
            Phone = dto.Phone,
            Jobtitleid = dto.JobTitleId,
            Brigadeid = dto.BrigadeId
        };
        _context.Employees.Add(employee);
        await _context.SaveChangesAsync();
        var user = new Users
        {
            Email = dto.Email,
            Passwordhash = BCrypt.Net.BCrypt.HashPassword(dto.Password),
            Employeeid = employee.Employeeid,
            Roleid = await GetRoleId("Employee")
        };
        _context.Userses.Add(user);
        await _context.SaveChangesAsync();
        return Ok("Сотрудник успешно зарегистрирован");
    }
        [HttpPost("login")]
        public async Task<IActionResult> Login([FromBody] LoginDto dto)
        {
            var user = await _context.Userses
                .Include(u => u.Roles)
                .FirstOrDefaultAsync(u => u.Email == dto.Email);

            if (user == null)
                return Unauthorized(new { error = "Неверный email или пароль" });
            if (!BCrypt.Net.BCrypt.Verify(dto.Password, user.Passwordhash))
                return Unauthorized(new { error = "Неверный email или пароль" });
            var claims = new List<Claim>
    {
        new Claim(ClaimTypes.Name, user.Email),
        new Claim(ClaimTypes.NameIdentifier, user.Userid.ToString()),
        new Claim(ClaimTypes.Role, user.Roles?.Rolename ?? "User")
    };
            if (user.Clientid.HasValue)
                claims.Add(new Claim("ClientId", user.Clientid.Value.ToString()));
            if (user.Employeeid.HasValue)
                claims.Add(new Claim("EmployeeId",
user.Employeeid.Value.ToString()));
            var key = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_configuration["Jwt:Key"]));
            var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);
            var token = new JwtSecurityToken(
                issuer: _configuration["Jwt:Issuer"],
                audience: _configuration["Jwt:Audience"],
                claims: claims,
                expires: DateTime.Now.AddHours(3),
                signingCredentials: creds
            );
            return Ok(new
            {
                token = new JwtSecurityTokenHandler().WriteToken(token)
            });
        }
    }
}
```

Для работы с реализованными функциями в базе данных были создан сервис, который обращается к ним. Листинг сервиса для работы с функциями представлен в листинге 13.

Листинг 13 – AnalysisService

```
using Microsoft.EntityFrameworkCore;
using System;
using WebApplication1.DTOS;
namespace WebApplication1
{
    public class AnalysisService
    {
        private readonly CenterContext _context;

        public AnalysisService(CenterContext context)
        {
            _context = context;
        }
        public async Task<List<ClientApplicationDto>>
AnalyzeClientApplications(string phone)
        {
            return await _context.ClientApplicationDtos
                .FromSqlRaw("SELECT * FROM analize_client_application({0})", phone)
                .ToListAsync();
        }
        public async Task<List<OrganizationAnalysisDto>>
AnalyzeOrganizations(DateOnly? start, DateOnly? end)
        {
            return await _context.OrganizationAnalysisDtos
                .FromSqlRaw("SELECT * FROM analize_organization({0}, {1})", start,
end)
                .ToListAsync();
        }
        public async Task<List<BrigadeAnalysisDto>> AnalyzeBrigades(DateOnly? start,
DateOnly? end)
        {
            return await _context.BrigadeAnalysisDtos
                .FromSqlRaw("SELECT * FROM analize_brigade({0}, {1})", start, end)
                .ToListAsync();
        }
    }
}
```

Также для работы с функциями был разработан отдельный контроллер (листинг 14).

Листинг 14 - AnalysisController

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
namespace WebApplication1.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class AnalysisController : ControllerBase
    {
```

```
        private readonly AnalysisService _service;
        private readonly CenterContext _context;

        public AnalysisController(AnalysisService service, CenterContext context)
        {
            _service = service;
            _context = context;
        }
        [Authorize(Roles = "Admin,Client")]
        [HttpGet("Client")]
        public async Task<IActionResult> GetClientApplications([FromQuery] string?
phone = null)
        {
            try
            {
                // Если пользователь - Client, вытаскиваем телефон из базы по его
ClientId из токена
                if (User.IsInRole("Client"))
                {
                    var clientIdStr = User.FindFirst("ClientId")?.Value;
                    if (string.IsNullOrEmpty(clientIdStr)) return Unauthorized("Не
удалось получить идентификатор клиента");

                    var clientId = int.Parse(clientIdStr);
                    var client = await _context.Clients.FindAsync(clientId);
                    if (client == null) return NotFound("Клиент не найден");

                    phone = client.Phone;
                }

                if (string.IsNullOrEmpty(phone))
                    return BadRequest("Необходимо указать номер телефона");

                var result = await _service.AnalyzeClientApplications(phone);
                return Ok(result);
            }
            catch (Exception ex)
            {
                return BadRequest($"Ошибка: {ex.Message}");
            }
        }
        [Authorize(Roles = "Admin,Employee")]
        [HttpGet("organization")]
        public async Task<IActionResult> GetOrganizationAnalysis([FromQuery]
DateOnly? start, [FromQuery] DateOnly? end)
        {
            try
            {
                var result = await _service.AnalyzeOrganizations(start, end);
                return Ok(result);
            }
            catch (Exception ex)
            {
                return BadRequest(ex.Message);
            }
        }
        [Authorize(Roles = "Admin,Employee")]
        [HttpGet("brigade")]
        public async Task<IActionResult> GetBrigadeAnalysis([FromQuery] DateOnly?
start, [FromQuery] DateOnly? end)
        {
            try
```

```
            {
                var result = await _service.AnalyzeBrigades(start, end);
                return Ok(result);
            }
            catch (Exception ex)
            {
                return BadRequest(ex.Message);
            }
        }
    }
}
```

Также были разработаны отдельные контроллеры для заказчика, обеспечивающие доступ ко всем необходимым данным и предоставляющие возможность полноценного взаимодействия с системой. Они хранятся в отдельной папке ClientControllers. Созданные контроллеры в папке ClientControllers отображены на рисунке 23.

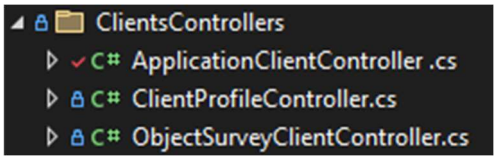


Рисунок 23 – Созданные контроллеры в папке ClientControllers

Их реализация представлена в приложении Б.

6. Экспериментальная част

После разработки сущностных классов и контроллеров необходимо осуществить проверку работоспособности серверной части. Проверка осуществлялась в Swagger.

Для регистрации нового пользователя необходимо заполнить данные и выполнить запрос (рисунок 24).

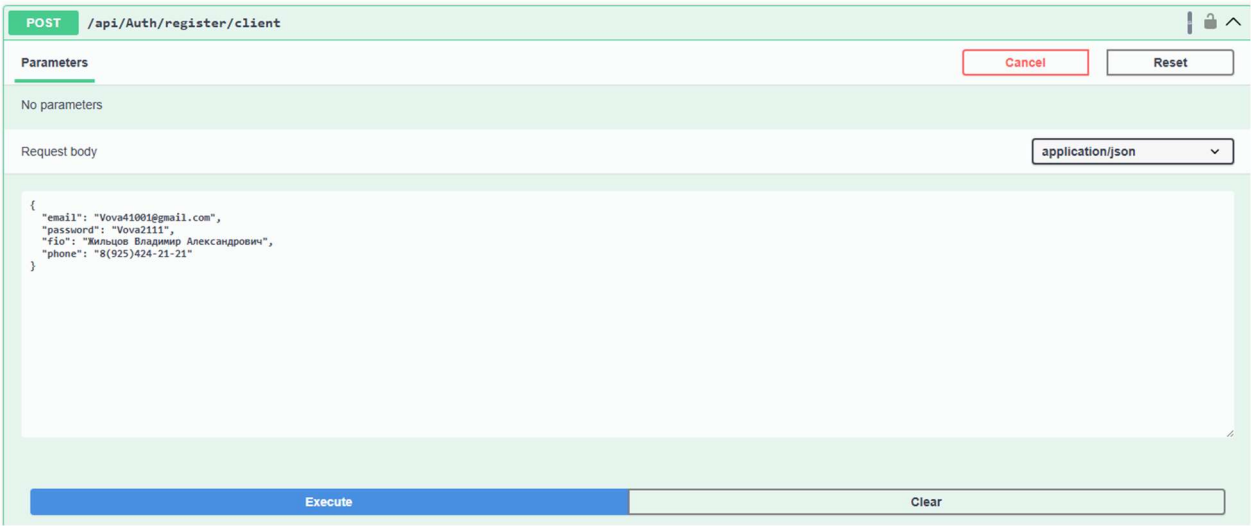


Рисунок 24 – Регистрация нового пользователя

Рисунок 25 – Результат регистрации

Данная запись была добавлена в базу данных (рисунок 26 – 27).

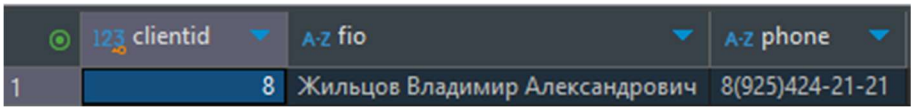


Рисунок 26 – Добавленная запись в таблицу Client

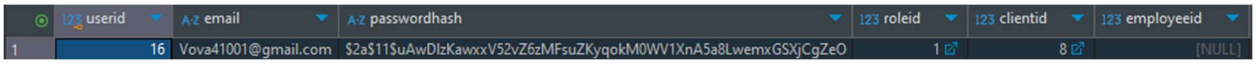


Рисунок 27 – Добавленная запись в таблицу Users

Для осуществления авторизации пользователя необходимо ввести Email и пароль, а затем получить токен. Данный токен необходимо скопировать в раздел авторизации пользователя и осуществить вход (рисунок 28 - 31).

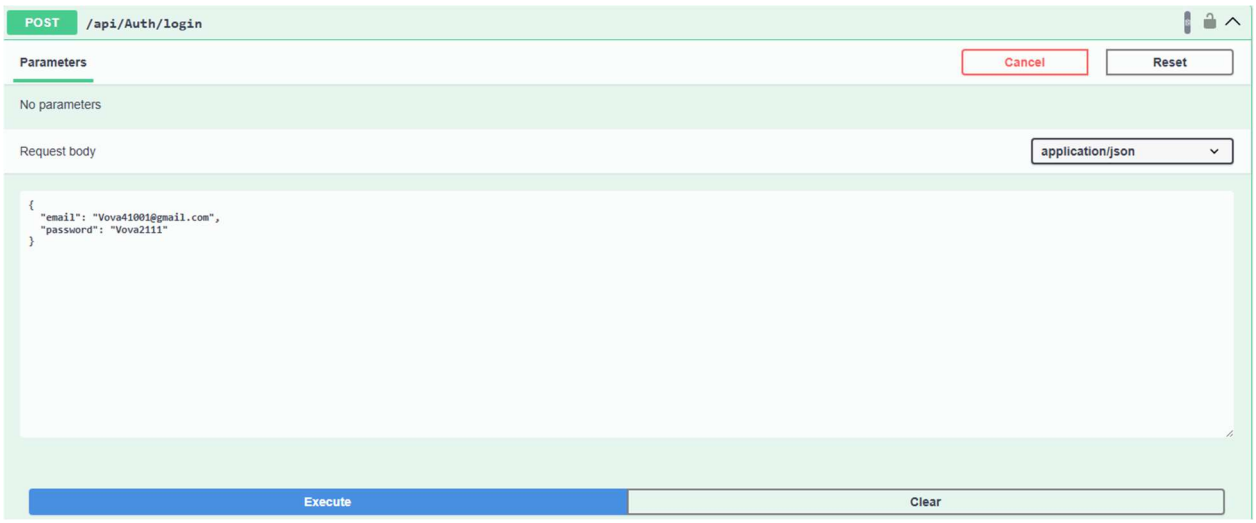


Рисунок 28 – Получение токена пользователем

Рисунок 29 – Выданный токен пользователю

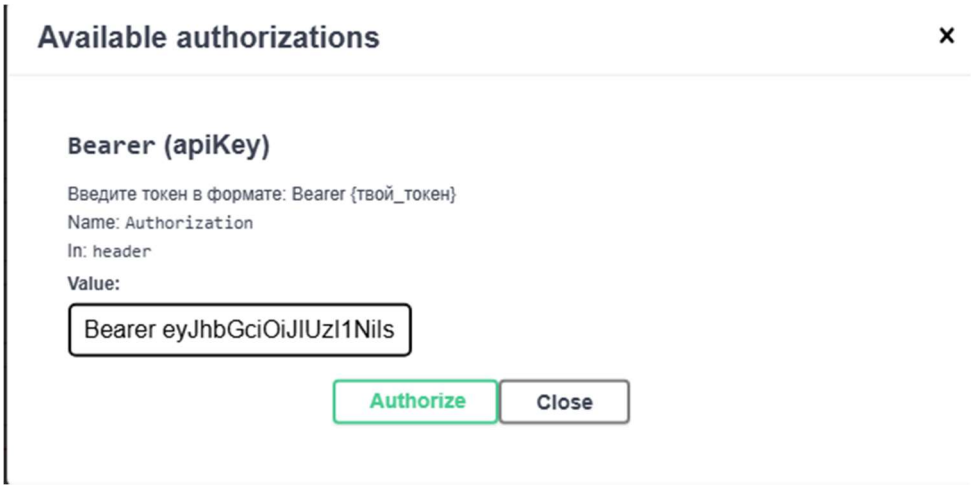


Рисунок 30 – Авторизация пользователя

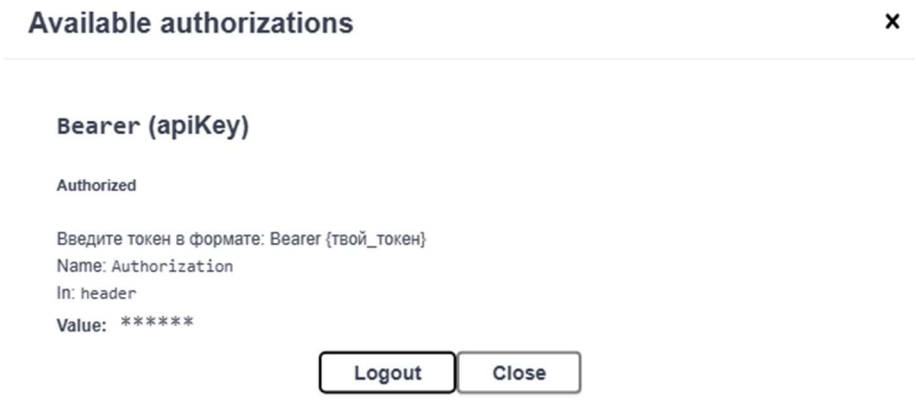


Рисунок 31 – Авторизация пользователя

Просмотрим всю информацию, доступную заказчику. Заказчик может просматривать свои объекты обследования, свои заявки свои выбранные процедуры, готовые отчеты по его заявкам и договоры (рисунок 32 -).

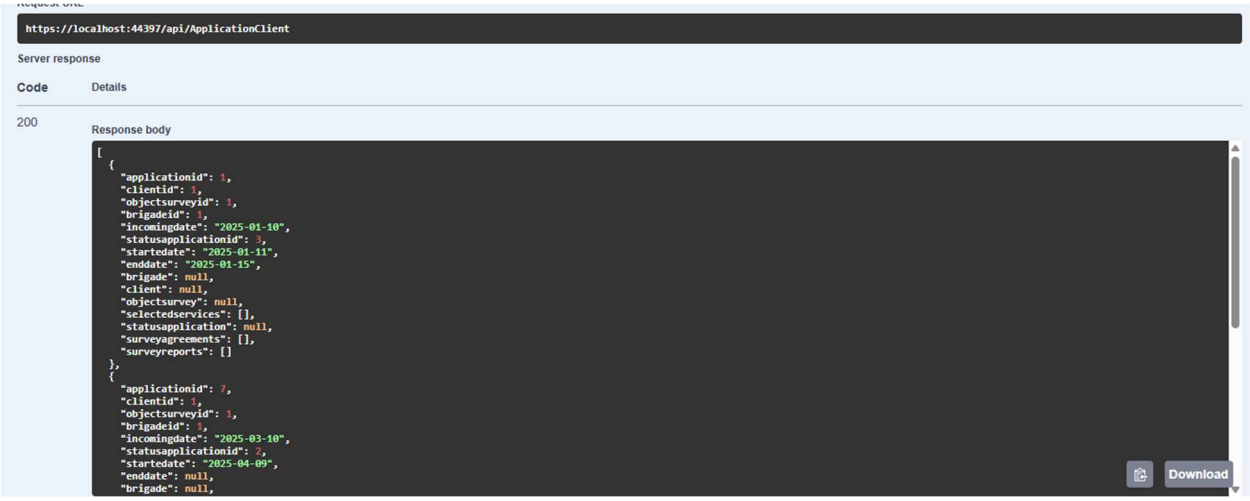


Рисунок 32 – Заявки клиента

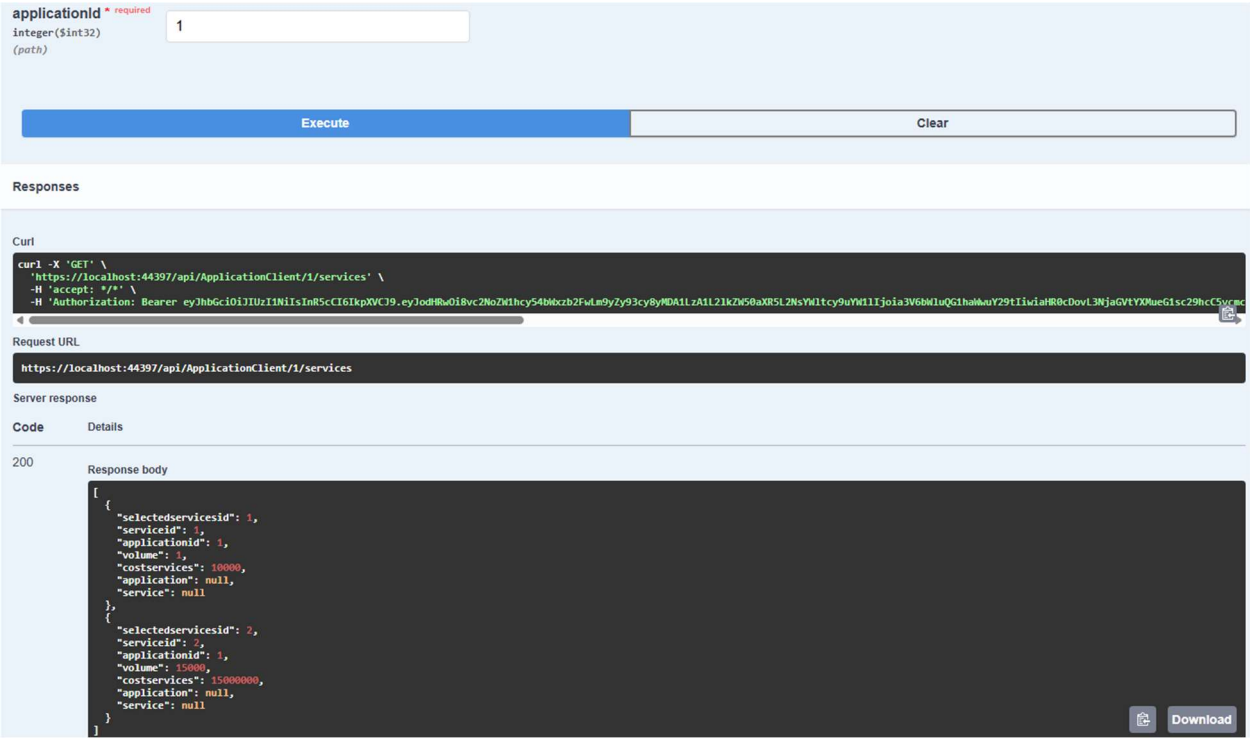


Рисунок 33 – Выбранные услуги к заявке с идентификатором 1

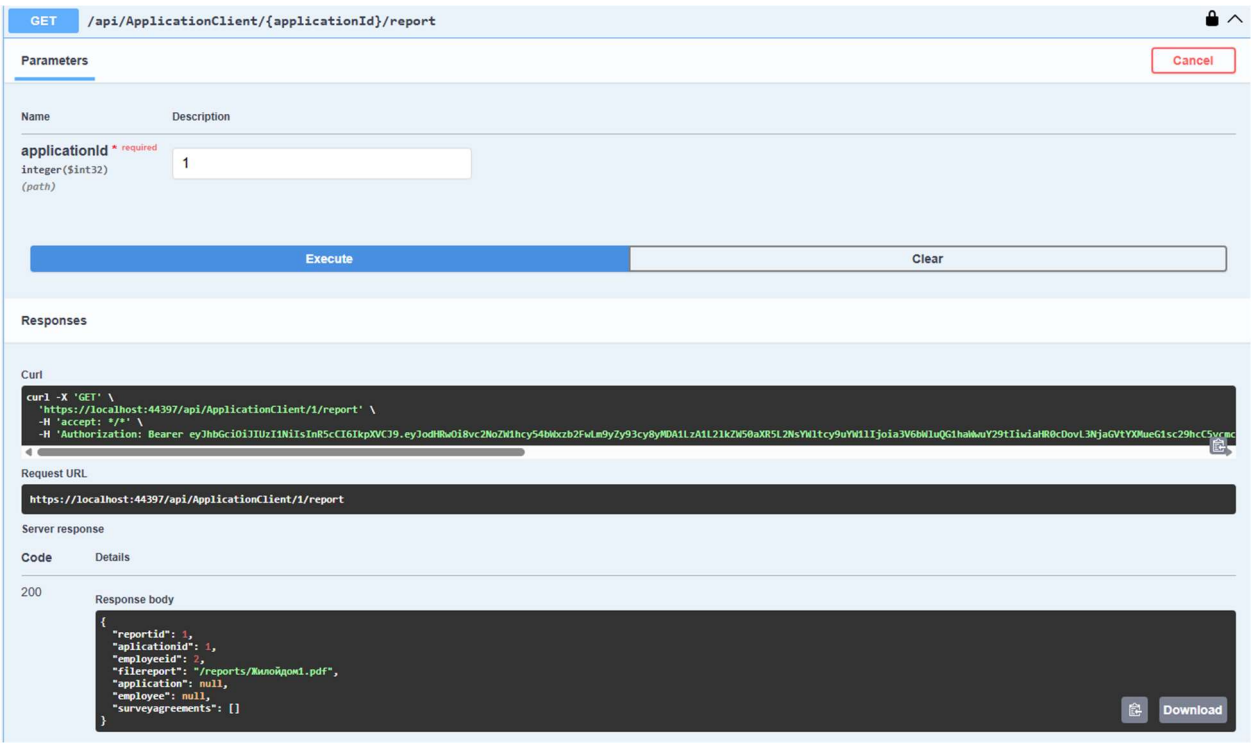


Рисунок 34 – Отчет к заявке с идентификатором 1

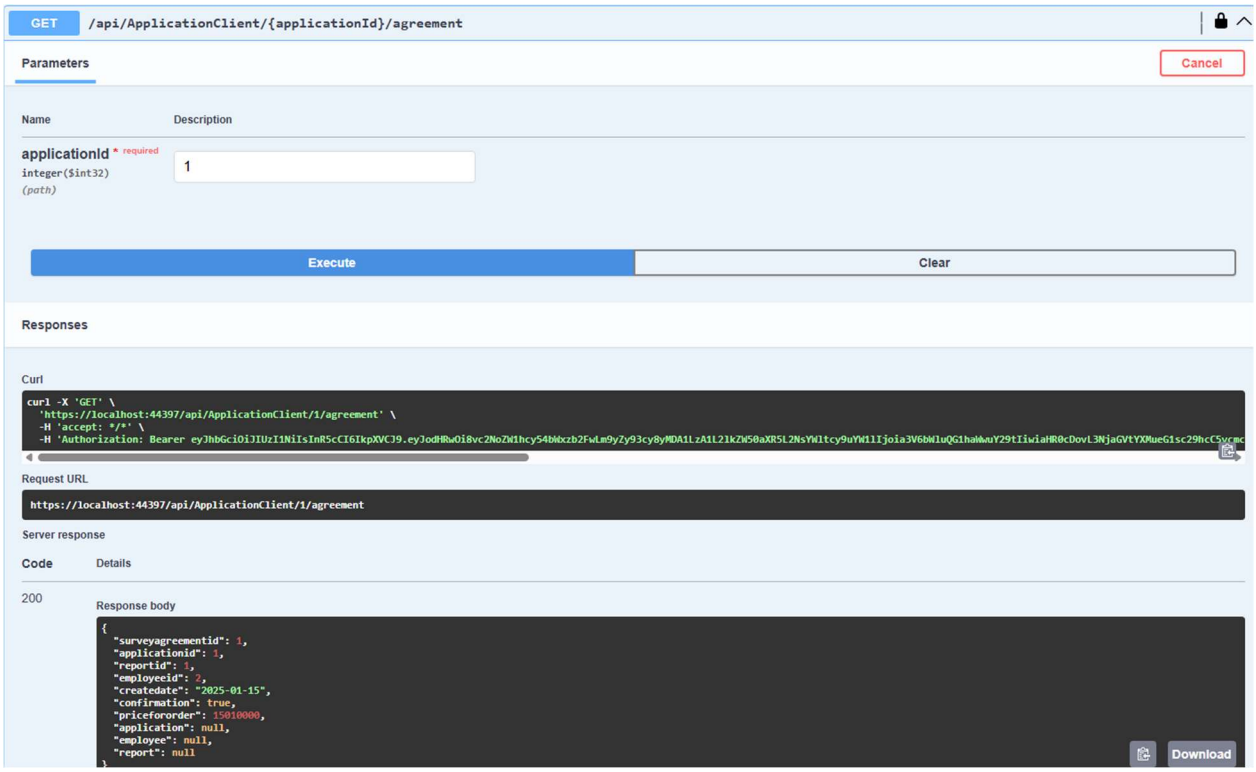


Рисунок 35 – Договор к заявке с идентификатором 1

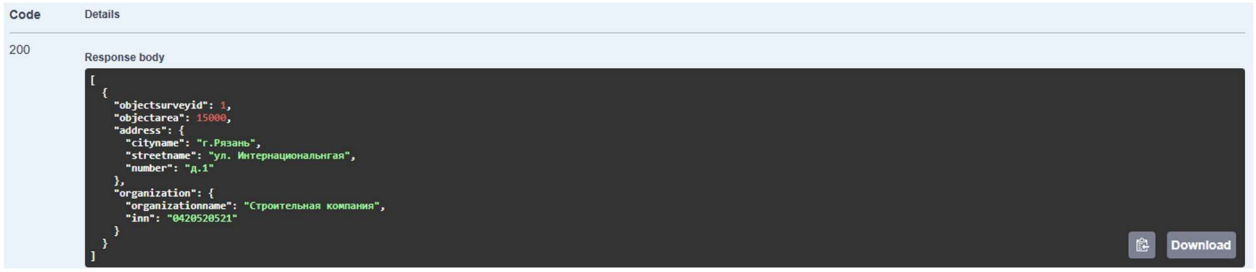


Рисунок 34 – Объекты обследования клиента

Заключение

В ходе преддипломной практики, проходившей в организации ООО «Центр исследования строительных конструкций и материалов», были выполнены следующие задачи:

1. настроено подключение к базе данных;
2. разработаны сущностные классы, соответствующие структуре базы данных;
3. реализована серверная часть информационной системы с использованием языка программирования C# и технологии ASP.NET Core.

Так как поставленные задачи были выполнены можно сделать вывод о том, что цель практики достигнута. Разработав и протестировав систему, был получен успешный результат, который удовлетворяет поставленным в начале задачам. В процессе выполнения выпускной квалификационной работы планируется добавление интерфейса.

Список литературы

1. Применение системы мониторинга технического состояния строительных конструкций // RIOR Научные публикации. URL: https://riorpub.com/ru/nauka/article/71994/view (дата обращения: 20.01.2025).

2. Основные понятия ER – диаграмм: сущность, экземпляр сущности, атрибуты сущности, ключ сущности, связь // Студопедия URL: https://studopedia.ru/8_43410_osnovnie-ponyatiya-ER--diagramm-sushchnost-ekzemplyar-sushchnosti-atributi-sushchnosti-klyuch-sushchnosti-svyaz.html (дата обращения: 16.02.2025).

3. Коннолли Т., Бегг К., Страчан А. Базы данных: проектирование, реализация и сопровождение. — М.: Издательский дом «Вильямс», 2011. — 1488 с.

4. Маркин А.В. Программирование на SQL в 2 ч. Часть 1. - 2-е изд. - Москва: Издательство Юрайт, 2019. - 403 с.

5. Маркин А.В. Программирование на SQL в 2 ч. Часть 2. - 2-е изд. - Москва: Издательство Юрайт, 2019. - 403 с.

6. Entity Framework Documentation // Microsoft Learn. URL: https://learn.microsoft.com/en-us/ef/ (дата обращения: 13.05.2025).

7. JSON Web Token (JWT) — введение и примеры использования // Metanit. URL: https://metanit.com/web/net/8.1.php (дата обращения: 15.05.2025).

8. CRUD-операции в веб-разработке: теория и практика // Metanit. URL: https://metanit.com/web/mvc/5.3.php (дата обращения: 16.05.2025).

Приложение А
Скрипт Базы данных

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Рязанский государственный радиотехнический университет имени В.Ф. Уткина»

(ФГБОУ ВО «РГРТУ», РГРТУ)

Кафедра Автоматизированных Систем Управления

«Разработка информационной системы центра исследований строительных конструкций»

Скрипт Базы данных

Листов 19

Рязань 2025

```
-- Создание базы данных
create DATABASE "Center"
    ENCODING 'UTF8'
    LC_COLLATE 'en_US.UTF-8'
    LC_CTYPE 'en_US.UTF-8'
    TEMPLATE template0;
-- Создание таблицы Заказчик
create table Client(
      ClientId SERIAL primary key,
      Fio VARCHAR (40) NOT NULL,
      Phone VARCHAR (18) unique NOT null
);
-- Создание таблицы Должность
create table JobTitle(
      JobTitleId Serial primary key,
      JobTitleName VARCHAR (30) NOT NULL
);
-- Создание таблицы Бригада
create table Brigade(
      BrigadeId Serial primary key,
      Brigadename VARCHAR (20) NOT NULL
);
-- Создание таблицы Сотрудник
create table Employee(
      Employeeid Serial primary key,
      Fio VARCHAR (40) NOT NULL,
      Phone VARCHAR (18) unique NOT NULL,
      JobTitleId int REFERENCES JobTitle(JobTitleId) ON DELETE SET NULL,
      BrigadeId int REFERENCES Brigade(BrigadeId) ON DELETE SET NULL
);

-- Создание таблицы Ролей
create table Roles(
      RoleId serial primary key,
      Rolename varchar(10) not null
);

-- Создание таблицы Пользователей
create table Users(
      UserId serial primary key,
      Email VARCHAR (100) unique NOT null,
      PasswordHash VARCHAR(256) NOT NULL,
      RoleId int not null REFERENCES Roles(RoleId) ON DELETE RESTRICT,
      ClientId int REFERENCES Client(ClientId) on DELETE cascade,
      Employeeid int REFERENCES Employee(Employeeid) on DELETE cascade,
      CHECK (
    (ClientId IS NOT NULL AND EmployeeId IS NULL) OR
    (ClientId IS NULL AND EmployeeId IS NOT NULL))
);
-- Создание таблицы Адрес
create table Address(
      AddressId Serial primary key,
      CityName VARCHAR (20) NOT NULL,
      StreetName VARCHAR (40) NOT NULL,
      Number VARCHAR (10) NOT NULL
);

-- Создание таблицы Организация
create table Organization(
      OrganizationId Serial primary key,
      OrganizationName VARCHAR (30) NOT NULL,
      INN VARCHAR (12) unique NOT NULL
```

```
);

-- Создание таблицы Объект обследования
create table ObjectSurvey(
	ObjectSurveyId Serial primary key,
	ClientId int references Client(ClientId) on delete set null,
	AddressId int REFERENCES Address(AddressId) ON DELETE SET NULL,
	OrganizationId int REFERENCES Organization(OrganizationId) ON DELETE SET NULL,
	ObjectArea FLOAT NOT null check (ObjectArea>0)
);

-- Создание Статус заявки на обследование
create table StatusApplication(
	StatusApplicationId Serial primary key,
	TypeStatus VARCHAR (15) NOT NULL
);

-- Создание таблицы Заявка на обследование
create table Application(
	ApplicationId Serial primary key,
	ClientId int not null REFERENCES Client(ClientId) ON DELETE SET NULL,
	ObjectSurveyId int not null REFERENCES ObjectSurvey(ObjectSurveyId) ON DELETE
SET null,
	BrigadeId int REFERENCES Brigade(BrigadeId) ON DELETE SET null,
	IncomingDate date not null default (current_date),
	StatusApplicationId int not null REFERENCES
StatusApplication(StatusApplicationId) ON DELETE SET null,
	StarteDate date check (IncomingDate<StarteDate),
	EndDate date check (StarteDate<EndDate)
);

-- Создание таблицы Каталог услуг
create table ServiceCatalog(
	ServiceId  Serial primary key,
	ServiceName  VARCHAR (100) NOT null,
	Price  NUMERIC(10,2) NOT null check (Price>0),
	Measurement varchar (30) NOT null,
	Description text not NULL
);

-- Создание таблицы Выбранные услуги
create table SelectedServices(
	SelectedServicesId  Serial primary key,
	ServiceId  int REFERENCES ServiceCatalog(ServiceId) ON DELETE SET null on
update cascade,
	ApplicationId int REFERENCES Application(ApplicationId) ON DELETE CASCADE,
	Volume FLOAT NOT null check (Volume>0),
	CostServices NUMERIC(15,2) check (CostServices>0)
);

-- Создание таблицы Отчет об обследовании
create table SurveyReport(
	ReportId Serial primary key,
	AplicationId int REFERENCES Application(ApplicationId) ON DELETE RESTRICT,
	EmployeeId int REFERENCES Employee(EmployeeId) ON DELETE SET null,
	FileReport text not NULL
);

-- Создание таблицы Договор
create table SurveyAgreement(
	SurveyAgreementId Serial primary key,
	Applicationid int REFERENCES Application(ApplicationId) ON DELETE RESTRICT,
```

```
	ReportId int REFERENCES SurveyReport(ReportId) ON DELETE set null,
	EmployeeId int REFERENCES Employee(EmployeeId) ON DELETE SET null,
	CreateDate date not null default(current_date),
	Confirmation BOOLEAN not null default false,
	PriceForOrder NUMERIC (20,2) check (PriceForOrder>0)
);
-- скрипт добавления данных в базу данных
insert into client(FIO,Phone) values('Кузьмин Сергей Иванович','8(905)584-98-94');
insert into client(FIO,Phone) values('Евсеев Адам Романович','8(910)952-05-13');
insert into client(FIO,Phone) values('Пахомова Алина Михайловна','8(950)005-31-93');
insert into client(FIO,Phone) values('Мальцев Константин Артёмович','8(942)507-57-
05');
insert into client(FIO,Phone) values('Романова Валентина Александровна','8(952)174-
27-75');

Insert into JobTitle(JobTitleName) values('Директор');
Insert into JobTitle(JobTitleName) values('Главный инженер проектировщик');
Insert into JobTitle(JobTitleName) values('Инженер проектировщик');
Insert into JobTitle(JobTitleName) values('Менеджер');

insert into StatusApplication(TypeStatus) values('Принята');
insert into StatusApplication(TypeStatus) values('Выполняется');
insert into StatusApplication(TypeStatus) values('Закончена');

INSERT INTO ServiceCatalog(ServiceName, Price, Measurement, Description) VALUES
('Составление отчета', 10000.00, 'ШТ', 'Наши специалисты составят отчет, в котором
будут указаны проблемные места конструкции, а также рекомендации по их устранению');
INSERT INTO ServiceCatalog(ServiceName, Price, Measurement, Description) VALUES
('Обследование технического состояния несущих и ограждающих конструкций', 350.00,
'М2', 'Проводится детальное обследование несущих и ограждающих конструкций здания на
предмет повреждений, деформаций и нарушений нормативов');
INSERT INTO ServiceCatalog(ServiceName, Price, Measurement, Description) VALUES
('Поиск и обследование технического состояния инженерных коммуникаций', 400.00, 'М2',
'Выполняется поиск и диагностика состояния внутренних коммуникаций: водоснабжения,
канализации, электрики и вентиляции');
INSERT INTO ServiceCatalog(ServiceName, Price, Measurement, Description) VALUES
('Обследование технического состояния инженерных систем', 450.00, 'М2', 'Оценивается
общее техническое состояние инженерных систем здания с предоставлением экспертного
заключения');
INSERT INTO ServiceCatalog(ServiceName, Price, Measurement, Description) VALUES
('Обмерные работы', 200.00, 'М2', 'Проводятся точные обмеры помещения или здания с
составлением чертежей и схем для последующего проектирования или ремонта');

insert into brigade(BrigadeName) values('Бригада №1');
insert into brigade(BrigadeName) values('Бригада №2');

insert into employee(Fio,Phone,JobTitleId,BrigadeId)values ('Ермолов Леон
Маркович','8(910)293-00-97',1,null);
insert into employee(Fio,Phone,JobTitleId,BrigadeId)values ('Симонов Николай
Егорович','8(965)867-22-69',2,1);
insert into employee(Fio,Phone,JobTitleId,BrigadeId)values ('Рябов Никита
Макарович','8(971)301-14-78',2,2);
insert into employee(Fio,Phone,JobTitleId,BrigadeId)values ('Григорьев Илья
Кириллович','8(944)991-09-27',3,1);
insert into employee(Fio,Phone,JobTitleId,BrigadeId)values ('Лобанов Виктор
Евгеньевич','8(956)295-85-89',3,2);
insert into employee(Fio,Phone,JobTitleId,BrigadeId)values ('Уткина Лея
Дмитриевна','8(985)874-96-44',4,null);

insert into address(cityname,streetname,"number") values('г.Рязань','ул.
Интернациональнгая','д.1');
```

```
insert into address(cityname,streetname,"number") values('г.Рязань','ул.
Прижелезнодорожная','д.52');
insert into address(cityname,streetname,"number") values('г.Рязань','Московское
шоссе','д.5А');
insert into address(cityname,streetname,"number") values('г.Рязань','ул.
Бирюзова','д.28А');
insert into address(cityname,streetname,"number") values('г.Рязань','ул.
Интернациональнгая','д.23');
insert into address(cityname,streetname,"number") values('г.Рязань','шоссе
Солотчинское','д.11');

insert into organization(organizationname,inn) values('Строительная
компания','0420520521');
insert into organization(organizationname,inn) values('Русская кожа','0602065848');
insert into organization(organizationname,inn) values('тц. Барс','6227009810');
insert into organization(organizationname,inn) values('Прайм','6234203335');
insert into organization(organizationname,inn) values('Круиз','6234085000');

insert into objectsurvey(ClientId,addressid,organizationid,objectarea)
values(1,1,1,15000.00);
insert into objectsurvey(ClientId,addressid,organizationid,objectarea)
values(2,2,2,160000.00);
insert into objectsurvey(ClientId,addressid,organizationid,objectarea)
values(3,3,3,36500.00);
insert into objectsurvey(ClientId,addressid,organizationid,objectarea)
values(4,4,4,1500.00);
insert into objectsurvey(ClientId,addressid,organizationid,objectarea)
values(4,5,4,1000.00);
insert into objectsurvey(ClientId,addressid,organizationid,objectarea)
values(5,5,5,27000.00);

insert into
application(clientid,objectsurveyid,brigadeid,incomingdate,statusapplicationid,starte
date,enddate) values(1,1,1,'2025-01-10',3,'2025-01-11','2025-01-15');
insert into
application(clientid,objectsurveyid,brigadeid,incomingdate,statusapplicationid,starte
date,enddate) values(2,2,2,'2025-01-15',3,'2025-01-16','2025-01-18');
insert into
application(clientid,objectsurveyid,brigadeid,incomingdate,statusapplicationid,starte
date,enddate) values(3,3,1,'2025-02-01',3,'2025-02-02','2025-02-05');
insert into
application(clientid,objectsurveyid,brigadeid,incomingdate,statusapplicationid,starte
date,enddate) values(4,4,2,'2025-02-10',3,'2025-02-11','2025-02-13');
insert into
application(clientid,objectsurveyid,brigadeid,incomingdate,statusapplicationid,starte
date,enddate) values(4,5,1,'2025-02-20',3,'2025-02-21','2025-02-25');
insert into
application(clientid,objectsurveyid,brigadeid,incomingdate,statusapplicationid,starte
date,enddate) values(5,6,2,'2025-03-01',2,'2025-04-07',null);
insert into
application(clientid,objectsurveyid,brigadeid,incomingdate,statusapplicationid,starte
date,enddate) values(1,1,1,'2025-03-10',2,'2025-04-09',null);
insert into
application(clientid,objectsurveyid,brigadeid,incomingdate,statusapplicationid,starte
date,enddate) values(2,2,2,'2025-03-20',1,null,null);

insert into SelectedServices(ServiceId,ApplicationId,volume,costservices)
values(1,1,1,10000.00);
insert into SelectedServices(ServiceId,ApplicationId,volume,costservices)
values(2,1,15000.00,15000000.00);
insert into SelectedServices(ServiceId,ApplicationId,volume,costservices)
values(1,2,1,10000.00);
```

```
insert into SelectedServices(ServiceId,ApplicationId,volume,costservices)
values(3,2,5000.00,1000000.00);
insert into SelectedServices(ServiceId,ApplicationId,volume,costservices)
values(1,3,1,10000.00);
insert into SelectedServices(ServiceId,ApplicationId,volume,costservices)
values(4,3,15000.00,15000000.00);
insert into SelectedServices(ServiceId,ApplicationId,volume,costservices)
values(1,4,1,10000.00);
insert into SelectedServices(ServiceId,ApplicationId,volume,costservices)
values(5,4,1500.00,262500.00);
insert into SelectedServices(ServiceId,ApplicationId,volume,costservices)
values(1,5,1,10000.00);
insert into SelectedServices(ServiceId,ApplicationId,volume,costservices)
values(2,5,1000.00,1000000.00);
insert into SelectedServices(ServiceId,ApplicationId,volume,costservices)
values(1,6,1,10000.00);
insert into SelectedServices(ServiceId,ApplicationId,volume,costservices)
values(3,6,27000.00,5400000.00);
insert into SelectedServices(ServiceId,ApplicationId,volume,costservices)
values(1,7,1,10000.00);
insert into SelectedServices(ServiceId,ApplicationId,volume,costservices)
values(4,7,15000.00,15000000.00);
insert into SelectedServices(ServiceId,ApplicationId,volume,costservices)
values(1,8,1,10000.00);
insert into SelectedServices(ServiceId,ApplicationId,volume,costservices)
values(5,8,20000.00,350000.00);

insert into SurveyReport(aplicationid,employeeid,filereport)
values(1,2,'/reports/Жилойдом1.pdf');
insert into SurveyReport(aplicationid,employeeid,filereport)
values(2,4,'/reports/Кожзавод1.pdf');
insert into SurveyReport(aplicationid,employeeid,filereport)
values(3,3,'/reports/Барс1.pdf');
insert into SurveyReport(aplicationid,employeeid,filereport)
values(4,5,'/reports/Прайм1.pdf');
insert into SurveyReport(aplicationid,employeeid,filereport)
values(5,2,'/reports/Прайм2.pdf');

insert into
SurveyAgreement(applicationid,reportid,employeeid,createdate,confirmation,priceforord
er) values(1,1,2,'2025-01-15',true,15010000.00);
insert into
SurveyAgreement(applicationid,reportid,employeeid,createdate,confirmation,priceforord
er) values(2,2,4,'2025-01-18',true,1010000.00);
insert into
SurveyAgreement(applicationid,reportid,employeeid,createdate,confirmation,priceforord
er) values(3,3,3,'2025-02-05',false,15010000.00);
insert into
SurveyAgreement(applicationid,reportid,employeeid,createdate,confirmation,priceforord
er) values(4,4,5,'2025-02-13',true,272500.00);
insert into
SurveyAgreement(applicationid,reportid,employeeid,createdate,confirmation,priceforord
er) values(5,5,2,'2025-02-25',true,1010000.00);

INSERT INTO Roles(Rolename) VALUES ('Client');
INSERT INTO Roles(Rolename) VALUES ('Employee');
INSERT INTO Roles(Rolename) VALUES ('Admin');

insert into Users(Email, PasswordHash, RoleId, ClientId) values ('kuzmin@mail.com',
'hash123', 1, 1);
insert into Users(Email, PasswordHash, RoleId, ClientId) values ('Evceev@mail.com',
'hash234', 1, 2);
```

```
insert into Users(Email, PasswordHash, RoleId, ClientId) values ('Paxomova@mail.com',
'hash345', 1, 3);
insert into Users(Email, PasswordHash, RoleId, ClientId) values ('Malcev@mail.com',
'hash456', 1, 4);
insert into Users(Email, PasswordHash, RoleId, ClientId) values ('Romanov@mail.com',
'hash567', 1, 5);
insert into Users(Email, PasswordHash, RoleId, EmployeeId) values
('Ermolov@mail.com', 'hash111', 3, 1);
insert into Users(Email, PasswordHash, RoleId, EmployeeId) values
('Simonov@mail.com', 'hash222', 2, 2);
insert into Users(Email, PasswordHash, RoleId, EmployeeId) values ('rabov@mail.com',
'hash333', 2, 3);
insert into Users(Email, PasswordHash, RoleId, EmployeeId) values
('Grigorger@mail.com', 'hash444', 2, 4);
insert into Users(Email, PasswordHash, RoleId, EmployeeId) values
('Lobanov@mail.com', 'hash555', 2, 5);
insert into Users(Email, PasswordHash, RoleId, EmployeeId) values ('Ytkina@mail.com',
'hash666', 2, 6);
-- скрипты создания хранимых функций
--Функция для анализа заявок клиента
CREATE OR REPLACE FUNCTION analize_client_application(sPhone VARCHAR(18))
RETURNS TABLE (
    cityname VARCHAR(20),
    streetname VARCHAR(20),
    number VARCHAR(10),
    objectarea FLOAT,
    organizationname VARCHAR(30),
    inn VARCHAR(12),
    brigadename VARCHAR(40),
    incomingdate DATE,
    status VARCHAR(12),
    startedate DATE,
    enddate DATE,
    createdate DATE
)
AS $$
BEGIN
    IF sPhone IS NULL THEN
        RAISE EXCEPTION 'Ошибка: Телефонный номер заказчика не может быть null';
    END IF;

    IF NOT EXISTS (SELECT 1 FROM client WHERE phone = sPhone) THEN
        RAISE EXCEPTION 'Ошибка: Данный клиент не найден';
    END IF;

    IF NOT EXISTS (
        SELECT 1
        FROM application
        WHERE clientid = (SELECT clientid FROM client WHERE phone = sPhone)
    ) THEN
        RAISE EXCEPTION 'Ошибка: У данного клиента нет заявок';
    END IF;

    RETURN QUERY
    SELECT
        a2.cityname,
        a2.streetname,
        a2.number,
        o.objectarea,
        org.organizationname,
        org.inn,
        b.brigadename,
```

```
        a.incomingdate,
        s2.TypeStatus,
        a.startedate,
        a.enddate,
        s.createdate
    FROM application a
    JOIN objectsurvey o USING (objectsurveyid)
    JOIN address a2 USING (addressid)
    JOIN organization org USING (organizationid)
    JOIN brigade b USING (brigadeid)
    left JOIN surveyagreement s USING (ApplicationId)
      Join StatusApplication s2 USING(StatusApplicationId)
    WHERE a.clientid = (SELECT clientid FROM client WHERE phone = sPhone);
END;
$$ LANGUAGE plpgsql;

-- Функция для анализа прибыли орагнизаций
CREATE OR REPLACE FUNCTION analize_organization(nstartdate date, nenddate date)
RETURNS TABLE (
    organizationname VARCHAR(30),
    inn VARCHAR(12),
    sumprice NUMERIC(20,2),
    countapplication BIGINT,
    percentapplication NUMERIC(5,2),
    countagreement BIGINT,
    percentagreement NUMERIC(5,2),
    firstapplicationdate DATE,
    lastapplicationdate DATE
)
AS $$
BEGIN
    IF NOT EXISTS ( SELECT 1 FROM organization) THEN
        RAISE EXCEPTION 'Ошибка: Нет организаций';
    END IF;

    IF NOT EXISTS (SELECT 1 FROM application) THEN
        RAISE EXCEPTION 'Ошибка: Нет заявок';
    END IF;

      if nstartdate is null and nenddate is null then
    RETURN QUERY
    SELECT
        o.organizationname,
        o.inn,
        SUM(CASE WHEN s.confirmation IS TRUE THEN s.priceforорder ELSE 0 END) AS
sumprice,
        COUNT(DISTINCT a.applicationid) AS countapplication,
        ROUND(100.0 * SUM(CASE WHEN a.StatusApplicationId = (select
StatusApplicationId from StatusApplication where TypeStatus = 'Закончена') THEN 1
ELSE 0 END) / NULLIF(COUNT(a.applicationid), 0), 2) AS percentapplication,
        COUNT(DISTINCT s.surveyagreementid) AS countagreement,
        ROUND(100.0 * SUM(CASE WHEN s.confirmation IS TRUE THEN 1 ELSE 0 END) /
NULLIF(COUNT(s.surveyagreementid), 0), 2) AS percentagreement,
        MIN(a.incomingdate) AS firstapplicationdate,
        MAX(a.incomingdate) AS lastapplicationdate
    FROM organization o JOIN objectsurvey os USING (organizationid) JOIN application
a USING (objectsurveyid) JOIN surveyagreement s using(applicationid)
    GROUP BY o.organizationname, o.inn;
      END IF;

      if nstartdate is null and nenddate is not null then
    RETURN QUERY
```

```
    SELECT
        o.organizationname,
        o.inn,
        SUM(CASE WHEN s.confirmation IS TRUE THEN s.priceforорder ELSE 0 END) AS
sumprice,
        COUNT(DISTINCT a.applicationid) AS countapplication,
        ROUND(100.0 * SUM(CASE WHEN a.StatusApplicationId = (select
StatusApplicationId from StatusApplication where TypeStatus = 'Закончена') THEN 1
ELSE 0 END) / NULLIF(COUNT(a.applicationid), 0), 2) AS percentapplication,
        COUNT(DISTINCT s.surveyagreementid) AS countagreement,
        ROUND(100.0 * SUM(CASE WHEN s.confirmation IS TRUE THEN 1 ELSE 0 END) /
NULLIF(COUNT(s.surveyagreementid), 0), 2) AS percentagreement,
        MIN(a.incomingdate) AS firstapplicationdate,
        MAX(a.incomingdate) AS lastapplicationdate
    FROM organization o JOIN objectsurvey os USING (organizationid) JOIN application
a USING (objectsurveyid) JOIN surveyagreement s using(applicationid)
       where s.createdate <= nenddate
    GROUP BY o.organizationname, o.inn;
      END IF;

      if nstartdate is not null and nenddate is null then
    RETURN QUERY
    SELECT
        o.organizationname,
        o.inn,
        SUM(CASE WHEN s.confirmation IS TRUE THEN s.priceforорder ELSE 0 END) AS
sumprice,
        COUNT(DISTINCT a.applicationid) AS countapplication,
        ROUND(100.0 * SUM(CASE WHEN a.StatusApplicationId = (select
StatusApplicationId from StatusApplication where TypeStatus = 'Закончена') THEN 1
ELSE 0 END) / NULLIF(COUNT(a.applicationid), 0), 2) AS percentapplication,
        COUNT(DISTINCT s.surveyagreementid) AS countagreement,
        ROUND(100.0 * SUM(CASE WHEN s.confirmation IS TRUE THEN 1 ELSE 0 END) /
NULLIF(COUNT(s.surveyagreementid), 0), 2) AS percentagreement,
        MIN(a.incomingdate) AS firstapplicationdate,
        MAX(a.incomingdate) AS lastapplicationdate
    FROM organization o JOIN objectsurvey os USING (organizationid) JOIN application
a USING (objectsurveyid) JOIN surveyagreement s using(applicationid)
       where s.createdate >= nstartdate
    GROUP BY o.organizationname, o.inn;
      END IF;

      if nstartdate is not null and nenddate is not null then
    RETURN QUERY
    SELECT
        o.organizationname,
        o.inn,
        SUM(CASE WHEN s.confirmation IS TRUE THEN s.priceforорder ELSE 0 END) AS
sumprice,
        COUNT(DISTINCT a.applicationid) AS countapplication,
        ROUND(100.0 * SUM(CASE WHEN a.StatusApplicationId = (select
StatusApplicationId from StatusApplication where TypeStatus = 'Закончена') THEN 1
ELSE 0 END) / NULLIF(COUNT(a.applicationid), 0), 2) AS percentapplication,
        COUNT(DISTINCT s.surveyagreementid) AS countagreement,
        ROUND(100.0 * SUM(CASE WHEN s.confirmation IS TRUE THEN 1 ELSE 0 END) /
NULLIF(COUNT(s.surveyagreementid), 0), 2) AS percentagreement,
        MIN(a.incomingdate) AS firstapplicationdate,
        MAX(a.incomingdate) AS lastapplicationdate
    FROM organization o JOIN objectsurvey os USING (organizationid) JOIN application
a USING (objectsurveyid) JOIN surveyagreement s using(applicationid)
       where s.createdate between nstartdate and nenddate
    GROUP BY o.organizationname, o.inn;
```

```
      END IF;
END;
$$ LANGUAGE plpgsql;


-- Функция для анализа прибыли исполнительных бригад
CREATE OR REPLACE FUNCTION analize_brigade(nstartdate DATE, nenddate DATE)
RETURNS TABLE (
    brigadename VARCHAR(50),
    sumprice NUMERIC(20,2),
    countapplication BIGINT,
    percentapplication NUMERIC(5,2),
    countagreement BIGINT,
    percentagreement NUMERIC(5,2),
    firstapplicationdate DATE,
    lastapplicationdate DATE
)
AS $$
BEGIN
    IF NOT EXISTS (SELECT 1 FROM brigade) THEN
        RAISE EXCEPTION 'Ошибка: Нет бригад';
    END IF;

    IF NOT EXISTS (SELECT 1 FROM application) THEN
        RAISE EXCEPTION 'Ошибка: Нет заявок';
    END IF;

    IF nstartdate IS NULL AND nenddate IS NULL THEN
        RETURN QUERY
        SELECT
            b.brigadename,
            SUM(CASE WHEN s.confirmation IS TRUE THEN s.priceforорder ELSE 0 END),
            COUNT(DISTINCT a.applicationid),
            ROUND(100.0 * SUM(CASE WHEN a.StatusApplicationId = (select
StatusApplicationId from StatusApplication where TypeStatus = 'Закончена') THEN 1
ELSE 0 END) / NULLIF(COUNT(a.applicationid), 0), 2),
            COUNT(DISTINCT s.surveyagreementid),
            ROUND(100.0 * SUM(CASE WHEN s.confirmation IS TRUE THEN 1 ELSE 0 END) /
NULLIF(COUNT(s.surveyagreementid), 0), 2),
            MIN(a.incomingdate),
            MAX(a.incomingdate)
        FROM brigade b
        JOIN application a USING (brigadeid)
        JOIN surveyagreement s using(applicationid)
        GROUP BY b.brigadename;
    END IF;

    IF nstartdate IS NULL AND nenddate IS NOT NULL THEN
        RETURN QUERY
        SELECT
            b.brigadename,
            SUM(CASE WHEN s.confirmation IS TRUE THEN s.priceforорder ELSE 0 END),
            COUNT(DISTINCT a.applicationid),
            ROUND(100.0 * SUM(CASE WHEN a.StatusApplicationId = (select
StatusApplicationId from StatusApplication where TypeStatus = 'Закончена') THEN 1
ELSE 0 END) / NULLIF(COUNT(a.applicationid), 0), 2),
            COUNT(DISTINCT s.surveyagreementid),
            ROUND(100.0 * SUM(CASE WHEN s.confirmation IS TRUE THEN 1 ELSE 0 END) /
NULLIF(COUNT(s.surveyagreementid), 0), 2),
            MIN(a.incomingdate),
            MAX(a.incomingdate)
        FROM brigade b
```

```
        JOIN application a USING (brigadeid)
        JOIN surveyagreement s using(applicationid)
        WHERE s.createdate <= nenddate
        GROUP BY b.brigadename;
    END IF;

    IF nstartdate IS NOT NULL AND nenddate IS NULL THEN
        RETURN QUERY
        SELECT
            b.brigadename,
            SUM(CASE WHEN s.confirmation IS TRUE THEN s.priceforорder ELSE 0 END),
            COUNT(DISTINCT a.applicationid),
            ROUND(100.0 * SUM(CASE WHEN a.StatusApplicationId = (select
StatusApplicationId from StatusApplication where TypeStatus = 'Закончена') THEN 1
ELSE 0 END) / NULLIF(COUNT(a.applicationid), 0), 2),
            COUNT(DISTINCT s.surveyagreementid),
            ROUND(100.0 * SUM(CASE WHEN s.confirmation IS TRUE THEN 1 ELSE 0 END) /
NULLIF(COUNT(s.surveyagreementid), 0), 2),
            MIN(a.incomingdate),
            MAX(a.incomingdate)
        FROM brigade b
        JOIN application a USING (brigadeid)
        JOIN surveyagreement s using(applicationid)
        WHERE s.createdate >= nstartdate
        GROUP BY b.brigadename;
    END IF;

    IF nstartdate IS NOT NULL AND nenddate IS NOT NULL THEN
        RETURN QUERY
        SELECT
            b.brigadename,
            SUM(CASE WHEN s.confirmation IS TRUE THEN s.priceforорder ELSE 0 END),
            COUNT(DISTINCT a.applicationid),
            ROUND(100.0 * SUM(CASE WHEN a.StatusApplicationId = (select
StatusApplicationId from StatusApplication where TypeStatus = 'Закончена') THEN 1
ELSE 0 END) / NULLIF(COUNT(a.applicationid), 0), 2),
            COUNT(DISTINCT s.surveyagreementid),
            ROUND(100.0 * SUM(CASE WHEN s.confirmation IS TRUE THEN 1 ELSE 0 END) /
NULLIF(COUNT(s.surveyagreementid), 0), 2),
            MIN(a.incomingdate),
            MAX(a.incomingdate)
        FROM brigade b
        JOIN application a USING (brigadeid)
        JOIN surveyagreement s using(applicationid)
        WHERE s.createdate BETWEEN nstartdate AND nenddate
        GROUP BY b.brigadename;
    END IF;
END;
$$ LANGUAGE plpgsql;
-- скрипты создания хранимых процедур
-- Процедура для добавления заказчика
create or replace procedure new_client(
    nFio VARCHAR(40),
    nPhone VARCHAR(18),
    nEmail VARCHAR(100),
    nPassword VARCHAR(100)
)
AS $$
declare
    reg_clientid integer;
    role_client_id integer;
begin
```

```
    if nFio is null then
        raise exception 'Ошибка: ФИО не может быть NULL';
    end if;

    if nPhone is null then
        raise exception 'Ошибка: Номер телефона не может быть NULL';
    end if;

    if exists (select 1 from client where phone = nPhone) then
        raise exception 'Ошибка: Заказчик с таким номером телефона уже существует';
    end if;

    if nEmail is null then
        raise exception 'Ошибка: Адрес электронной почты не может быть NULL';
    end if;

    if exists (select 1 from client where email = nEmail) then
        raise exception 'Ошибка: Пользователь с такой почтой уже существует';
    end if;

    if nPassword is null then
        raise exception 'Ошибка: Пароль не может быть NULL';
    end if;

    insert into Client(Fio, Phone, Email)
    values (nFio, nPhone, nEmail)
    returning ClientId into reg_clientid;

    insert into Users(Email, PasswordHash, RoleId, ClientId)
    values (nEmail, nPassword, 1, reg_clientid);
exception
    when others then
        raise exception 'Ошибка при добавлении заказчика: %', SQLERRM;
end;
$$ language plpgsql;


-- Процедура для добавления заявки
create or replace procedure new_application(nPhone VARCHAR (18), nOrgName VARCHAR
(30), nINN VARCHAR (12), nCity VARCHAR (20), nStreet VARCHAR (20), nNumber VARCHAR
(10), nArea Float)
as $$
declare
nOrgId int;
naddressId int;
nObjectId int;
begin
      IF nPhone IS NULL THEN
      RAISE EXCEPTION 'Ошибка: Телефон не может быть NULL';
      END IF;

      IF NOT exists (select 1 from client where phone = nPhone) THEN
      RAISE EXCEPTION 'Ошибка: Клиента с указанным номером телефона нет';
      END IF;

      IF nOrgName IS NULL THEN
      RAISE EXCEPTION 'Ошибка: Наименование организации не может быть NULL';
      END IF;

      IF nINN IS NULL THEN
      RAISE EXCEPTION 'Ошибка: ИНН организации не может быть NULL';
      END IF;
```

```
       IF ncity IS NULL THEN
       RAISE EXCEPTION 'Ошибка: Город не может быть NULL';
       END IF;

       IF nstreet IS NULL THEN
       RAISE EXCEPTION 'Ошибка: Улица не может быть NULL';
       END IF;

       IF nNumber IS NULL THEN
       RAISE EXCEPTION 'Ошибка: Номер строения не может быть NULL';
       END IF;

       IF nArea IS NULL THEN
       RAISE EXCEPTION 'Ошибка: Площадь объекта не может быть NULL';
       END IF;

       IF nArea <= 0 THEN
       RAISE EXCEPTION 'Ошибка: Площадь объекта не может быть отрицательной или
равняться нулю';
       END IF;

       if    EXISTS (select 1 from Organization where INN=nINN) then
       select OrganizationId into nOrgId
       from Organization where
       inn=nINN;
       else
       insert into organization(OrganizationName,INN) values(nOrgName,nINN)
       returning organizationid into nOrgId;
       end if;
       IF EXISTS (SELECT 1 FROM address WHERE cityname = nCity AND streetname =
nStreet AND number = nNumber) THEN
    SELECT addressid INTO naddressId
    FROM address
    WHERE cityname = nCity AND streetname = nStreet AND number = nNumber;
      ELSE
    INSERT INTO address (cityname, streetname, number) VALUES (nCity, nStreet,
nNumber)
    RETURNING addressid INTO naddressId;
      END IF;
       if EXISTS (select 1 from ObjectSurvey where AddressId = naddressId and
OrganizationId = nOrgId and ObjectArea = nArea) then
       select ObjectSurveyId into nObjectId
       from ObjectSurvey
       where AddressId = naddressId and OrganizationId = nOrgId and ObjectArea =
nArea;
       else
       insert into ObjectSurvey(Clientid,AddressId,OrganizationId,ObjectArea)
values((select ClientId from client where phone=nPhone),naddressId,nOrgId,nArea)
       RETURNING ObjectSurveyId INTO nObjectId;
       end if;
       insert into
application(ClientId,ObjectSurveyId,StatusApplicationId,StarteDate,EndDate) values
((select ClientId from client where phone=nPhone),nObjectId,(select
StatusApplicationId from StatusApplication where TypeStatus = 'Принята'),null,null);
       EXCEPTION
         WHEN OTHERS THEN
             RAISE EXCEPTION 'Ошибка при добавлении заявки: %', SQLERRM;
end;
$$ LANGUAGE plpgsql;

-- Процедура для добавления выбранной улсуги
```

```
create or replace procedure new_selectedservices(nPhone VARCHAR (18),nIncomingDate
date, nINN VARCHAR (12), nCity VARCHAR (20), nStreet VARCHAR (20), nNumber VARCHAR
(10),nServiceName VARCHAR (30), nVolume Float)
as $$
declare
nApplicationId int;
begin

	IF nPhone IS NULL THEN
	RAISE EXCEPTION 'Ошибка: Телефон не может быть NULL';
	END IF;

	IF NOT exists (select 1 from client where phone = nPhone) THEN
	RAISE EXCEPTION 'Ошибка: Клиента с указанным номером телефона нет';
	END IF;

	IF nIncomingDate IS NULL THEN
	RAISE EXCEPTION 'Ошибка: Дата не может быть NULL';
	END IF;

	IF nINN IS NULL THEN
	RAISE EXCEPTION 'Ошибка: ИНН организации не может быть NULL';
	END IF;

	IF not exists(select 1 from organization where INN = nINN) THEN
	RAISE EXCEPTION 'Ошибка: Нет организации с указанным ИНН';
	END IF;

	IF ncity IS NULL THEN
	RAISE EXCEPTION 'Ошибка: Город не может быть NULL';
	END IF;

	IF nstreet IS NULL THEN
	RAISE EXCEPTION 'Ошибка: Улица не может быть NULL';
	END IF;

	IF nNumber IS NULL THEN
	RAISE EXCEPTION 'Ошибка: Номер строения не может быть NULL';
	END IF;

	if not exists (select 1 from address where cityname = nCity AND streetname =
nStreet AND number = nNumber) then
	RAISE EXCEPTION 'Ошибка: Указанного адреса нет';
	END IF;

	IF nServiceName IS NULL THEN
	RAISE EXCEPTION 'Ошибка: Наименование услуги не может быть NULL';
	END IF;

	IF not exists(select 1 from ServiceCatalog where ServiceName = nServiceName)
THEN
	RAISE EXCEPTION 'Ошибка: Указанной услуги нет в каталоге услуг';
	END IF;

	IF nVolume IS NULL THEN
	RAISE EXCEPTION 'Ошибка: Объем выполняемой работы не может быть NULL';
	END IF;

	IF nVolume <= 0 THEN
	RAISE EXCEPTION 'Ошибка: Объем выполняемой работы не может быть отрицательной
или равняться нулю';
	END IF;
```

```
       select ApplicationId into nApplicationId
       from Application
       where ClientId = (select ClientId from Client where phone = nPhone) and
       ObjectSurveyId = (select ObjectSurveyId from ObjectSurvey where OrganizationId
= (select OrganizationId from Organization where inn = nInn) and AddressId = (select
AddressId from address where cityname = nCity AND streetname = nStreet AND number =
nNumber))
       and IncomingDate = nIncomingDate
       limit 1;

       if nApplicationId is null then
       RAISE EXCEPTION 'Ошибка: Заявки на эту дату нет';
       END IF;

       insert into SelectedServices(ServiceId,ApplicationId,Volume) values ((select
ServiceId from ServiceCatalog where
ServiceName=nServiceName),nApplicationId,nVolume);

       EXCEPTION
         WHEN OTHERS THEN
             RAISE EXCEPTION 'Ошибка при добавлении заявки: %', SQLERRM;
end;
$$ LANGUAGE plpgsql;
-- скрипты триггеров
-- скрипты триггеров бизнесс-правил
--триггер для подстчета стоимости выбранной услуги
CREATE OR REPLACE FUNCTION calculate_costservices()
RETURNS TRIGGER AS $$
declare
nPrice Numeric(10,2);
BEGIN
  SELECT price INTO nPrice
  FROM servicecatalog
  WHERE ServiceId = NEW.ServiceId;

  NEW.costservices := NEW.volume * nPrice;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_calculate_costservices
BEFORE INSERT OR UPDATE ON SelectedServices
FOR EACH ROW
EXECUTE FUNCTION calculate_costservices();

--триггер для подсчета стоимости договора
CREATE OR REPLACE FUNCTION calculate_priceforорder()
RETURNS TRIGGER AS $$
BEGIN

  SELECT SUM(costservices) INTO NEW.priceforорder
  FROM SelectedServices
  WHERE ApplicationId = NEW.applicationid;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_calculate_priceforорder
BEFORE INSERT ON SurveyAgreement
FOR EACH ROW
EXECUTE FUNCTION calculate_priceforорder();
```

```
--триггер для назначения бригады на заявку
CREATE OR REPLACE FUNCTION assign_brigade_to_application()
RETURNS TRIGGER AS $$
DECLARE
       selected_brigade_id INT;
BEGIN
       SELECT b.brigadeid INTO selected_brigade_id
       FROM brigade b
       LEFT JOIN application a using (brigadeid)
       GROUP BY b.brigadeid
       ORDER BY COUNT(a.applicationid)
       LIMIT 1;
  NEW.brigadeid := selected_brigade_id;

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_assign_brigade
BEFORE INSERT ON Application
FOR EACH ROW
EXECUTE FUNCTION assign_brigade_to_application();
 --скрипты триггеров бизнес-ограничений
-- триггер на ограничение добавления отчета

create or replace function check_application_status_before_report()
returns trigger as $$
declare
    app_status integer;
begin
    select StatusApplicationId into app_status
    from Application
    where ApplicationId = NEW.AplicationId;

    if app_status != 3 then
        raise exception 'Невозможно добавить отчет: заявка ещё не выполнена.';
    end if;

    return NEW;
end;
$$ language plpgsql;

create trigger trg_check_status_before_report
before insert on SurveyReport
for each row
execute function check_application_status_before_report();

-- триггер на ограничение добавления договора
create or replace function check_before_agreement()
returns trigger as $$
declare
    app_status integer;
    report_exists boolean;
begin
    select StatusApplicationId into app_status
    from Application
    where ApplicationId = NEW.ApplicationId;

    select exists (
        select 1 from SurveyReport
        where AplicationId = NEW.ApplicationId
```

```
    ) into report_exists;

    if app_status != 3 or not report_exists then
        raise exception 'Невозможно сформировать договор: заявка не выполнена или
отсутствует отчет.';
    end if;

    return NEW;
end;
$$ language plpgsql;

create trigger trg_check_before_agreement
before insert on SurveyAgreement
for each row
execute function check_before_agreement();

-- триггер на ограничение даты создания договора
create or replace function check_agreement_date()
returns trigger as $$
declare
    app_end_date date;
begin
    select EndDate into app_end_date
    from Application
    where ApplicationId = NEW.ApplicationId;

    if NEW.CreateDate < app_end_date then
        raise exception 'Дата договора не может быть раньше окончания обследования.';
    end if;

    return NEW;
end;
$$ language plpgsql;

create trigger trg_check_agreement_date
before insert on SurveyAgreement
for each row
execute function check_agreement_date();
--скрипт триггера ограничения целостности
create or replace function prevent_report_delete_if_confirmed()
returns trigger as $$
declare
    is_confirmed boolean;
begin
    select Confirmation into is_confirmed
    from SurveyAgreement
    where ReportId = OLD.ReportId;

    if is_confirmed is true then
        raise exception 'Отчет нельзя удалить: связан с подтвержденным договором.';
    end if;

    return OLD;
end;
$$ language plpgsql;

create trigger trg_prevent_report_delete
before delete on SurveyReport
for each row
execute function prevent_report_delete_if_confirmed();
-- скрипты триггера ограничения действия
--триггер на ограничнеие добавления отчета
```

```
create or replace function check_employee_brigade_for_report()
returns trigger as $$
declare
    app_brigade_id integer;
    emp_brigade_id integer;
begin
    select BrigadeId into app_brigade_id
    from Application
    where ApplicationId = NEW.AplicationId;

    select BrigadeId into emp_brigade_id
    from Employee
    where EmployeeId = NEW.EmployeeId;

    if app_brigade_id is distinct from emp_brigade_id then
        raise exception 'Сотрудник не состоит в бригаде, выполнявшей заявку.';
    end if;

    return NEW;
end;
$$ language plpgsql;

create trigger trg_check_employee_brigade
before insert or update on SurveyReport
for each row
execute function check_employee_brigade_for_report();

-- триггер на ограничение удаления договора
create or replace function prevent_agreement_delete_if_confirmed()
returns trigger as $$
begin
    if OLD.Confirmation then
        raise exception 'Нельзя удалить подтвержденный договор.';
    end if;
    return OLD;
end;
$$ language plpgsql;

create trigger trg_prevent_agreement_delete
before delete on SurveyAgreement
for each row
execute function prevent_agreement_delete_if_confirmed();
-- скрипт тригера DDL
--триггер на ограничение удаления таблиц
CREATE OR REPLACE FUNCTION prevent_drop_table()
RETURNS event_trigger AS
$$
BEGIN
    IF current_user <> 'postgres' THEN
        RAISE EXCEPTION 'Удаление таблиц разрешено только пользователю postgres!';
    END IF;
END;
$$
LANGUAGE plpgsql;
CREATE EVENT TRIGGER drop_table_restrict
ON ddl_command_start
WHEN TAG IN ('DROP TABLE')
EXECUTE FUNCTION prevent_drop_table();
-- скрипт триггера бд
-- таблица для логирования подключения к бд
create table userlog (
id serial primary key,
```

```
usern text,
datetime TIMESTAMP WITH TIME zone default CURRENT_TIMESTAMP
);

--триггер для логирования подключения к бд
CREATE OR REPLACE FUNCTION connectuser()
RETURNS EVENT_TRIGGER AS $$
DECLARE
username TEXT;
datetimes TIMESTAMP;
BEGIN
SELECT session_user INTO username;
SELECT CURRENT_TIMESTAMP INTO datetimes;
INSERT INTO userlog (usern, datetime)VALUES (username, datetimes);
END;
$$ LANGUAGE plpgsql;

CREATE EVENT TRIGGER log_user
ON LOGIN
EXECUTE FUNCTION connectuser();
--скрипты реализации политики безопастности
-- Роли соответствуют AccessLevelName
CREATE ROLE "Admin" LOGIN PASSWORD 'admin_pass';
CREATE ROLE "Employee" LOGIN PASSWORD 'employee_pass';
CREATE ROLE "Client" LOGIN PASSWORD 'client_pass';

-- Права администратора
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO "Admin";
GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA public TO "Admin";
GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA public TO "Admin";
GRANT EXECUTE ON ALL PROCEDURES IN SCHEMA public TO "Admin";

ALTER ROLE "Admin" WITH CREATEROLE;

-- Права сотрудника
GRANT SELECT, INSERT, UPDATE ON Application TO "Employee";
GRANT SELECT, INSERT, UPDATE ON SurveyReport TO "Employee";
GRANT SELECT, INSERT, UPDATE ON SurveyAgreement TO "Employee";

GRANT SELECT ON
    Client,
    SelectedServices,
    ServiceCatalog,
    Organization,
    Address,
    ObjectSurvey,
    Brigade
TO "Employee";

GRANT EXECUTE ON FUNCTION analize_brigade(date, date) TO "Employee";
GRANT EXECUTE ON FUNCTION analize_organization(date, date) TO "Employee";

-- Права клиента
GRANT SELECT, INSERT, UPDATE ON Organization TO "Client";
GRANT SELECT, INSERT, UPDATE ON Address TO "Client";

GRANT INSERT ON Application TO "Client";
GRANT INSERT ON SelectedServices TO "Client";

GRANT SELECT ON
    ServiceCatalog,
    Application,
```

```
    SurveyAgreement,
    SurveyReport
TO "Client";

GRANT EXECUTE ON FUNCTION analize_client_application(varchar) TO "Client";
GRANT EXECUTE ON PROCEDURE new_client(varchar, varchar,varchar,varchar) TO "Client";
GRANT EXECUTE ON PROCEDURE new_application(varchar, varchar, varchar, varchar,
varchar, varchar, float) TO "Client";
GRANT EXECUTE ON PROCEDURE new_selectedservices(varchar, date, varchar, varchar,
varchar, varchar,varchar, float) TO "Client";
```

Приложение Б
Реализация серверной части информационной системы на языке C#

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Рязанский государственный радиотехнический университет имени В.Ф. Уткина»

(ФГБОУ ВО «РГРТУ», РГРТУ)

Кафедра Автоматизированных Систем Управления

«Разработка информационной системы центра исследований строительных конструкций»

Реализация серверной части информационной системы на языке C#

Листов 62

Рязань 2025

```
//appseting.json
{
  "ConnectionStrings": {
    "DefaultConnection":
"Host=localhost;Port=5432;Database=Center;Username=postgres;Password=2111"
  },
  "Jwt": {
    "Key": "Очень_секретный_ключ",
    "Issuer": "YourApp",
    "Audience": "YourAppUsers",
    "ExpiresMinutes": 60
  }
}
//program.cs
using Microsoft.EntityFrameworkCore;
using WebApplication1;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.IdentityModel.Tokens;
using System.Text;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddScoped<AnalysisService>();
builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new() { Title = "WebApplication1", Version = "v1" });

    c.AddSecurityDefinition("Bearer", new
Microsoft.OpenApi.Models.OpenApiSecurityScheme
    {
        Name = "Authorization",
        Type = Microsoft.OpenApi.Models.SecuritySchemeType.ApiKey,
        Scheme = "Bearer",
        BearerFormat = "JWT",
        In = Microsoft.OpenApi.Models.ParameterLocation.Header,
        Description = "Введите токен в формате: Bearer {твой_токен}"
    });

    c.AddSecurityRequirement(new Microsoft.OpenApi.Models.OpenApiSecurityRequirement
    {
        {
            new Microsoft.OpenApi.Models.OpenApiSecurityScheme
            {
                Reference = new Microsoft.OpenApi.Models.OpenApiReference
                {
                    Type = Microsoft.OpenApi.Models.ReferenceType.SecurityScheme,
                    Id = "Bearer"
                },
                Scheme = "Bearer",
                Name = "Bearer",
                In = Microsoft.OpenApi.Models.ParameterLocation.Header
            },
            new List<string>()
        }
    });
});

builder.Services.AddDbContext<CenterContext>(options =>
```

```
options.UseNpgsql(builder.Configuration.GetConnectionString("DefaultConnection")));

var key = Encoding.UTF8.GetBytes(builder.Configuration["Jwt:Key"]);

builder.Services.AddAuthentication(options =>
{
    options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
    options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
})
.AddJwtBearer(options =>
{
    options.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuer = false,
        ValidateAudience = false,
        ValidateLifetime = true,
        ValidateIssuerSigningKey = true,
        IssuerSigningKey = new SymmetricSecurityKey(key)
    };
});

builder.Services.AddAuthorization();

var app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseMiddleware<ExceptionHandlingMiddleware>();

app.UseHttpsRedirection();

app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

app.Run();

//JwtService.cs
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;
using Microsoft.Extensions.Configuration;
using Microsoft.IdentityModel.Tokens;
using WebApplication1.Models;

public class JwtService
{
    private readonly IConfiguration _configuration;

    public JwtService(IConfiguration configuration)
    {
        _configuration = configuration;
    }

    public string GenerateToken(Users user)
```

```
    {
        var claims = new List<Claim>
    {
        new Claim(ClaimTypes.NameIdentifier, user.Userid.ToString()),
        new Claim(ClaimTypes.Name, user.Email),
        new Claim(ClaimTypes.Role, user.Roles.Rolename)
    };

        if (user.Clientid.HasValue)
            claims.Add(new Claim("ClientId", user.Clientid.Value.ToString()));

        if (user.Employeeid.HasValue)
            claims.Add(new Claim("EmployeeId", user.Employeeid.Value.ToString()));

        var key = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_configuration["Jwt:Key"]));
        var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);
        var expires =
DateTime.Now.AddMinutes(Convert.ToDouble(_configuration["Jwt:ExpiresMinutes"]));

        var token = new JwtSecurityToken(
            issuer: _configuration["Jwt:Issuer"],
            audience: _configuration["Jwt:Audience"],
            claims: claims,
            expires: expires,
            signingCredentials: creds
        );

        return new JwtSecurityTokenHandler().WriteToken(token);
    }

}
//ExceptionHandlingMiddleware.cs
using Microsoft.EntityFrameworkCore;
using Npgsql;
using System.Text.Json;

namespace WebApplication1
{
    public class ExceptionHandlingMiddleware
    {
        private readonly RequestDelegate _next;
        private readonly ILogger<ExceptionHandlingMiddleware> _logger;

        public ExceptionHandlingMiddleware(RequestDelegate next,
ILogger<ExceptionHandlingMiddleware> logger)
        {
            _next = next;
            _logger = logger;
        }

        public async Task Invoke(HttpContext context)
        {
            try
            {
                await _next(context);
            }
            catch (DbUpdateException ex) when (ex.InnerException is PostgresException
pgEx)
            {
                _logger.LogError(ex, "Database update exception");
```

```
                context.Response.StatusCode = 400;
                context.Response.ContentType = "application/json";

                await context.Response.WriteAsync(JsonSerializer.Serialize(new
                {
                    error = pgEx.MessageText // Только текст из RAISE EXCEPTION
                }));
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Unhandled exception");

                context.Response.StatusCode = 500;
                context.Response.ContentType = "application/json";

                await context.Response.WriteAsync(JsonSerializer.Serialize(new
                {
                    error = "Внутренняя ошибка сервера"
                }));
            }
        }
    }

}
//CenterContext.cs
using System;
using System.Collections.Generic;
using Microsoft.EntityFrameworkCore;
using WebApplication1.DTOS;
using WebApplication1.Models;

namespace WebApplication1;

public partial class CenterContext : DbContext
{
    public CenterContext()
    {
    }

    public CenterContext(DbContextOptions<CenterContext> options)
        : base(options)
    {
    }

    public virtual DbSet<Address> Addresses { get; set; }

    public virtual DbSet<Application> Applications { get; set; }

    public virtual DbSet<Brigade> Brigades { get; set; }

    public virtual DbSet<Client> Clients { get; set; }

    public virtual DbSet<Employee> Employees { get; set; }

    public virtual DbSet<Jobtitle> Jobtitles { get; set; }

    public virtual DbSet<Objectsurvey> Objectsurveys { get; set; }

    public virtual DbSet<Organization> Organizations { get; set; }

    public virtual DbSet<Selectedservice> Selectedservices { get; set; }
```

```
    public virtual DbSet<Servicecatalog> Servicecatalogs { get; set; }

    public virtual DbSet<Statusapplication> Statusapplications { get; set; }

    public virtual DbSet<Surveyagreement> Surveyagreements { get; set; }

    public virtual DbSet<Surveyreport> Surveyreports { get; set; }

    public virtual DbSet<Users> Userses { get; set; }

    public virtual DbSet<Roles> Roleses { get; set; }

    public virtual DbSet<ClientApplicationDto> ClientApplicationDtos { get; set; }

    public virtual DbSet<OrganizationAnalysisDto> OrganizationAnalysisDtos { get;
set; }

    public virtual DbSet<BrigadeAnalysisDto> BrigadeAnalysisDtos { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
#warning To protect potentially sensitive information in your connection string, you
should move it out of source code. You can avoid scaffolding the connection string by
using the Name= syntax to read it from configuration - see
https://go.microsoft.com/fwlink/?linkid=2131148. For more guidance on storing
connection strings, see https://go.microsoft.com/fwlink/?LinkId=723263.
        =>
optionsBuilder.UseNpgsql("Host=localhost;Port=5432;Database=Center;Username=postgres;
Password=2111");

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.UseCollation("en_US.UTF-8");

        modelBuilder.Entity<Address>(entity =>
        {
            entity.HasKey(e => e.Addressid).HasName("address_pkey");

            entity.ToTable("address");

            entity.Property(e => e.Addressid).HasColumnName("addressid");
            entity.Property(e => e.Cityname)
                .HasMaxLength(20)
                .HasColumnName("cityname");
            entity.Property(e => e.Number)
                .HasMaxLength(10)
                .HasColumnName("number");
            entity.Property(e => e.Streetname)
                .HasMaxLength(40)
                .HasColumnName("streetname");
        });

        modelBuilder.Entity<Application>(entity =>
        {
            entity.HasKey(e => e.Applicationid).HasName("application_pkey");

            entity.ToTable("application");

            entity.Property(e => e.Applicationid).HasColumnName("applicationid");
            entity.Property(e => e.Brigadeid).HasColumnName("brigadeid");
            entity.Property(e => e.Clientid).HasColumnName("clientid");
            entity.Property(e => e.Enddate).HasColumnName("enddate");
            entity.Property(e => e.Incomingdate)
```

```
                .HasDefaultValueSql("CURRENT_DATE")
                .HasColumnName("incomingdate");
            entity.Property(e => e.Objectsurveyid).HasColumnName("objectsurveyid");
            entity.Property(e => e.Startedate).HasColumnName("startedate");
            entity.Property(e =>
e.Statusapplicationid).HasColumnName("statusapplicationid");

            entity.HasOne(d => d.Brigade).WithMany(p => p.Applications)
                .HasForeignKey(d => d.Brigadeid)
                .OnDelete(DeleteBehavior.SetNull)
                .HasConstraintName("application_brigadeid_fkey");

            entity.HasOne(d => d.Client).WithMany(p => p.Applications)
                .HasForeignKey(d => d.Clientid)
                .OnDelete(DeleteBehavior.SetNull)
                .HasConstraintName("application_clientid_fkey");

            entity.HasOne(d => d.Objectsurvey).WithMany(p => p.Applications)
                .HasForeignKey(d => d.Objectsurveyid)
                .OnDelete(DeleteBehavior.SetNull)
                .HasConstraintName("application_objectsurveyid_fkey");

            entity.HasOne(d => d.Statusapplication).WithMany(p => p.Applications)
                .HasForeignKey(d => d.Statusapplicationid)
                .OnDelete(DeleteBehavior.SetNull)
                .HasConstraintName("application_statusapplicationid_fkey");
        });

        modelBuilder.Entity<Brigade>(entity =>
        {
            entity.HasKey(e => e.Brigadeid).HasName("brigade_pkey");

            entity.ToTable("brigade");

            entity.Property(e => e.Brigadeid).HasColumnName("brigadeid");
            entity.Property(e => e.Brigadename)
                .HasMaxLength(20)
                .HasColumnName("brigadename");
        });

        modelBuilder.Entity<Client>(entity =>
        {
            entity.HasKey(e => e.Clientid).HasName("client_pkey");

            entity.ToTable("client");

            entity.HasIndex(e => e.Phone, "client_phone_key").IsUnique();

            entity.Property(e => e.Clientid).HasColumnName("clientid");

            entity.Property(e => e.Fio)
                .HasMaxLength(40)
                .HasColumnName("fio");
            entity.Property(e => e.Phone)
                .HasMaxLength(18)
                .HasColumnName("phone");
        });

        modelBuilder.Entity<Employee>(entity =>
        {
            entity.HasKey(e => e.Employeeid).HasName("employee_pkey");
```

```
    entity.ToTable("employee");

    entity.HasIndex(e => e.Phone, "employee_phone_key").IsUnique();

    entity.Property(e => e.Employeeid).HasColumnName("employeeid");
    entity.Property(e => e.Brigadeid).HasColumnName("brigadeid");

    entity.Property(e => e.Fio)
        .HasMaxLength(40)
        .HasColumnName("fio");
    entity.Property(e => e.Jobtitleid).HasColumnName("jobtitleid");
    entity.Property(e => e.Phone)
        .HasMaxLength(18)
        .HasColumnName("phone");

    entity.HasOne(d => d.Brigade).WithMany(p => p.Employees)
        .HasForeignKey(d => d.Brigadeid)
        .OnDelete(DeleteBehavior.SetNull)
        .HasConstraintName("employee_brigadeid_fkey");

    entity.HasOne(d => d.Jobtitle).WithMany(p => p.Employees)
        .HasForeignKey(d => d.Jobtitleid)
        .OnDelete(DeleteBehavior.SetNull)
        .HasConstraintName("employee_jobtitleid_fkey");
});

modelBuilder.Entity<Jobtitle>(entity =>
{
    entity.HasKey(e => e.Jobtitleid).HasName("jobtitle_pkey");

    entity.ToTable("jobtitle");

    entity.Property(e => e.Jobtitleid).HasColumnName("jobtitleid");
    entity.Property(e => e.Jobtitlename)
        .HasMaxLength(30)
        .HasColumnName("jobtitlename");
});

modelBuilder.Entity<Objectsurvey>(entity =>
{
    entity.HasKey(e => e.Objectsurveyid).HasName("objectsurvey_pkey");

    entity.ToTable("objectsurvey");

    entity.Property(e => e.Objectsurveyid).HasColumnName("objectsurveyid");
    entity.Property(e => e.Clientid).HasColumnName("clientid");
    entity.Property(e => e.Addressid).HasColumnName("addressid");
    entity.Property(e => e.Objectarea).HasColumnName("objectarea");
    entity.Property(e => e.Organizationid).HasColumnName("organizationid");

    entity.HasOne(d => d.Address).WithMany(p => p.Objectsurveys)
        .HasForeignKey(d => d.Addressid)
        .OnDelete(DeleteBehavior.SetNull)
        .HasConstraintName("objectsurvey_addressid_fkey");

    entity.HasOne(d => d.Organization).WithMany(p => p.Objectsurveys)
        .HasForeignKey(d => d.Organizationid)
        .OnDelete(DeleteBehavior.SetNull)
        .HasConstraintName("objectsurvey_organizationid_fkey");

    entity.HasOne(d => d.Client).WithMany(p => p.Objectsurveys)
        .HasForeignKey(d => d.Clientid)
```

```
                .OnDelete(DeleteBehavior.SetNull)
                .HasConstraintName("objectsurvey_clientid_fkey");
        });


        modelBuilder.Entity<Organization>(entity =>
        {
            entity.HasKey(e => e.Organizationid).HasName("organization_pkey");

            entity.ToTable("organization");

            entity.HasIndex(e => e.Inn, "organization_inn_key").IsUnique();

            entity.Property(e => e.Organizationid).HasColumnName("organizationid");
            entity.Property(e => e.Inn)
                .HasMaxLength(12)
                .HasColumnName("inn");
            entity.Property(e => e.Organizationname)
                .HasMaxLength(30)
                .HasColumnName("organizationname");
        });

        modelBuilder.Entity<Selectedservice>(entity =>
        {
            entity.HasKey(e =>
e.Selectedservicesid).HasName("selectedservices_pkey");

            entity.ToTable("selectedservices");

            entity.Property(e =>
e.Selectedservicesid).HasColumnName("selectedservicesid");
            entity.Property(e => e.Applicationid).HasColumnName("applicationid");
            entity.Property(e => e.Costservices)
                .HasPrecision(15, 2)
                .HasColumnName("costservices");
            entity.Property(e => e.Serviceid).HasColumnName("serviceid");
            entity.Property(e => e.Volume).HasColumnName("volume");

            entity.HasOne(d => d.Application).WithMany(p => p.Selectedservices)
                .HasForeignKey(d => d.Applicationid)
                .OnDelete(DeleteBehavior.Cascade)
                .HasConstraintName("selectedservices_applicationid_fkey");

            entity.HasOne(d => d.Service).WithMany(p => p.Selectedservices)
                .HasForeignKey(d => d.Serviceid)
                .OnDelete(DeleteBehavior.SetNull)
                .HasConstraintName("selectedservices_serviceid_fkey");
        });

        modelBuilder.Entity<Servicecatalog>(entity =>
        {
            entity.HasKey(e => e.Serviceid).HasName("servicecatalog_pkey");

            entity.ToTable("servicecatalog");

            entity.Property(e => e.Serviceid).HasColumnName("serviceid");
            entity.Property(e => e.Description).HasColumnName("description");
            entity.Property(e => e.Measurement)
                .HasMaxLength(30)
                .HasColumnName("measurement");
            entity.Property(e => e.Price)
                .HasPrecision(10, 2)
```

```
                .HasColumnName("price");
            entity.Property(e => e.Servicename)
                .HasMaxLength(100)
                .HasColumnName("servicename");
        });

        modelBuilder.Entity<Statusapplication>(entity =>
        {
            entity.HasKey(e =>
e.Statusapplicationid).HasName("statusapplication_pkey");

            entity.ToTable("statusapplication");

            entity.Property(e =>
e.Statusapplicationid).HasColumnName("statusapplicationid");
            entity.Property(e => e.Typestatus)
                .HasMaxLength(15)
                .HasColumnName("typestatus");
        });

        modelBuilder.Entity<Surveyagreement>(entity =>
        {
            entity.HasKey(e => e.Surveyagreementid).HasName("surveyagreement_pkey");

            entity.ToTable("surveyagreement");

            entity.Property(e =>
e.Surveyagreementid).HasColumnName("surveyagreementid");
            entity.Property(e => e.Applicationid).HasColumnName("applicationid");
            entity.Property(e => e.Confirmation)
                .HasDefaultValue(false)
                .HasColumnName("confirmation");
            entity.Property(e => e.Createdate)
                .HasDefaultValueSql("CURRENT_DATE")
                .HasColumnName("createdate");
            entity.Property(e => e.Employeeid).HasColumnName("employeeid");
            entity.Property(e => e.Pricefororder)
                .HasPrecision(20, 2)
                .HasColumnName("pricefororder");
            entity.Property(e => e.Reportid).HasColumnName("reportid");

            entity.HasOne(d => d.Application).WithMany(p => p.Surveyagreements)
                .HasForeignKey(d => d.Applicationid)
                .OnDelete(DeleteBehavior.Restrict)
                .HasConstraintName("surveyagreement_applicationid_fkey");

            entity.HasOne(d => d.Employee).WithMany(p => p.Surveyagreements)
                .HasForeignKey(d => d.Employeeid)
                .OnDelete(DeleteBehavior.SetNull)
                .HasConstraintName("surveyagreement_employeeid_fkey");

            entity.HasOne(d => d.Report).WithMany(p => p.Surveyagreements)
                .HasForeignKey(d => d.Reportid)
                .OnDelete(DeleteBehavior.SetNull)
                .HasConstraintName("surveyagreement_reportid_fkey");
        });

        modelBuilder.Entity<Surveyreport>(entity =>
        {
            entity.HasKey(e => e.Reportid).HasName("surveyreport_pkey");

            entity.ToTable("surveyreport");
```

```
            entity.Property(e => e.Reportid).HasColumnName("reportid");
            entity.Property(e => e.Aplicationid).HasColumnName("aplicationid");
            entity.Property(e => e.Employeeid).HasColumnName("employeeid");
            entity.Property(e => e.Filereport).HasColumnName("filereport");

            entity.HasOne(d => d.Application).WithMany(p => p.Surveyreports)
                .HasForeignKey(d => d.Aplicationid)
                .OnDelete(DeleteBehavior.Restrict)
                .HasConstraintName("surveyreport_aplicationid_fkey");

            entity.HasOne(d => d.Employee).WithMany(p => p.Surveyreports)
                .HasForeignKey(d => d.Employeeid)
                .OnDelete(DeleteBehavior.SetNull)
                .HasConstraintName("surveyreport_employeeid_fkey");
        });

        modelBuilder.Entity<Users>(entity =>
        {
            entity.HasKey(e => e.Userid).HasName("users_pkey");

            entity.ToTable("users");

            entity.HasIndex(e => e.Email, "users_email_key").IsUnique();

            entity.Property(e => e.Userid).HasColumnName("userid");
            entity.Property(e => e.Email)
                .HasMaxLength(100)
                .HasColumnName("email");
            entity.Property(e => e.Passwordhash)
                .HasMaxLength(255)
                .HasColumnName("passwordhash");
            entity.Property(e => e.Roleid).HasColumnName("roleid");
            entity.Property(e => e.Clientid).HasColumnName("clientid");
            entity.Property(e => e.Employeeid).HasColumnName("employeeid");

            entity.HasOne(d => d.Roles).WithMany(p => p.Userses)
                .HasForeignKey(d => d.Roleid)
                .OnDelete(DeleteBehavior.SetNull)
                .HasConstraintName("users_roleid_fkey");

            entity.HasOne(d => d.Client).WithMany(p => p.Userses)
                .HasForeignKey(d => d.Clientid)
                .OnDelete(DeleteBehavior.SetNull)
                .HasConstraintName("users_clientid_fkey");

            entity.HasOne(d => d.Employee).WithMany(p => p.Userses)
                .HasForeignKey(d => d.Employeeid)
                .OnDelete(DeleteBehavior.SetNull)
                .HasConstraintName("users_employeeid_fkey");
        });

        modelBuilder.Entity<ClientApplicationDto>().HasNoKey();
        modelBuilder.Entity<OrganizationAnalysisDto>().HasNoKey();
        modelBuilder.Entity<BrigadeAnalysisDto>().HasNoKey();

        OnModelCreatingPartial(modelBuilder);
    }

    partial void OnModelCreatingPartial(ModelBuilder modelBuilder);
}
// AnalysisServices.cs
```

```
using Microsoft.EntityFrameworkCore;
using System;
using WebApplication1.DTOS;

namespace WebApplication1
{
    public class AnalysisService
    {
        private readonly CenterContext _context;

        public AnalysisService(CenterContext context)
        {
            _context = context;
        }

        public async Task<List<ClientApplicationDto>>
AnalyzeClientApplications(string phone)
        {
            return await _context.ClientApplicationDtos
                .FromSqlRaw("SELECT * FROM analize_client_application({0})", phone)
                .ToListAsync();
        }

        public async Task<List<OrganizationAnalysisDto>>
AnalyzeOrganizations(DateOnly? start, DateOnly? end)
        {
            return await _context.OrganizationAnalysisDtos
                .FromSqlRaw("SELECT * FROM analize_organization({0}, {1})", start,
end)
                .ToListAsync();
        }

        public async Task<List<BrigadeAnalysisDto>> AnalyzeBrigades(DateOnly? start,
DateOnly? end)
        {
            return await _context.BrigadeAnalysisDtos
                .FromSqlRaw("SELECT * FROM analize_brigade({0}, {1})", start, end)
                .ToListAsync();
        }
    }

}
// Модели
// Address
using System;
using System.Collections.Generic;

namespace WebApplication1.Models;

public partial class Address
{
    public int Addressid { get; set; }

    public string Cityname { get; set; } = null!;

    public string Streetname { get; set; } = null!;

    public string Number { get; set; } = null!;

    public virtual ICollection<Objectsurvey> Objectsurveys { get; set; } = new
List<Objectsurvey>();
}
```

```
//Application
using System;
using System.Collections.Generic;

namespace WebApplication1.Models;

public partial class Application
{
    public int Applicationid { get; set; }

    public int Clientid { get; set; }

    public int Objectsurveyid { get; set; }

    public int? Brigadeid { get; set; }

    public DateOnly Incomingdate { get; set; }

    public int Statusapplicationid { get; set; }

    public DateOnly? Startedate { get; set; }

    public DateOnly? Enddate { get; set; }

    public virtual Brigade? Brigade { get; set; }

    public virtual Client Client { get; set; } = null!;

    public virtual Objectsurvey Objectsurvey { get; set; } = null!;

    public virtual ICollection<Selectedservice> Selectedservices { get; set; } = new
List<Selectedservice>();

    public virtual Statusapplication Statusapplication { get; set; } = null!;

    public virtual ICollection<Surveyagreement> Surveyagreements { get; set; } = new
List<Surveyagreement>();

    public virtual ICollection<Surveyreport> Surveyreports { get; set; } = new
List<Surveyreport>();
}
// Brigade
using System;
using System.Collections.Generic;

namespace WebApplication1.Models;

public partial class Brigade
{
    public int Brigadeid { get; set; }

    public string Brigadename { get; set; } = null!;

    public virtual ICollection<Application> Applications { get; set; } = new
List<Application>();

    public virtual ICollection<Employee> Employees { get; set; } = new
List<Employee>();
}
//ClientApplicationDto
namespace WebApplication1.DTOS
{
```

```
    public class ClientApplicationDto
    {
        public string Cityname { get; set; }
        public string Streetname { get; set; }
        public string Number { get; set; }
        public float Objectarea { get; set; }
        public string Organizationname { get; set; }
        public string Inn { get; set; }
        public string Brigadename { get; set; }
        public DateTime Incomingdate { get; set; }
        public string Status { get; set; }
        public DateTime? Startedate { get; set; }
        public DateTime? Enddate { get; set; }
        public DateTime? Createdate { get; set; }
    }

}
//Client
using System;
using System.Collections.Generic;

namespace WebApplication1.Models;

public partial class Client
{
    public int Clientid { get; set; }

    public string Fio { get; set; } = null!;

    public string Phone { get; set; } = null!;

    public virtual ICollection<Objectsurvey> Objectsurveys { get; set; } = new
List<Objectsurvey>();
    public virtual ICollection<Application> Applications { get; set; } = new
List<Application>();
    public virtual ICollection<Users> Userses { get; set; } = new List<Users>();
}
//Employee
using System;
using System.Collections.Generic;

namespace WebApplication1.Models;

public partial class Employee
{
    public int Employeeid { get; set; }

    public string Fio { get; set; } = null!;

    public string Phone { get; set; } = null!;

    public int? Jobtitleid { get; set; }

    public int? Brigadeid { get; set; }

    public virtual Brigade? Brigade { get; set; }

    public virtual Jobtitle? Jobtitle { get; set; }

    public virtual ICollection<Surveyagreement> Surveyagreements { get; set; } = new
List<Surveyagreement>();
```

```
    public virtual ICollection<Surveyreport> Surveyreports { get; set; } = new
List<Surveyreport>();

    public virtual ICollection<Users> Userses { get; set; } = new List<Users>();
}
//Jobtitle
using System;
using System.Collections.Generic;

namespace WebApplication1.Models;

public partial class Jobtitle
{
    public int Jobtitleid { get; set; }

    public string Jobtitlename { get; set; } = null!;

    public virtual ICollection<Employee> Employees { get; set; } = new
List<Employee>();
}
//ObjectSurvey
using System;
using System.Collections.Generic;

namespace WebApplication1.Models;

public partial class Objectsurvey
{
    public int Objectsurveyid { get; set; }

    public int? Clientid { get; set; }

    public int? Addressid { get; set; }

    public int? Organizationid { get; set; }

    public double Objectarea { get; set; }

    public virtual Client? Client { get; set; }

    public virtual Address? Address { get; set; }

    public virtual ICollection<Application> Applications { get; set; } = new
List<Application>();

    public virtual Organization? Organization { get; set; }
}
//Organization
using System;
using System.Collections.Generic;

namespace WebApplication1.Models;

public partial class Organization
{
    public int Organizationid { get; set; }

    public string Organizationname { get; set; } = null!;

    public string Inn { get; set; } = null!;
```

```
    public virtual ICollection<Objectsurvey> Objectsurveys { get; set; } = new
List<Objectsurvey>();
}
//Roles
namespace WebApplication1.Models
{
    public class Roles
    {
        public int Roleid { get; set; }

        public string Rolename { get; set; } = null!;

        public virtual ICollection<Users> Userses { get; set; } = new List<Users>();
    }
}
//SelectedService
using System;
using System.Collections.Generic;

namespace WebApplication1.Models;

public partial class Selectedservice
{
    public int Selectedservicesid { get; set; }

    public int? Serviceid { get; set; }

    public int? Applicationid { get; set; }

    public float Volume { get; set; }

    public decimal? Costservices { get; set; }

    public virtual Application? Application { get; set; }

    public virtual Servicecatalog? Service { get; set; }
}
//ServiceCatalog
using System;
using System.Collections.Generic;

namespace WebApplication1.Models;

public partial class Servicecatalog
{
    public int Serviceid { get; set; }

    public string Servicename { get; set; } = null!;

    public decimal Price { get; set; }

    public string Measurement { get; set; } = null!;

    public string Description { get; set; } = null!;

    public virtual ICollection<Selectedservice> Selectedservices { get; set; } = new
List<Selectedservice>();
}
//StatusApplication
using System;
using System.Collections.Generic;
```

```
namespace WebApplication1.Models;

public partial class Statusapplication
{
    public int Statusapplicationid { get; set; }

    public string Typestatus { get; set; } = null!;

    public virtual ICollection<Application> Applications { get; set; } = new
List<Application>();
}
//SurveyAgreement
using System;
using System.Collections.Generic;

namespace WebApplication1.Models;

public partial class Surveyagreement
{
    public int Surveyagreementid { get; set; }

    public int? Applicationid { get; set; }

    public int? Reportid { get; set; }

    public int? Employeeid { get; set; }

    public DateOnly Createdate { get; set; }

    public bool Confirmation { get; set; }

    public decimal? Pricefororder { get; set; }

    public virtual Application? Application { get; set; }

    public virtual Employee? Employee { get; set; }

    public virtual Surveyreport? Report { get; set; }
}
//Surveyreport
using System;
using System.Collections.Generic;

namespace WebApplication1.Models;

public partial class Surveyreport
{
    public int Reportid { get; set; }

    public int? Aplicationid { get; set; }

    public int? Employeeid { get; set; }

    public string Filereport { get; set; } = null!;

    public virtual Application? Application { get; set; }

    public virtual Employee? Employee { get; set; }

    public virtual ICollection<Surveyagreement> Surveyagreements { get; set; } = new
List<Surveyagreement>();
}
```

```
//Users
namespace WebApplication1.Models
{
    public class Users
    {
        public int Userid { get; set; }

        public string Email { get; set; } = null!;

        public string Passwordhash { get; set; } = null!;

        public int Roleid { get; set; }

        public int? Clientid { get; set; }

        public int? Employeeid { get; set; }

        public virtual Roles? Roles { get; set; }

        public virtual Client? Client { get; set; }

        public virtual Employee? Employee { get; set; }
    }
}
//DTOS
//AddressDto
namespace WebApplication1.DTOS
{

    public class AddressCreateDto
    {
        public string CityName { get; set; }
        public string StreetName { get; set; }
        public string Number { get; set; }
    }

    public class AddressDto : AddressCreateDto
    {
        public int AddressId { get; set; }

    }
}
//AplicationDto
namespace WebApplication1.DTOS
{
    public class ApplicationCreateForClientDto
    {
        public int ObjectSurveyId { get; set; }
    }

    public class ApplicationCreateDto : ApplicationCreateForClientDto
    {
        public int ClientId { get; set; }
        public int? BrigadeId { get; set; }
        public DateOnly IncomingDate { get; set; }
        public int StatusApplicationId { get; set; }
        public DateOnly? StarteDate { get; set; }
        public DateOnly? EndDate { get; set; }
    }
    public class ApplicationDto : ApplicationCreateDto
    {
        public int ApplicationId { get; set; }
```

```
    }
}
//BrigadeAnalysisDto
namespace WebApplication1.DTOS
{
    public class BrigadeAnalysisDto
    {
        public string Brigadename { get; set; }
        public decimal Sumprice { get; set; }
        public long Countapplication { get; set; }
        public decimal Percentapplication { get; set; }
        public long Countagreement { get; set; }
        public decimal Percentagreement { get; set; }
        public DateOnly? Firstapplicationdate { get; set; }
        public DateOnly? Lastapplicationdate { get; set; }
    }

}
//BrigadeDto
namespace WebApplication1.DTOS
{
    public class BrigadeCreateDto
    {
        public string BrigadeName { get; set; }
    }

    public class BrigadeDto: BrigadeCreateDto
    {
        public int BrigadeId { get; set; }
    }
}
//ClientApplication
namespace WebApplication1.DTOS
{
    public class ClientCreateDto
    {
        public string Fio { get; set; }
        public string Phone { get; set; }
    }

    public class ClientDto : ClientCreateDto
    {
        public int ClientId { get; set; }
    }

}
//EmployeeDto
namespace WebApplication1.DTOS
{
    public class EmployeeCreateDto
    {
        public string Fio { get; set; }
        public string Phone { get; set; }
        public int? JobTitleId { get; set; }
        public int? BrigadeId { get; set; }
    }

    public class EmployeeDto : EmployeeCreateDto
    {
        public int Employeeid { get; set; }
    }
```

```
}
//JobTitleDto
namespace WebApplication1.DTOS
{
    public class JobTitleCreateDto
    {
        public string JobTitleName { get; set; }

    }
    public class JobTitleDto : JobTitleCreateDto
    {
        public int JobTitleId { get; set; }

    }
}
//LoginDto
namespace WebApplication1.DTOS
{
    public class LoginDto
    {
        public string Email { get; set; }
        public string Password { get; set; }
    }

}
//ObjectSurveyDto
namespace WebApplication1.DTOS
{
    public class ObjectSurveyCreateClientDto
    {
        public float ObjectArea { get; set; }
        public AddressDto Address { get; set; }
        public OrganizationDto Organization { get; set; }
    }

    public class ObjectSurveyCreateDto
    {
        public int? ClientId { get; set; }
        public int? AddressId { get; set; }
        public int? OrganizationId { get; set; }
        public double ObjectArea { get; set; }
    }

    public class ObjectSurveyDto : ObjectSurveyCreateDto
    {
        public int ObjectSurveyId { get; set; }

    }
}
//OrganizationAnalysisDto
namespace WebApplication1.DTOS
{
    public class OrganizationAnalysisDto
    {
        public string Organizationname { get; set; }
        public string Inn { get; set; }
        public decimal Sumprice { get; set; }
        public long Countapplication { get; set; }
        public decimal Percentapplication { get; set; }
        public long Countagreement { get; set; }
        public decimal Percentagreement { get; set; }
        public DateOnly? Firstapplicationdate { get; set; }
```

```
        public DateOnly? Lastapplicationdate { get; set; }
    }

}
//OrganizationDto
namespace WebApplication1.DTOS
{

    public class OrganizationCreateDto
    {
        public string OrganizationName { get; set; }
        public string Inn { get; set; }
    }

    public class OrganizationDto : OrganizationCreateDto
    {
        public int OrganizationId { get; set; }

    }
}
//RegisterClientDto
namespace WebApplication1.DTOS
{
    public class RegisterClientDto
    {
        public string Email { get; set; }
        public string Password { get; set; }

        public string Fio { get; set; }
        public string Phone { get; set; }
    }

}
//RegisterEmployeeDto
namespace WebApplication1.DTOS
{
    public class RegisterEmployeeDto
    {
        public string Email { get; set; }
        public string Password { get; set; }

        public string Fio { get; set; }
        public string Phone { get; set; }
        public int JobTitleId { get; set; }
        public int? BrigadeId { get; set; }
    }

}
//RolesDto
namespace WebApplication1.DTOS
{
    public class RolesCreateDTO
    {
        public string RoleName { get; set; }
    }

    public class RolesDTO : RolesCreateDTO
    {
        public int RoleId { get; set; }
    }
}
//SelectedServicesDto
```

```
namespace WebApplication1.DTOS
{
    public class SelectedServicesCreateDto
    {
        public int? ServiceId { get; set; }
        public int? ApplicationId { get; set; }
        public float Volume { get; set; }
        public decimal? CostServices { get; set; }
    }
    public class SelectedServicesDto : SelectedServicesCreateDto
    {
        public int SelectedServicesId { get; set; }

    }
}
//ServiceCatalogDto
namespace WebApplication1.DTOS
{
    public class ServiceCatalogCreateDto
    {
        public string ServiceName { get; set; }
        public decimal Price { get; set; }
        public string Measurement { get; set; }
        public string Description { get; set; }
    }

    public class ServiceCatalogDto : ServiceCatalogCreateDto
    {
        public int ServiceId { get; set; }
    }
}
//StatusApplicationDto
namespace WebApplication1.DTOS
{
    public class StatusApplicationCreateDto
    {
        public string TypeStatus { get; set; }

    }

    public class StatusApplicationDto : StatusApplicationCreateDto
    {
        public int StatusApplicationId { get; set; }

    }
}
//SurveyAgreementDto
namespace WebApplication1.DTOS
{
    public class SurveyAgreementCreateDto
    {
        public int? ApplicationId { get; set; }
        public int? ReportId { get; set; }
        public int? EmployeeId { get; set; }
        public DateOnly CreateDate { get; set; }
        public bool Confirmation { get; set; }
        public decimal? PriceForOrder { get; set; }
    }
    public class SurveyAgreementDto : SurveyAgreementCreateDto
    {
        public int SurveyAgreementId { get; set; }
```

```
    }
}
//SurveyReportDto
namespace WebApplication1.DTOS
{
    public class SurveyReportCreateDto
    {
        public int? AplicationId { get; set; }
        public int? EmployeeId { get; set; }
        public string FileReport { get; set; }
    }
    public class SurveyReportDto : SurveyReportCreateDto
    {
        public int ReportId { get; set; }

    }
}
//UpdateSurveyReportDto
using Microsoft.AspNetCore.Mvc;

namespace WebApplication1.DTOS
{
    public class UpdateSurveyReportDto
    {
        [FromForm]
        public int ReportId { get; set; }

        [FromForm]
        public int ApplicationId { get; set; }

        [FromForm]
        public int EmployeeId { get; set; }

        [FromForm]
        public IFormFile? File { get; set; } // Файл необязателен для обновления
    }

}
//UploadSurveyReportDto
using Microsoft.AspNetCore.Mvc;

namespace WebApplication1.DTOS
{
    public class UploadSurveyReportDto
    {
        [FromForm]
        public int ApplicationId { get; set; }

        [FromForm]
        public int EmployeeId { get; set; }

        [FromForm]
        public IFormFile File { get; set; }
    }
}
//UsersDto
using WebApplication1.Models;

namespace WebApplication1.DTOS
{
    public class UsersUpdateDTO
    {
```

```
        public string Email { get; set; }
        public string PasswordHash { get; set; }
    }

    public class UsersCreateDTO: UsersUpdateDTO
    {
        public int RoleId { get; set; }
        public int? ClientId { get; set; }
        public int? EmployeeId { get; set; }
    }

    public class UsersDTO : UsersCreateDTO
    {
        public int UserId { get; set; }

        public string RoleName { get; set; }

    }
}
//Контроллеры
//AddressController
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System;
using WebApplication1;
using WebApplication1.DTOS;
using WebApplication1.Models;

[ApiController]
[Route("api/[controller]")]
public class AddressController : ControllerBase
{
    private readonly CenterContext _context;

    public AddressController(CenterContext context)
    {
        _context = context;
    }

    // GET: api/Address
    [HttpGet]
    [Authorize(Roles = "Admin,Employee")]
    public async Task<ActionResult<IEnumerable<AddressDto>>> GetAddresses()
    {
        var address = await _context.Addresses
            .Select(a => new AddressDto
            {
                AddressId = a.Addressid,
                CityName = a.Cityname,
                StreetName = a.Streetname,
                Number = a.Number
            }).ToListAsync();

        return Ok(address);
    }

    // GET: api/Address/5
    [HttpGet("{id}")]
    [Authorize(Roles = "Admin,Employee")]
    public async Task<ActionResult<AddressDto>> GetAddress(int id)
    {
```

```
        var address = await _context.Addresses.FindAsync(id);
        if (address == null) return NotFound();

        return Ok(new AddressDto
        {
            AddressId = address.Addressid,
            CityName = address.Cityname,
            StreetName = address.Streetname,
            Number = address.Number
        });
    }

    // POST: api/Address
    [HttpPost]
    [Authorize(Roles = "Admin")]
    public async Task<ActionResult<AddressDto>> CreateAddress(AddressCreateDto dto)
    {
        var address = new Address
        {
            Cityname = dto.CityName,
            Streetname = dto.StreetName,
            Number = dto.Number
        };

        _context.Addresses.Add(address);
        await _context.SaveChangesAsync();

        return CreatedAtAction(nameof(GetAddress), new { id = address.Addressid },
new AddressDto
        {
            AddressId = address.Addressid,
            CityName = address.Cityname,
            StreetName = address.Streetname,
            Number = address.Number
        });
    }

    // PUT: api/Address/5
    [HttpPut("{id}")]
    [Authorize(Roles = "Admin")]
    public async Task<IActionResult> UpdateAddress(int id, AddressCreateDto dto)
    {
        var address = await _context.Addresses.FindAsync(id);
        if (address == null) return NotFound();

        address.Cityname = dto.CityName;
        address.Streetname = dto.StreetName;
        address.Number = dto.Number;

        await _context.SaveChangesAsync();
        return NoContent();
    }

    // DELETE: api/Address/5
    [HttpDelete("{id}")]
    [Authorize(Roles = "Admin")]
    public async Task<IActionResult> DeleteAddress(int id)
    {
        var address = await _context.Addresses.FindAsync(id);
        if (address == null) return NotFound();

        _context.Addresses.Remove(address);
```

```
        await _context.SaveChangesAsync();
        return NoContent();
    }
}
//AnalysisController
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace WebApplication1.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class AnalysisController : ControllerBase
    {
        private readonly AnalysisService _service;
        private readonly CenterContext _context;

        public AnalysisController(AnalysisService service, CenterContext context)
        {
            _service = service;
            _context = context;
        }

        [Authorize(Roles = "Admin,Client")]
        [HttpGet("Client")]
        public async Task<IActionResult> GetClientApplications([FromQuery] string?
phone = null)
        {
            try
            {
                // Если пользователь - Client, вытаскиваем телефон из базы по его
ClientId из токена
                if (User.IsInRole("Client"))
                {
                    var clientIdStr = User.FindFirst("ClientId")?.Value;
                    if (string.IsNullOrEmpty(clientIdStr)) return Unauthorized("Не
удалось получить идентификатор клиента");

                    var clientId = int.Parse(clientIdStr);
                    var client = await _context.Clients.FindAsync(clientId);
                    if (client == null) return NotFound("Клиент не найден");

                    phone = client.Phone;
                }

                if (string.IsNullOrEmpty(phone))
                    return BadRequest("Необходимо указать номер телефона");

                var result = await _service.AnalyzeClientApplications(phone);
                return Ok(result);
            }
            catch (Exception ex)
            {
                return BadRequest($"Ошибка: {ex.Message}");
            }
        }


        [Authorize(Roles = "Admin,Employee")]
        [HttpGet("organization")]
```

```
        public async Task<IActionResult> GetOrganizationAnalysis([FromQuery]
DateOnly? start, [FromQuery] DateOnly? end)
        {
            try
            {
                var result = await _service.AnalyzeOrganizations(start, end);
                return Ok(result);
            }
            catch (Exception ex)
            {
                return BadRequest(ex.Message);
            }
        }

        [Authorize(Roles = "Admin,Employee")]
        [HttpGet("brigade")]
        public async Task<IActionResult> GetBrigadeAnalysis([FromQuery] DateOnly?
start, [FromQuery] DateOnly? end)
        {
            try
            {
                var result = await _service.AnalyzeBrigades(start, end);
                return Ok(result);
            }
            catch (Exception ex)
            {
                return BadRequest(ex.Message);
            }
        }
    }

}
//ApplicationControllers
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System;
using WebApplication1;
using WebApplication1.DTOS;
using WebApplication1.Models;

[ApiController]
[Route("api/[controller]")]
public class ApplicationController : ControllerBase
{
    private readonly CenterContext _context;

    public ApplicationController(CenterContext context)
    {
        _context = context;
    }

    // GET: api/Application
    [HttpGet]
    [Authorize(Roles = "Admin,Employee")]
    public async Task<ActionResult<IEnumerable<ApplicationDto>>> GetApplications()
    {
        var application = await _context.Applications
            .Select(a => new ApplicationDto
            {
                ApplicationId = a.Applicationid,
```

```
                ClientId = a.Clientid,
                ObjectSurveyId = a.Objectsurveyid,
                BrigadeId = a.Brigadeid,
                IncomingDate = a.Incomingdate,
                StatusApplicationId = a.Statusapplicationid,
                StarteDate = a.Startedate,
                EndDate = a.Enddate
            }).ToListAsync();

        return Ok(application);
    }

    // GET: api/Application/5
    [HttpGet("{id}")]
    [Authorize(Roles = "Admin,Employee")]
    public async Task<ActionResult<ApplicationDto>> GetApplication(int id)
    {
        var application = await _context.Applications.FindAsync(id);
        if (application == null) return NotFound();

        return Ok(new ApplicationDto
        {
            ApplicationId = application.Applicationid,
            ClientId = application.Clientid,
            ObjectSurveyId = application.Objectsurveyid,
            BrigadeId = application.Brigadeid,
            IncomingDate = application.Incomingdate,
            StatusApplicationId = application.Statusapplicationid,
            StarteDate = application.Startedate,
            EndDate = application.Enddate
        });
    }

    // POST: api/Application
    [HttpPost]
    [Authorize(Roles = "Admin,Employee")]
    public async Task<ActionResult<ApplicationDto>>
CreateApplication(ApplicationCreateDto dto)
    {
        var application = new Application
        {
            Clientid = dto.ClientId,
            Objectsurveyid = dto.ObjectSurveyId,
            Brigadeid = dto.BrigadeId,
            Incomingdate = dto.IncomingDate,
            Statusapplicationid = dto.StatusApplicationId,
            Startedate = dto.StarteDate,
            Enddate = dto.EndDate
        };

        _context.Applications.Add(application);
        await _context.SaveChangesAsync();

        return CreatedAtAction(nameof(GetApplication), new { id =
application.Applicationid }, new ApplicationDto
        {
            ClientId = application.Clientid,
            ObjectSurveyId = application.Objectsurveyid,
            BrigadeId = application.Brigadeid,
            IncomingDate = application.Incomingdate,
            StatusApplicationId = application.Statusapplicationid,
            StarteDate = application.Startedate,
```

```
            EndDate = application.Enddate
        });
    }

    // PUT: api/Application/5
    [HttpPut("{id}")]
    [Authorize(Roles = "Admin,Employee")]
    public async Task<IActionResult> UpdateApplication(int id, ApplicationCreateDto
dto)
    {
        var application = await _context.Applications.FindAsync(id);
        if (application == null) return NotFound();

        application.Clientid = dto.ClientId;
        application.Objectsurveyid = dto.ObjectSurveyId;
        application.Brigadeid = dto.BrigadeId;
        application.Incomingdate = dto.IncomingDate;
        application.Statusapplicationid = dto.StatusApplicationId;
        application.Startedate = dto.StarteDate;
        application.Enddate = dto.EndDate;

        await _context.SaveChangesAsync();
        return NoContent();
    }

    // DELETE: api/Application/5
    [HttpDelete("{id}")]
    [Authorize(Roles = "Admin")]
    public async Task<IActionResult> DeleteApplication(int id)
    {
        var application = await _context.Applications.FindAsync(id);
        if (application == null) return NotFound();

        _context.Applications.Remove(application);
        await _context.SaveChangesAsync();
        return NoContent();
    }
}
//AuthController
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.AspNetCore.Identity;
using Microsoft.IdentityModel.Tokens;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;
using WebApplication1.DTOS;
using WebApplication1.Models;
using BCrypt.Net;
using Microsoft.AspNetCore.Authorization;

namespace WebApplication1.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class AuthController : ControllerBase
    {
        private readonly CenterContext _context;
        private readonly IConfiguration _configuration;
        private readonly PasswordHasher<Users> _passwordHasher;

        public AuthController(CenterContext context, IConfiguration configuration)
```

```
        {
            _context = context;
            _configuration = configuration;
            _passwordHasher = new PasswordHasher<Users>();
        }

        private async Task<int> GetRoleId(string roleName)
        {
            var role = await _context.Roleses.FirstOrDefaultAsync(r => r.Rolename ==
roleName);
            if (role == null)
                throw new Exception($"Роль '{roleName}' не найдена");
            return role.Roleid;
        }


        [HttpPost("register/client")]
    public async Task<IActionResult> RegisterClient(RegisterClientDto dto)
    {
        if (await _context.Userses.AnyAsync(u => u.Email == dto.Email))
            return BadRequest("Пользователь с таким Email уже существует");

        var client = new Client
        {
            Fio = dto.Fio,
            Phone = dto.Phone
        };

        _context.Clients.Add(client);
        await _context.SaveChangesAsync();

        var user = new Users
        {
            Email = dto.Email,
            Passwordhash = BCrypt.Net.BCrypt.HashPassword(dto.Password),
            Clientid = client.Clientid,
            Roleid = await GetRoleId("Client")
        };

        _context.Userses.Add(user);
        await _context.SaveChangesAsync();

        return Ok("Клиент успешно зарегистрирован");
    }

    [HttpPost("register/employee")]
        [Authorize(Roles = "Admin")]
        public async Task<IActionResult> RegisterEmployee(RegisterEmployeeDto dto)
    {
        if (await _context.Userses.AnyAsync(u => u.Email == dto.Email))
            return BadRequest("Пользователь с таким Email уже существует");

        var employee = new Employee
        {
            Fio = dto.Fio,
            Phone = dto.Phone,
            Jobtitleid = dto.JobTitleId,
            Brigadeid = dto.BrigadeId
        };

        _context.Employees.Add(employee);
        await _context.SaveChangesAsync();
```

```
        var user = new Users
        {
            Email = dto.Email,
            Passwordhash = BCrypt.Net.BCrypt.HashPassword(dto.Password),
            Employeeid = employee.Employeeid,
            Roleid = await GetRoleId("Employee")
        };

        _context.Userses.Add(user);
        await _context.SaveChangesAsync();

        return Ok("Сотрудник успешно зарегистрирован");
    }


        [HttpPost("login")]
        public async Task<IActionResult> Login([FromBody] LoginDto dto)
        {
            var user = await _context.Userses
                .Include(u => u.Roles)
                .FirstOrDefaultAsync(u => u.Email == dto.Email);

            if (user == null)
                return Unauthorized(new { error = "Неверный email или пароль" });

            if (!BCrypt.Net.BCrypt.Verify(dto.Password, user.Passwordhash))
                return Unauthorized(new { error = "Неверный email или пароль" });

            var claims = new List<Claim>
    {
        new Claim(ClaimTypes.Name, user.Email),
        new Claim(ClaimTypes.NameIdentifier, user.Userid.ToString()),
        new Claim(ClaimTypes.Role, user.Roles?.Rolename ?? "User")
    };

            if (user.Clientid.HasValue)
                claims.Add(new Claim("ClientId", user.Clientid.Value.ToString()));

            if (user.Employeeid.HasValue)
                claims.Add(new Claim("EmployeeId",
user.Employeeid.Value.ToString()));

            var key = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_configuration["Jwt:Key"]));
            var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

            var token = new JwtSecurityToken(
                issuer: _configuration["Jwt:Issuer"],
                audience: _configuration["Jwt:Audience"],
                claims: claims,
                expires: DateTime.Now.AddHours(3),
                signingCredentials: creds
            );

            return Ok(new
            {
                token = new JwtSecurityTokenHandler().WriteToken(token)
            });
        }
```

```
    }
}
//BrigadeController
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System;
using WebApplication1;
using WebApplication1.DTOS;
using WebApplication1.Models;

[ApiController]
[Route("api/[controller]")]
public class BrigadeController : ControllerBase
{
    private readonly CenterContext _context;

    public BrigadeController(CenterContext context)
    {
        _context = context;
    }

    // GET: api/Brigade
    [HttpGet]
    [Authorize(Roles = "Admin,Employee")]
    public async Task<ActionResult<IEnumerable<BrigadeDto>>> GetBrigades()
    {
        var brigades = await _context.Brigades
            .Select(b => new BrigadeDto
            {
                BrigadeId = b.Brigadeid,
                BrigadeName = b.Brigadename
            }).ToListAsync();

        return Ok(brigades);
    }

    // GET: api/Brigade/5
    [HttpGet("{id}")]
    [Authorize(Roles = "Admin,Employee")]
    public async Task<ActionResult<BrigadeDto>> GetBrigade(int id)
    {
        var brigade = await _context.Brigades.FindAsync(id);
        if (brigade == null) return NotFound();

        return Ok(new BrigadeDto
        {
            BrigadeId = brigade.Brigadeid,
            BrigadeName = brigade.Brigadename
        });
    }

    // POST: api/Brigade
    [HttpPost]
    [Authorize(Roles = "Admin")]
    public async Task<ActionResult<BrigadeDto>> CreateBrigade(BrigadeCreateDto dto)
    {
        var brigade = new Brigade
        {
            Brigadename = dto.BrigadeName,
        };
```

```
        _context.Brigades.Add(brigade);
        await _context.SaveChangesAsync();

        return CreatedAtAction(nameof(GetBrigade), new { id = brigade.Brigadeid },
new BrigadeDto
        {
            BrigadeId = brigade.Brigadeid,
            BrigadeName = brigade.Brigadename
        });
    }

    // PUT: api/Brigade/5
    [HttpPut("{id}")]
    [Authorize(Roles = "Admin")]
    public async Task<IActionResult> UpdateBrigade(int id, BrigadeCreateDto dto)
    {
        var brigade = await _context.Brigades.FindAsync(id);
        if (brigade == null) return NotFound();

        brigade.Brigadename = dto.BrigadeName;

        await _context.SaveChangesAsync();
        return NoContent();
    }

    // DELETE: api/Brigade/5
    [HttpDelete("{id}")]
    [Authorize(Roles = "Admin")]
    public async Task<IActionResult> DeleteBrigade(int id)
    {
        var brigade = await _context.Brigades.FindAsync(id);
        if (brigade == null) return NotFound();

        _context.Brigades.Remove(brigade);
        await _context.SaveChangesAsync();
        return NoContent();
    }
}
//ClientController
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System;
using WebApplication1;
using WebApplication1.DTOS;
using WebApplication1.Models;

[ApiController]
[Route("api/[controller]")]
public class ClientController : ControllerBase
{
    private readonly CenterContext _context;

    public ClientController(CenterContext context)
    {
        _context = context;
    }

    // GET: api/Client
    [HttpGet]
    [Authorize(Roles = "Admin,Employee")]
    public async Task<ActionResult<IEnumerable<ClientDto>>> GetClients()
```

```
    {
        var clients = await _context.Clients
            .Select(c => new ClientDto
            {
                ClientId = c.Clientid,
                Fio = c.Fio,
                Phone = c.Phone
            }).ToListAsync();

        return Ok(clients);
    }

    // GET: api/Client/5
    [HttpGet("{id}")]
    [Authorize(Roles = "Admin,Employee")]
    public async Task<ActionResult<ClientDto>> GetClient(int id)
    {
        var client = await _context.Clients.FindAsync(id);
        if (client == null) return NotFound();

        return Ok(new ClientDto
        {
            ClientId = client.Clientid,
            Fio = client.Fio,
            Phone = client.Phone
        });
    }

    // POST: api/Client
    [HttpPost]
    [Authorize(Roles = "Admin")]
    public async Task<ActionResult<ClientDto>> CreateClient(ClientCreateDto dto)
    {
        var client = new Client
        {
            Fio = dto.Fio,
            Phone = dto.Phone,
        };

        _context.Clients.Add(client);
        await _context.SaveChangesAsync();

        return CreatedAtAction(nameof(GetClient), new { id = client.Clientid }, new
ClientDto
        {
            ClientId = client.Clientid,
            Fio = client.Fio,
            Phone = client.Phone
        });
    }

    // PUT: api/Client/5
    [HttpPut("{id}")]
    [Authorize(Roles = "Admin")]
    public async Task<IActionResult> UpdateClient(int id, ClientCreateDto dto)
    {
        var client = await _context.Clients.FindAsync(id);
        if (client == null) return NotFound();

        client.Fio = dto.Fio;
        client.Phone = dto.Phone;
```

```
        await _context.SaveChangesAsync();
        return NoContent();
    }

    // DELETE: api/Client/5
    [HttpDelete("{id}")]
    [Authorize(Roles = "Admin")]
    public async Task<IActionResult> DeleteClient(int id)
    {
        var client = await _context.Clients.FindAsync(id);
        if (client == null) return NotFound();

        _context.Clients.Remove(client);
        await _context.SaveChangesAsync();
        return NoContent();
    }

}
//EmployeeController
using Microsoft.AspNetCore.Mvc;
using WebApplication1.DTOS;
using WebApplication1.Models;
using WebApplication1;
using Microsoft.EntityFrameworkCore;
using Microsoft.AspNetCore.Authorization;

[ApiController]
[Route("api/[controller]")]

public class EmployeeController : ControllerBase
{
    private readonly CenterContext _context;

    public EmployeeController(CenterContext context)
    {
        _context = context;
    }

    // GET: api/Employee
    [Authorize(Roles = "Admin")]
    [HttpGet]
    public async Task<ActionResult<IEnumerable<EmployeeDto>>> GetEmployees()
    {
        var list = await _context.Employees
            .Select(e => new EmployeeDto
            {
                Employeeid = e.Employeeid,
                Fio = e.Fio,
                Phone = e.Phone,
                JobTitleId = e.Jobtitleid,
                BrigadeId = e.Brigadeid
            }).ToListAsync();
        return Ok(list);
    }

    // GET: api/Employee/5
    [Authorize(Roles = "Admin,Client")]
    [HttpGet("{id}")]
    public async Task<ActionResult<EmployeeDto>> GetEmployee(int id)
    {
        var e = await _context.Employees.FindAsync(id);
        if (e == null) return NotFound();
```

```
        return Ok(new EmployeeDto
        {
            Employeeid = e.Employeeid,
            Fio = e.Fio,
            Phone = e.Phone,
            JobTitleId = e.Jobtitleid,
            BrigadeId = e.Brigadeid
        });
    }

    // POST: api/Employee
    [Authorize(Roles = "Admin")]
    [HttpPost]
    public async Task<ActionResult<EmployeeDto>> CreateEmployee(EmployeeCreateDto
dto)
    {
        var entity = new Employee
        {
            Fio = dto.Fio,
            Phone = dto.Phone,
            Jobtitleid = dto.JobTitleId,
            Brigadeid = dto.BrigadeId
        };

        _context.Employees.Add(entity);
        await _context.SaveChangesAsync();

        return CreatedAtAction(nameof(GetEmployee), new { id = entity.Employeeid },
new EmployeeDto
        {
            Employeeid = entity.Employeeid,
            Fio = entity.Fio,
            Phone = entity.Phone,
            JobTitleId = entity.Jobtitleid,
            BrigadeId = entity.Brigadeid
        });
    }

    // PUT: api/Employee/5
    [Authorize(Roles = "Admin")]
    [HttpPut("{id}")]
    public async Task<IActionResult> UpdateEmployee(int id, EmployeeCreateDto dto)
    {
        var entity = await _context.Employees.FindAsync(id);
        if (entity == null) return NotFound();

        entity.Fio = dto.Fio;
        entity.Phone = dto.Phone;
        entity.Jobtitleid = dto.JobTitleId;
        entity.Brigadeid = dto.BrigadeId;

        await _context.SaveChangesAsync();
        return NoContent();
    }

    // DELETE: api/Employee/5
    [Authorize(Roles = "Admin")]
    [HttpDelete("{id}")]
    public async Task<IActionResult> DeleteEmployee(int id)
    {
        var entity = await _context.Employees.FindAsync(id);
```

```
        if (entity == null) return NotFound();

        _context.Employees.Remove(entity);
        await _context.SaveChangesAsync();
        return NoContent();
    }
}
//JobTitleController
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System;
using WebApplication1;
using WebApplication1.DTOS;
using WebApplication1.Models;

[ApiController]
[Route("api/[controller]")]
[Authorize(Roles = "Admin")]

public class JobTitleController : ControllerBase
{
    private readonly CenterContext _context;

    public JobTitleController(CenterContext context)
    {
        _context = context;
    }

    // GET: api/JobTitle
    [HttpGet]
    public async Task<ActionResult<IEnumerable<JobTitleDto>>> GetJobTitles()
    {
        var jobtitles = await _context.Jobtitles
            .Select(j => new JobTitleDto
            {
                JobTitleId = j.Jobtitleid,
                JobTitleName = j.Jobtitlename
            }).ToListAsync();

        return Ok(jobtitles);
    }

    // GET: api/JobTitle/5
    [HttpGet("{id}")]
    public async Task<ActionResult<JobTitleDto>> GetJobTitle(int id)
    {
        var jobtitle = await _context.Jobtitles.FindAsync(id);
        if (jobtitle == null) return NotFound();

        return Ok(new JobTitleDto
        {
            JobTitleId = jobtitle.Jobtitleid,
            JobTitleName = jobtitle.Jobtitlename
        });
    }

    // POST: api/JobTitle
    [HttpPost]
    public async Task<ActionResult<JobTitleDto>> CreateJobTitle(JobTitleCreateDto
dto)
    {
```

```
        var jobtitle = new Jobtitle
        {
            Jobtitlename = dto.JobTitleName,
        };

        _context.Jobtitles.Add(jobtitle);
        await _context.SaveChangesAsync();

        return CreatedAtAction(nameof(GetJobTitle), new { id = jobtitle.Jobtitleid },
new JobTitleDto
        {
            JobTitleId = jobtitle.Jobtitleid,
            JobTitleName = jobtitle.Jobtitlename
        });
    }

    // PUT: api/JobTitle/5
    [HttpPut("{id}")]
    public async Task<IActionResult> UpdateJobTitle(int id, JobTitleCreateDto dto)
    {
        var jobtitle = await _context.Jobtitles.FindAsync(id);
        if (jobtitle == null) return NotFound();

        jobtitle.Jobtitlename = dto.JobTitleName;

        await _context.SaveChangesAsync();
        return NoContent();
    }

    // DELETE: api/JobTitle/5
    [HttpDelete("{id}")]
    public async Task<IActionResult> DeleteJobTitle(int id)
    {
        var jobtitle = await _context.Jobtitles.FindAsync(id);
        if (jobtitle == null) return NotFound();

        _context.Jobtitles.Remove(jobtitle);
        await _context.SaveChangesAsync();
        return NoContent();
    }
}
//ObjectSurveyController
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System;
using System.Security.AccessControl;
using WebApplication1;
using WebApplication1.DTOS;
using WebApplication1.Models;

[ApiController]
[Route("api/[controller]")]
public class ObjectSurveyController : ControllerBase
{
    private readonly CenterContext _context;

    public ObjectSurveyController(CenterContext context)
    {
        _context = context;
    }
```

```
    // GET: api/ObjectSurvey
    [HttpGet]
    [Authorize(Roles = "Admin,Employee")]
    public async Task<ActionResult<IEnumerable<ObjectSurveyDto>>> GetObjectSurveys()
    {
        var objectsurvey = await _context.Objectsurveys
            .Select(o => new ObjectSurveyDto
            {
                ObjectSurveyId = o.Objectsurveyid,
                ClientId = o.Clientid,
                AddressId = o.Addressid,
                OrganizationId = o.Organizationid,
                ObjectArea=o.Objectarea
            }).ToListAsync();

        return Ok(objectsurvey);
    }

    // GET: api/ObjectSurvey/5
    [HttpGet("{id}")]
    [Authorize(Roles = "Admin,Employee")]
    public async Task<ActionResult<ObjectSurveyDto>> GetObjectSurvey(int id)
    {
        var objectsurvey = await _context.Objectsurveys.FindAsync(id);
        if (objectsurvey == null) return NotFound();

        return Ok(new ObjectSurveyDto
        {
            ObjectSurveyId = objectsurvey.Objectsurveyid,
            ClientId = objectsurvey.Clientid,
            AddressId = objectsurvey.Addressid,
            OrganizationId = objectsurvey.Organizationid,
            ObjectArea = objectsurvey.Objectarea,
        });
    }

    // POST: api/ObjectSurvey
    [HttpPost]
    [Authorize(Roles = "Admin")]
    public async Task<ActionResult<ObjectSurveyDto>>
CreateObjectSurvey(ObjectSurveyCreateDto dto)
    {
        var objectsurvey = new Objectsurvey
        {
            Clientid = dto.ClientId,
            Addressid = dto.AddressId,
            Organizationid = dto.OrganizationId,
            Objectarea= dto.ObjectArea
        };

        _context.Objectsurveys.Add(objectsurvey);
        await _context.SaveChangesAsync();

        return CreatedAtAction(nameof(GetObjectSurvey), new { id =
objectsurvey.Objectsurveyid }, new ObjectSurveyDto
        {
            ClientId = objectsurvey.Clientid,
            AddressId = objectsurvey.Addressid,
            OrganizationId = objectsurvey.Organizationid,
            ObjectArea = objectsurvey.Objectarea
        });
    }
```

```
    // PUT: api/ObjectSurvey/5
    [HttpPut("{id}")]
    [Authorize(Roles = "Admin")]
    public async Task<IActionResult> UpdateObjectSurvey(int id, ObjectSurveyCreateDto
dto)
    {
        var objectsurvey = await _context.Objectsurveys.FindAsync(id);
        if (objectsurvey == null) return NotFound();

        objectsurvey.Clientid = dto.ClientId;
        objectsurvey.Addressid = dto.AddressId;
        objectsurvey.Organizationid = dto.OrganizationId;
        objectsurvey.Objectarea = dto.ObjectArea;

        await _context.SaveChangesAsync();
        return NoContent();
    }

    // DELETE: api/ObjectSurvey/5
    [HttpDelete("{id}")]
    [Authorize(Roles = "Admin")]
    public async Task<IActionResult> DeleteObjectSurvey(int id)
    {
        var objectsurvey = await _context.Objectsurveys.FindAsync(id);
        if (objectsurvey == null) return NotFound();

        _context.Objectsurveys.Remove(objectsurvey);
        await _context.SaveChangesAsync();
        return NoContent();
    }
}
//OrganizationController
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System;
using WebApplication1;
using WebApplication1.DTOS;
using WebApplication1.Models;

[ApiController]
[Route("api/[controller]")]
public class OrganizationController : ControllerBase
{
    private readonly CenterContext _context;

    public OrganizationController(CenterContext context)
    {
        _context = context;
    }

    // GET: api/Organization
    [HttpGet]
    [Authorize(Roles = "Admin,Employee")]
    public async Task<ActionResult<IEnumerable<OrganizationDto>>> GetOrganization()
    {
        var organization = await _context.Organizations
            .Select(o => new OrganizationDto
            {
                OrganizationId = o.Organizationid,
                OrganizationName = o.Organizationname,
```

```
                Inn = o.Inn
            }).ToListAsync();

        return Ok(organization);
    }

    // GET: api/Organization/5
    [HttpGet("{id}")]
    [Authorize(Roles = "Admin,Employee")]
    public async Task<ActionResult<OrganizationDto>> GetOrganizations(int id)
    {
        var organization = await _context.Organizations.FindAsync(id);
        if (organization == null) return NotFound();

        return Ok(new OrganizationDto
        {
            OrganizationId = organization.Organizationid,
            OrganizationName = organization.Organizationname,
            Inn = organization.Inn
        });
    }

    // POST: api/Organization
    [HttpPost]
    [Authorize(Roles = "Admin")]
    public async Task<ActionResult<OrganizationDto>>
CreateOrganization(OrganizationCreateDto dto)
    {
        var organization = new Organization
        {
            Organizationname = dto.OrganizationName,
            Inn = dto.Inn
        };

        _context.Organizations.Add(organization);
        await _context.SaveChangesAsync();

        return CreatedAtAction(nameof(GetOrganization), new { id =
organization.Organizationid }, new OrganizationDto
        {
            OrganizationId = organization.Organizationid,
            OrganizationName = organization.Organizationname,
            Inn = organization.Inn
        });
    }

    // PUT: api/Organization/5
    [HttpPut("{id}")]
    [Authorize(Roles = "Admin")]
    public async Task<IActionResult> UpdateOrganization(int id, OrganizationCreateDto
dto)
    {
        var organization = await _context.Organizations.FindAsync(id);
        if (organization == null) return NotFound();

        organization.Organizationname = dto.OrganizationName;
        organization.Inn = dto.Inn;

        await _context.SaveChangesAsync();
        return NoContent();
    }
```

```
    // DELETE: api/Organization/5
    [HttpDelete("{id}")]
    [Authorize(Roles = "Admin")]
    public async Task<IActionResult> DeleteOrganization(int id)
    {
        var organization = await _context.Organizations.FindAsync(id);
        if (organization == null) return NotFound();

        _context.Organizations.Remove(organization);
        await _context.SaveChangesAsync();
        return NoContent();
    }
}
//RolesController
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System;
using WebApplication1;
using WebApplication1.DTOS;
using WebApplication1.Models;

[ApiController]
[Route("api/[controller]")]
[Authorize(Roles = "Admin")]

public class RolesController : ControllerBase
{
    private readonly CenterContext _context;

    public RolesController(CenterContext context)
    {
        _context = context;
    }

    // GET: api/Roles
    [HttpGet]
    public async Task<ActionResult<IEnumerable<RolesDTO>>> GetRoleses()
    {
        var roles = await _context.Roleses
            .Select(r => new RolesDTO
            {
                RoleId = r.Roleid,
                RoleName = r.Rolename
            }).ToListAsync();

        return Ok(roles);
    }

    // GET: api/Roles/5
    [HttpGet("{id}")]
    public async Task<ActionResult<RolesDTO>> GetRoles(int id)
    {
        var roles = await _context.Roleses.FindAsync(id);
        if (roles == null) return NotFound();

        return Ok(new RolesDTO
        {
            RoleId = roles.Roleid,
            RoleName = roles.Rolename
        });
    }
```

```
    // POST: api/Roles
    [HttpPost]
    public async Task<ActionResult<RolesDTO>> CreateRoles(RolesCreateDTO dto)
    {
        var roles = new Roles
        {
            Rolename = dto.RoleName,
        };

        _context.Roleses.Add(roles);
        await _context.SaveChangesAsync();

        return CreatedAtAction(nameof(GetRoles), new { id = roles.Roleid }, new
RolesDTO
        {
            RoleId = roles.Roleid,
            RoleName = roles.Rolename
        });
    }

    // PUT: api/Roles/5
    [HttpPut("{id}")]
    public async Task<IActionResult> UpdateRoles(int id, RolesCreateDTO dto)
    {
        var roles = await _context.Roleses.FindAsync(id);
        if (roles == null) return NotFound();

        roles.Rolename = dto.RoleName;

        await _context.SaveChangesAsync();
        return NoContent();
    }

    // DELETE: api/Roles/5
    [HttpDelete("{id}")]
    public async Task<IActionResult> DeleteRoles(int id)
    {
        var roles = await _context.Roleses.FindAsync(id);
        if (roles == null) return NotFound();

        _context.Roleses.Remove(roles);
        await _context.SaveChangesAsync();
        return NoContent();
    }
}
//SelectedServicesController
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System;
using WebApplication1;
using WebApplication1.DTOS;
using WebApplication1.Models;

[ApiController]
[Route("api/[controller]")]
public class SelectedServicesController : ControllerBase
{
    private readonly CenterContext _context;

    public SelectedServicesController(CenterContext context)
```

```
    {
        _context = context;
    }

    // GET: api/SelectedServices
    [HttpGet]
    [Authorize(Roles = "Admin,Employee")]
    public async Task<ActionResult<IEnumerable<SelectedServicesDto>>>
GetSelectedServiceses()
    {
        var selectedservices = await _context.Selectedservices
            .Select(s => new SelectedServicesDto
            {
                SelectedServicesId = s.Selectedservicesid,
                ServiceId = s.Serviceid,
                ApplicationId = s.Applicationid,
                Volume = s.Volume,
                CostServices = s.Costservices
            }).ToListAsync();

        return Ok(selectedservices);
    }

    // GET: api/SelectedServices/5
    [HttpGet("{id}")]
    [Authorize(Roles = "Admin,Employee")]
    public async Task<ActionResult<SelectedServicesDto>> GetSelectedServices(int id)
    {
        var selectedservices = await _context.Selectedservices.FindAsync(id);
        if (selectedservices == null) return NotFound();

        return Ok(new SelectedServicesDto
        {
            SelectedServicesId = selectedservices.Selectedservicesid,
            ServiceId = selectedservices.Serviceid,
            ApplicationId = selectedservices.Applicationid,
            Volume = selectedservices.Volume,
            CostServices = selectedservices.Costservices
        });
    }
    // GET: api/SelectedServices/by-application/5
    [HttpGet("by-application/{applicationId}")]
    [Authorize(Roles = "Admin,Employee")]
    public async Task<ActionResult<IEnumerable<SelectedServicesDto>>>
GetSelectedServicesByApplication(int applicationId)
    {
        var selectedServices = await _context.Selectedservices
            .Where(s => s.Applicationid == applicationId)
            .Select(s => new SelectedServicesDto
            {
                SelectedServicesId = s.Selectedservicesid,
                ServiceId = s.Serviceid,
                ApplicationId = s.Applicationid,
                Volume = s.Volume,
                CostServices = s.Costservices
            })
            .ToListAsync();

        if (selectedServices == null || selectedServices.Count == 0)
            return NotFound("Для данной заявки не найдено выбранных услуг.");

        return Ok(selectedServices);
```

```
    }

    // POST: api/SelectedServices
    [HttpPost]
    [Authorize(Roles = "Admin")]
    public async Task<ActionResult<SelectedServicesDto>>
CreateSelectedServices(SelectedServicesCreateDto dto)
    {
        var selectedservices = new Selectedservice
        {
            Serviceid = dto.ServiceId,
            Applicationid = dto.ApplicationId,
            Volume = dto.Volume,
            Costservices = dto.CostServices
        };

        _context.Selectedservices.Add(selectedservices);
        await _context.SaveChangesAsync();

        return CreatedAtAction(nameof(GetSelectedServices), new { id =
selectedservices.Selectedservicesid }, new SelectedServicesDto
        {
            SelectedServicesId = selectedservices.Selectedservicesid,
            ServiceId = selectedservices.Serviceid,
            ApplicationId = selectedservices.Applicationid,
            Volume = selectedservices.Volume,
            CostServices = selectedservices.Costservices
        });
    }

    // PUT: api/SelectedServices/5
    [HttpPut("{id}")]
    [Authorize(Roles = "Admin")]
    public async Task<IActionResult> UpdateSelectedServices(int id,
SelectedServicesCreateDto dto)
    {
        var selectedservices = await _context.Selectedservices.FindAsync(id);
        if (selectedservices == null) return NotFound();

        selectedservices.Serviceid = dto.ServiceId;
        selectedservices.Applicationid = dto.ApplicationId;
        selectedservices.Volume = dto.Volume;
        selectedservices.Costservices = dto.CostServices;

        await _context.SaveChangesAsync();
        return NoContent();
    }

    // DELETE: api/SelectedServices/5
    [HttpDelete("{id}")]
    [Authorize(Roles = "Admin")]
    public async Task<IActionResult> DeleteSelectedServices(int id)
    {
        var selectedservices = await _context.Selectedservices.FindAsync(id);
        if (selectedservices == null) return NotFound();

        _context.Selectedservices.Remove(selectedservices);
        await _context.SaveChangesAsync();
        return NoContent();
    }
}
```

```
//ServiceCatalogController
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System;
using WebApplication1;
using WebApplication1.DTOS;
using WebApplication1.Models;

[ApiController]
[Route("api/[controller]")]
public class ServiceCatalogController : ControllerBase
{
    private readonly CenterContext _context;

    public ServiceCatalogController(CenterContext context)
    {
        _context = context;
    }

    // GET: api/ServiceCatalog
    [HttpGet]
    public async Task<ActionResult<IEnumerable<ServiceCatalogDto>>>
GetServiceCatalogs()
    {
        var servicecatalog = await _context.Servicecatalogs
            .Select(s => new ServiceCatalogDto
            {
                ServiceId = s.Serviceid,
                ServiceName = s.Servicename,
                Price = s.Price,
                Measurement = s.Measurement,
                Description = s.Description
            }).ToListAsync();

        return Ok(servicecatalog);
    }

    // GET: api/ServiceCatalog/5
    [HttpGet("{id}")]

    public async Task<ActionResult<ServiceCatalogDto>> GetServiceCatalog(int id)
    {
        var servicecatalog = await _context.Servicecatalogs.FindAsync(id);
        if (servicecatalog == null) return NotFound();

        return Ok(new ServiceCatalogDto
        {
            ServiceId = servicecatalog.Serviceid,
            ServiceName = servicecatalog.Servicename,
            Price = servicecatalog.Price,
            Measurement = servicecatalog.Measurement,
            Description = servicecatalog.Description
        });
    }

    // POST: api/ServiceCatalog
    [HttpPost]
    [Authorize(Roles = "Admin")]

    public async Task<ActionResult<ServiceCatalogDto>>
CreateServiceCatalog(ServiceCatalogCreateDto dto)
```

```
    {
        var servicecatalog = new Servicecatalog
        {
            Servicename = dto.ServiceName,
            Price = dto.Price,
            Measurement = dto.Measurement,
            Description = dto.Description
        };

        _context.Servicecatalogs.Add(servicecatalog);
        await _context.SaveChangesAsync();

        return CreatedAtAction(nameof(GetServiceCatalog), new { id =
servicecatalog.Serviceid }, new ServiceCatalogDto
        {
            ServiceId = servicecatalog.Serviceid,
            ServiceName = servicecatalog.Servicename,
            Price = servicecatalog.Price,
            Measurement = servicecatalog.Measurement,
            Description = servicecatalog.Description
        });
    }

    // PUT: api/ServiceCatalog/5
    [HttpPut("{id}")]
    [Authorize(Roles = "Admin")]

    public async Task<IActionResult> UpdateServiceCatalog(int id,
ServiceCatalogCreateDto dto)
    {
        var servicecatalog = await _context.Servicecatalogs.FindAsync(id);
        if (servicecatalog == null) return NotFound();

        servicecatalog.Servicename = dto.ServiceName;
        servicecatalog.Price = dto.Price;
        servicecatalog.Measurement = dto.Measurement;
        servicecatalog.Description = dto.Description;

        await _context.SaveChangesAsync();
        return NoContent();
    }

    // DELETE: api/ServiceCatalog/5
    [HttpDelete("{id}")]
    [Authorize(Roles = "Admin")]

    public async Task<IActionResult> DeleteServiceCatalog(int id)
    {
        var servicecatalog = await _context.Servicecatalogs.FindAsync(id);
        if (servicecatalog == null) return NotFound();

        _context.Servicecatalogs.Remove(servicecatalog);
        await _context.SaveChangesAsync();
        return NoContent();
    }
}
//StatusApplicationController
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System;
using WebApplication1;
```

```
using WebApplication1.DTOS;
using WebApplication1.Models;

[ApiController]
[Route("api/[controller]")]

public class StatusAplicationController : ControllerBase
{
    private readonly CenterContext _context;

    public StatusAplicationController(CenterContext context)
    {
        _context = context;
    }

    // GET: api/StatusApplication
    [HttpGet]
    public async Task<ActionResult<IEnumerable<StatusApplicationDto>>>
GetStatusApplications()
    {
        var statusapplication = await _context.Statusapplications
            .Select(s => new StatusApplicationDto
            {
                StatusApplicationId = s.Statusapplicationid,
                TypeStatus = s.Typestatus
            }).ToListAsync();

        return Ok(statusapplication);
    }

    // GET: api/StatusApplication/5
    [HttpGet("{id}")]
    public async Task<ActionResult<StatusApplicationDto>> GetStatusApplication(int
id)
    {
        var statusapplication = await _context.Statusapplications.FindAsync(id);
        if (statusapplication == null) return NotFound();

        return Ok(new StatusApplicationDto
        {
            StatusApplicationId = statusapplication.Statusapplicationid,
            TypeStatus = statusapplication.Typestatus
        });
    }

    // POST: api/StatusApplication
    [HttpPost]
    [Authorize(Roles = "Admin")]
    public async Task<ActionResult<StatusApplicationDto>>
CreateStatusApplication(StatusApplicationCreateDto dto)
    {
        var statusapplication = new Statusapplication
        {
            Typestatus = dto.TypeStatus,
        };

        _context.Statusapplications.Add(statusapplication);
        await _context.SaveChangesAsync();

        return CreatedAtAction(nameof(GetStatusApplication), new { id =
statusapplication.Statusapplicationid }, new StatusApplicationDto
        {
```

```
            StatusApplicationId = statusapplication.Statusapplicationid,
            TypeStatus = statusapplication.Typestatus
        });
    }

    // PUT: api/StatusApplication/5
    [HttpPut("{id}")]
    [Authorize(Roles = "Admin")]
    public async Task<IActionResult> UpdateStatusApplication(int id,
StatusApplicationCreateDto dto)
    {
        var statusapplication = await _context.Statusapplications.FindAsync(id);
        if (statusapplication == null) return NotFound();

        statusapplication.Typestatus = dto.TypeStatus;

        await _context.SaveChangesAsync();
        return NoContent();
    }

    // DELETE: api/StatusApplication/5
    [HttpDelete("{id}")]
    [Authorize(Roles = "Admin")]
    public async Task<IActionResult> DeleteStatusApplication(int id)
    {
        var statusapplication = await _context.Statusapplications.FindAsync(id);
        if (statusapplication == null) return NotFound();

        _context.Statusapplications.Remove(statusapplication);
        await _context.SaveChangesAsync();
        return NoContent();
    }
}
//SurveyAgreementController
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System;
using WebApplication1;
using WebApplication1.DTOS;
using WebApplication1.Models;
using static System.Net.Mime.MediaTypeNames;

[ApiController]
[Route("api/[controller]")]
public class SurveyAgreementController : ControllerBase
{
    private readonly CenterContext _context;

    public SurveyAgreementController(CenterContext context)
    {
        _context = context;
    }

    // GET: api/SurveyAgreement
    [HttpGet]
    [Authorize(Roles = "Admin,Employee")]
    public async Task<ActionResult<IEnumerable<SurveyAgreementDto>>>
GetSurveyAgreements()
    {
        var surveyagreement = await _context.Surveyagreements
            .Select(s => new SurveyAgreementDto
```

```
            {
                SurveyAgreementId = s.Surveyagreementid,
                ApplicationId = s.Applicationid,
                ReportId = s.Reportid,
                EmployeeId = s.Employeeid,
                CreateDate = s.Createdate,
                Confirmation = s.Confirmation,
                PriceForOrder = s.Priceforорder
            }).ToListAsync();

        return Ok(surveyagreement);
    }

    // GET: api/SurveyAgreement/5
    [HttpGet("{id}")]
    [Authorize(Roles = "Admin,Employee")]
    public async Task<ActionResult<SurveyAgreementDto>> GetSurveyAgreement(int id)
    {
        var surveyagreement = await _context.Surveyagreements.FindAsync(id);
        if (surveyagreement == null) return NotFound();

        return Ok(new SurveyAgreementDto
        {
            SurveyAgreementId = surveyagreement.Surveyagreementid,
            ApplicationId = surveyagreement.Applicationid,
            ReportId = surveyagreement.Reportid,
            EmployeeId = surveyagreement.Employeeid,
            CreateDate = surveyagreement.Createdate,
            Confirmation = surveyagreement.Confirmation,
            PriceForOrder = surveyagreement.Priceforорder
        });
    }

    // POST: api/SurveyAgreement
    [HttpPost]
    [Authorize(Roles = "Admin,Employee")]
    public async Task<ActionResult<SurveyAgreementDto>>
CreateSurveyAgreement(SurveyAgreementCreateDto dto)
    {
        var surveyagreement = new Surveyagreement
        {
            Applicationid = dto.ApplicationId,
            Reportid = dto.ReportId,
            Employeeid = dto.EmployeeId,
            Createdate = dto.CreateDate,
            Confirmation = dto.Confirmation,
            Priceforорder = dto.PriceForOrder
        };

        _context.Surveyagreements.Add(surveyagreement);
        await _context.SaveChangesAsync();

        return CreatedAtAction(nameof(GetSurveyAgreement), new { id =
surveyagreement.Applicationid }, new SurveyAgreementDto
        {
            ApplicationId = surveyagreement.Applicationid,
            ReportId = surveyagreement.Reportid,
            EmployeeId = surveyagreement.Employeeid,
            CreateDate = surveyagreement.Createdate,
            Confirmation = surveyagreement.Confirmation,
            PriceForOrder = surveyagreement.Priceforорder
        });
```

```
    }

    // PUT: api/SurveyAgreement/5
    [HttpPut("{id}")]
    [Authorize(Roles = "Admin,Employee")]
    public async Task<IActionResult> UpdateSurveyAgreement(int id,
SurveyAgreementCreateDto dto)
    {
        var surveyagreement = await _context.Surveyagreements.FindAsync(id);
        if (surveyagreement == null) return NotFound();

        surveyagreement.Applicationid = dto.ApplicationId;
        surveyagreement.Reportid = dto.ReportId;
        surveyagreement.Employeeid = dto.EmployeeId;
        surveyagreement.Createdate = dto.CreateDate;
        surveyagreement.Confirmation = dto.Confirmation;
        surveyagreement.Pricefororder = dto.PriceForOrder;

        await _context.SaveChangesAsync();
        return NoContent();
    }

    // DELETE: api/SurveyAgreement/5
    [HttpDelete("{id}")]
    [Authorize(Roles = "Admin")]
    public async Task<IActionResult> DeleteSurveyAgreement(int id)
    {
        var surveyagreement = await _context.Surveyagreements.FindAsync(id);
        if (surveyagreement == null) return NotFound();

        _context.Surveyagreements.Remove(surveyagreement);
        await _context.SaveChangesAsync();
        return NoContent();
    }
}
//SurveyReportController
using Microsoft.AspNetCore.Mvc;
using WebApplication1.DTOS;
using WebApplication1.Models;
using WebApplication1;
using Microsoft.EntityFrameworkCore;
using Microsoft.AspNetCore.Authorization;

[ApiController]
[Route("api/[controller]")]
public class SurveyReportController : ControllerBase
{
    private readonly CenterContext _context;
    private readonly IWebHostEnvironment _env;

    public SurveyReportController(CenterContext context, IWebHostEnvironment env)
    {
        _context = context;
        _env = env;
    }

    // GET: api/SurveyReport
    [HttpGet]
    [Authorize(Roles = "Admin,Employee")]
    public async Task<ActionResult<IEnumerable<SurveyReportDto>>> GetSurveyReports()
    {
        var reports = await _context.Surveyreports
```

```
            .Select(r => new SurveyReportDto
            {
                ReportId = r.Reportid,
                AplicationId = r.Aplicationid,
                EmployeeId = r.Employeeid,
                FileReport = r.Filereport
            }).ToListAsync();

        return Ok(reports);
    }


    // GET: api/SurveyReport/5
    [HttpGet("{id}")]
    [Authorize(Roles = "Admin,Employee")]
    public async Task<ActionResult<SurveyReportDto>> GetSurveyReport(int id)
    {
        var report = await _context.Surveyreports.FindAsync(id);
        if (report == null) return NotFound();

        return Ok(new SurveyReportDto
        {
            ReportId = report.Reportid,
            AplicationId = report.Aplicationid,
            EmployeeId = report.Employeeid,
            FileReport = report.Filereport
        });
    }

    // POST: api/SurveyReport
    [HttpPost("upload")]
    [Consumes("multipart/form-data")]
    [Authorize(Roles = "Admin,Employee")]
    public async Task<IActionResult> Upload([FromForm] UploadSurveyReportDto dto)
    {
        if (dto.File == null || dto.File.Length == 0)
            return BadRequest(new { error = "Файл не загружен" });

        var fileName = Guid.NewGuid().ToString() +
Path.GetExtension(dto.File.FileName);

        var report = new Surveyreport
        {
            Aplicationid = dto.ApplicationId,
            Employeeid = dto.EmployeeId,
            Filereport = fileName
        };

        _context.Surveyreports.Add(report);
        await _context.SaveChangesAsync(); // Сначала проверяем, что всё ок в БД

        // Только если всё хорошо — сохраняем файл
        var uploadsFolder = Path.Combine(_env.ContentRootPath, "UploadedReports");
        if (!Directory.Exists(uploadsFolder))
            Directory.CreateDirectory(uploadsFolder);

        var filePath = Path.Combine(uploadsFolder, fileName);
        using (var stream = new FileStream(filePath, FileMode.Create))
        {
            await dto.File.CopyToAsync(stream);
        }
```

```
        return Ok(new { report.Reportid });
    }


    // PUT: api/SurveyReport
    [HttpPut("update")]
    [Consumes("multipart/form-data")]
    [Authorize(Roles = "Admin,Employee")]
    public async Task<IActionResult> Update([FromForm] UpdateSurveyReportDto dto)
    {
        var report = await _context.Surveyreports.FindAsync(dto.ReportId);
        if (report == null) return NotFound("Отчёт не найден");

        report.Aplicationid = dto.ApplicationId;
        report.Employeeid = dto.EmployeeId;

        if (dto.File != null && dto.File.Length > 0)
        {
            // Удаляем старый файл
            var uploadsFolder = Path.Combine(_env.ContentRootPath,
"UploadedReports");
            var oldFilePath = Path.Combine(uploadsFolder, report.Filereport);
            if (System.IO.File.Exists(oldFilePath))
            {
                System.IO.File.Delete(oldFilePath);
            }

            // Загружаем новый файл
            var newFileName = Guid.NewGuid().ToString() +
Path.GetExtension(dto.File.FileName);
            var newFilePath = Path.Combine(uploadsFolder, newFileName);

            using (var stream = new FileStream(newFilePath, FileMode.Create))
            {
                await dto.File.CopyToAsync(stream);
            }

            report.Filereport = newFileName;
        }

        await _context.SaveChangesAsync();

        return Ok("Отчёт успешно обновлён");
    }

    // GET: api/SurveyReport/Download/5
    [HttpGet("download/{id}")]
    [Authorize(Roles = "Admin,Employee")]
    public async Task<IActionResult> DownloadSurveyReport(int id)
    {
        var report = await _context.Surveyreports.FindAsync(id);
        if (report == null) return NotFound();

        var uploadsFolder = Path.Combine(_env.ContentRootPath, "UploadedReports");
        var filePath = Path.Combine(uploadsFolder, report.Filereport);

        if (!System.IO.File.Exists(filePath))
            return NotFound("Файл не найден");

        var memory = new MemoryStream();
        using (var stream = new FileStream(filePath, FileMode.Open))
        {
```

```
            await stream.CopyToAsync(memory);
        }
        memory.Position = 0;

        return File(memory, "application/pdf", Path.GetFileName(filePath));
    }

    // DELETE: api/SurveyReport
    [HttpDelete("{id}")]
    [Authorize(Roles = "Admin")]
    public async Task<IActionResult> DeleteSurveyReport(int id)
    {
        var report = await _context.Surveyreports.FindAsync(id);
        if (report == null) return NotFound();

        var uploadsFolder = Path.Combine(_env.ContentRootPath, "UploadedReports");
        var filePath = Path.Combine(uploadsFolder, report.Filereport);
        if (System.IO.File.Exists(filePath))
            System.IO.File.Delete(filePath);

        _context.Surveyreports.Remove(report);
        await _context.SaveChangesAsync();

        return NoContent();
    }
}
//UsersController
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using WebApplication1;
using WebApplication1.DTOS;
using WebApplication1.Models;

[ApiController]
[Route("api/[controller]")]
//[Authorize(Roles = "Admin")]

public class UsersController : ControllerBase
{
    private readonly CenterContext _context;

    public UsersController(CenterContext context)
    {
        _context = context;
    }

    // GET: api/Users
    [HttpGet]
    public async Task<ActionResult<IEnumerable<UsersDTO>>> GetUserses()
    {
        var users = await _context.Userses
            .Select(u => new UsersDTO
            {
                UserId = u.Userid,
                Email = u.Email,
                PasswordHash = u.Passwordhash,
                RoleId = u.Roleid,
                ClientId = u.Clientid,
                EmployeeId = u.Employeeid
            }).ToListAsync();
```

```
        return Ok(users);
    }

    // GET: api/Users/5
    [HttpGet("{id}")]
    public async Task<ActionResult<UsersDTO>> GetUsers(int id)
    {
        var users = await _context.Userses.FindAsync(id);
        if (users == null) return NotFound();

        return Ok(new UsersDTO
        {
            UserId = users.Userid,
            Email = users.Email,
            PasswordHash = users.Passwordhash,
            RoleId = users.Roleid,
            ClientId = users.Clientid,
            EmployeeId = users.Employeeid
        });
    }

    // POST: api/Users
    [HttpPost]
    public async Task<ActionResult<UsersDTO>> CreateUsers(UsersCreateDTO dto)
    {
        var hashedPassword = BCrypt.Net.BCrypt.HashPassword(dto.PasswordHash);

        var users = new Users
        {
            Email = dto.Email,
            Passwordhash = hashedPassword,
            Roleid = dto.RoleId,
            Clientid = dto.ClientId,
            Employeeid = dto.EmployeeId
        };

        _context.Userses.Add(users);
        await _context.SaveChangesAsync();

        return CreatedAtAction(nameof(GetUsers), new { id = users.Userid }, new
UsersDTO
        {
            UserId = users.Userid,
            Email = users.Email,
            PasswordHash = users.Passwordhash,
            RoleId = users.Roleid,
            ClientId = users.Clientid,
            EmployeeId = users.Employeeid
        });
    }

    // PUT: api/Users/5
    [HttpPut("{id}")]
    public async Task<IActionResult> UpdateUsers(int id, UsersCreateDTO dto)
    {
        var users = await _context.Userses.FindAsync(id);
        if (users == null) return NotFound();

        users.Email = dto.Email;
        users.Passwordhash = BCrypt.Net.BCrypt.HashPassword(dto.PasswordHash); //
Хэширование!
        users.Roleid = dto.RoleId;
```

```
        users.Clientid = dto.ClientId;
        users.Employeeid = dto.EmployeeId;

        await _context.SaveChangesAsync();
        return NoContent();
    }

    // DELETE: api/Users/5
    [HttpDelete("{id}")]
    public async Task<IActionResult> DeleteUsers(int id)
    {
        var users = await _context.Userses.FindAsync(id);
        if (users == null) return NotFound();

        _context.Userses.Remove(users);
        await _context.SaveChangesAsync();
        return NoContent();
    }
}
//Контроллеры для работы клиента
//ApplicationClientControllers
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Security.Claims;
using System.IO;
using WebApplication1;
using WebApplication1.Models;
using WebApplication1.DTOS;

[ApiController]
[Route("api/[controller]")]
[Authorize(Roles = "Client")]
public class ApplicationClientController : ControllerBase
{
    private readonly CenterContext _context;
    private readonly IWebHostEnvironment _env;

    public ApplicationClientController(CenterContext context, IWebHostEnvironment
env)
    {
        _context = context;
        _env = env;
    }

    private int GetClientIdFromToken()
    {
        var claim = User.FindFirst("ClientId")?.Value;

        if (string.IsNullOrEmpty(claim))
            throw new UnauthorizedAccessException("ClientId отсутствует в JWT.");

        return int.Parse(claim);
    }

    // -------------------- APPLICATION --------------------

    [HttpGet]
    public async Task<IActionResult> GetMyApplications()
    {
        var clientId = int.Parse(User.FindFirst("ClientId")?.Value ?? "0");
        Console.WriteLine($"ClientId: {clientId}");
```

```
        var applications = await _context.Applications
            //.Include(a => a.Objectsurvey) // временно убрано
            .Where(a => a.Clientid == clientId)
            .ToListAsync();

        Console.WriteLine($"Applications count: {applications.Count}");

        return Ok(applications);
    }

    [HttpPost]
    public async Task<IActionResult> CreateApplication([FromBody]
ApplicationCreateForClientDto dto)
    {
        try
        {
            var clientId = GetClientIdFromToken();

            var application = new Application
            {
                Clientid = clientId,
                Objectsurveyid = dto.ObjectSurveyId,
                Brigadeid = null,
                Incomingdate = DateOnly.FromDateTime(DateTime.Now),
                Statusapplicationid = 1,
                Startedate = null,
                Enddate = null
            };

            _context.Applications.Add(application);
            await _context.SaveChangesAsync();

            return Ok(new
            {
                application.Applicationid,
                application.Clientid,
                application.Objectsurveyid,
                application.Incomingdate,
                application.Statusapplicationid
            });
        }
        catch (Exception ex)
        {
            return StatusCode(500, $"Ошибка при создании заявки: {ex.Message}");
        }
    }


    // -------------------- SELECTED SERVICES --------------------

    [HttpGet("{applicationId}/services")]
    public async Task<IActionResult> GetSelectedServices(int applicationId)
    {
        try
        {
            var clientId = GetClientIdFromToken();

            var isOwner = await _context.Applications
                .AnyAsync(a => a.Applicationid == applicationId && a.Clientid ==
clientId);
            if (!isOwner) return Forbid();
```

```
            var services = await _context.Selectedservices
                .Where(s => s.Applicationid == applicationId)
                .ToListAsync();

            return Ok(services);
        }
        catch (Exception ex)
        {
            return StatusCode(500, $"Ошибка при получении выбранных услуг:
{ex.Message}");
        }
    }

    [HttpPost("{applicationId}/services")]
    public async Task<IActionResult> AddSelectedService(int applicationId, [FromBody]
SelectedServicesCreateDto dto)
    {
        try
        {
            var clientId = GetClientIdFromToken();

            var isOwner = await _context.Applications
                .AnyAsync(a => a.Applicationid == applicationId && a.Clientid ==
clientId);
            if (!isOwner) return Forbid();

            var newService = new Selectedservice
            {
                Applicationid = applicationId,
                Serviceid = dto.ServiceId,
                Volume = dto.Volume
            };

            _context.Selectedservices.Add(newService);
            await _context.SaveChangesAsync();

            return Ok(newService);
        }
        catch (Exception ex)
        {
            return StatusCode(500, $"Ошибка при добавлении услуги: {ex.Message}");
        }
    }


    // -------------------- SURVEY REPORT --------------------

    [HttpGet("{applicationId}/report")]
    public async Task<IActionResult> GetSurveyReport(int applicationId)
    {
        try
        {
            var clientId = GetClientIdFromToken();

            var isOwner = await _context.Applications
                .AnyAsync(a => a.Applicationid == applicationId && a.Clientid ==
clientId);
            if (!isOwner) return Forbid();

            var report = await _context.Surveyreports
                .FirstOrDefaultAsync(r => r.Aplicationid == applicationId);
```

```
            if (report == null) return NotFound();

            return Ok(report);
        }
        catch (Exception ex)
        {
            return StatusCode(500, $"Ошибка при получении отчета: {ex.Message}");
        }
    }

    [HttpGet("report/download/{id}")]
    public async Task<IActionResult> DownloadSurveyReport(int id)
    {
        try
        {
            var report = await _context.Surveyreports
                .Include(r => r.Application)
                .FirstOrDefaultAsync(r => r.Reportid == id);

            if (report == null)
                return NotFound("Отчет не найден");

            var clientId = GetClientIdFromToken();
            if (report.Application?.Clientid != clientId)
                return Forbid();

            var uploadsFolder = Path.Combine(_env.ContentRootPath,
"UploadedReports");
            var filePath = Path.Combine(uploadsFolder, report.Filereport);

            if (!System.IO.File.Exists(filePath))
                return NotFound("Файл отчета не найден на сервере");

            var memory = new MemoryStream();
            using (var stream = new FileStream(filePath, FileMode.Open,
FileAccess.Read))
            {
                await stream.CopyToAsync(memory);
            }
            memory.Position = 0;

            return File(memory, "application/pdf", Path.GetFileName(filePath));
        }
        catch (Exception ex)
        {
            return StatusCode(500, $"Ошибка при скачивании отчета: {ex.Message}");
        }
    }

    // -------------------- SURVEY AGREEMENT --------------------

    [HttpGet("{applicationId}/agreement")]
    public async Task<IActionResult> GetSurveyAgreement(int applicationId)
    {
        try
        {
            var clientId = GetClientIdFromToken();

            var isOwner = await _context.Applications
                .AnyAsync(a => a.Applicationid == applicationId && a.Clientid ==
clientId);
```

```
            if (!isOwner) return Forbid();

            var agreement = await _context.Surveyagreements
                .FirstOrDefaultAsync(a => a.Applicationid == applicationId);

            if (agreement == null) return NotFound();

            return Ok(agreement);
        }
        catch (Exception ex)
        {
            return StatusCode(500, $"Ошибка при получении соглашения: {ex.Message}");
        }
    }

    [HttpPatch("{applicationId}/agreement/confirm")]
    public async Task<IActionResult> ConfirmAgreement(int applicationId)
    {
        try
        {
            var clientId = GetClientIdFromToken();

            var agreement = await _context.Surveyagreements
                .Include(a => a.Application)
                .FirstOrDefaultAsync(a => a.Applicationid == applicationId &&
a.Application.Clientid == clientId);

            if (agreement == null) return Forbid();

            agreement.Confirmation = true;
            await _context.SaveChangesAsync();

            return Ok(new { message = "Соглашение подтверждено." });
        }
        catch (Exception ex)
        {
            return StatusCode(500, $"Ошибка при подтверждении соглашения:
{ex.Message}");
        }
    }
}
//ClientProfileController
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Security.Claims;
using WebApplication1.Models;
using WebApplication1.DTOS;
using WebApplication1;

[Authorize(Roles = "Client")]
[ApiController]
[Route("api/[controller]")]
public class ClientProfileController : ControllerBase
{
    private readonly CenterContext _context;

    public ClientProfileController(CenterContext context)
    {
        _context = context;
    }
```

```
    // GET: api/ClientProfile
    [HttpGet]
    public async Task<IActionResult> GetClientProfile()
    {
        var clientId = int.Parse(User.FindFirst("ClientId")?.Value ?? "0");

        var client = await _context.Clients.FindAsync(clientId);

        return Ok(new
        {
            client.Fio,
            client.Phone
        });
    }

    // PUT: api/ClientProfile
    [HttpPut]
    public async Task<IActionResult> UpdateClientProfile([FromBody] ClientCreateDto
dto)
    {
        var clientId = int.Parse(User.FindFirst("ClientId")?.Value ?? "0");

        var client = await _context.Clients.FindAsync(clientId);
        if (client == null) return NotFound("Клиент не найден");

        client.Fio = dto.Fio;
        client.Phone = dto.Phone;

        await _context.SaveChangesAsync();
        return NoContent();
    }

    // GET: api/ClientProfile/email
    [HttpGet("email")]
    public async Task<IActionResult> GetClientEmail()
    {
        var clientId = int.Parse(User.FindFirst("ClientId")?.Value ?? "0");

        var user = await _context.Userses.FirstOrDefaultAsync(u => u.Clientid ==
clientId);
        if (user == null) return NotFound("Пользователь не найден");

        return Ok(new
        {
            user.Email
        });
    }

    // PUT: api/ClientProfile/email
    [HttpPut("email")]
    public async Task<IActionResult> UpdateClientEmailAndPassword([FromBody]
UsersUpdateDTO dto)
    {
        var clientId = int.Parse(User.FindFirst("ClientId")?.Value ?? "0");

        var user = await _context.Userses.FirstOrDefaultAsync(u => u.Clientid ==
clientId);
        if (user == null) return NotFound("Пользователь не найден");

        user.Email = dto.Email;
        user.Passwordhash = BCrypt.Net.BCrypt.HashPassword(dto.PasswordHash);
```

```
        await _context.SaveChangesAsync();
        return NoContent();
    }
}
//ObjectSurveyController
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Security.Claims;
using WebApplication1;
using WebApplication1.Models;
using WebApplication1.DTOS;

[ApiController]
[Route("api/[controller]")]
[Authorize(Roles = "Client")]
public class ObjectSurveyClientController : ControllerBase
{
    private readonly CenterContext _context;

    public ObjectSurveyClientController(CenterContext context)
    {
        _context = context;
    }

    private int GetClientIdFromToken()
    {
        return int.Parse(User.FindFirst("ClientId")?.Value ?? "0");
    }

    // GET: api/ObjectSurveyClient
    [HttpGet]

    public async Task<IActionResult> GetClientObjectSurveys()
    {
        var clientId = GetClientIdFromToken();

        var surveys = await _context.Objectsurveys
            .Where(o => o.Clientid == clientId)
            .Select(o => new
            {
                o.Objectsurveyid,
                o.Objectarea,
                Address = o.Address == null ? null : new
                {
                    o.Address.Cityname,
                    o.Address.Streetname,
                    o.Address.Number
                },
                Organization = o.Organization == null ? null : new
                {
                    o.Organization.Organizationname,
                    o.Organization.Inn
                }
            })
            .ToListAsync();

        return Ok(surveys);
    }

    // POST: api/ObjectSurveyClient
    [HttpPost]
```

```
    public async Task<IActionResult> CreateObjectSurvey([FromBody]
ObjectSurveyCreateClientDto dto)
    {
        try
        {
            var clientId = GetClientIdFromToken();

            // Проверка и создание адреса
            var address = await _context.Addresses.FirstOrDefaultAsync(a =>
                a.Cityname == dto.Address.CityName &&
                a.Streetname == dto.Address.StreetName &&
                a.Number == dto.Address.Number);

            if (address == null)
            {
                address = new Address
                {
                    Cityname = dto.Address.CityName,
                    Streetname = dto.Address.StreetName,
                    Number = dto.Address.Number
                };
                _context.Addresses.Add(address);
                await _context.SaveChangesAsync();
            }

            // Проверка и создание организации
            var organization = await _context.Organizations.FirstOrDefaultAsync(o =>
                o.Organizationname == dto.Organization.OrganizationName &&
                o.Inn == dto.Organization.Inn);

            if (organization == null)
            {
                organization = new Organization
                {
                    Organizationname = dto.Organization.OrganizationName,
                    Inn = dto.Organization.Inn
                };
                _context.Organizations.Add(organization);
                await _context.SaveChangesAsync();
            }

            var objectSurvey = new Objectsurvey
            {
                Clientid = clientId,
                Objectarea = dto.ObjectArea,
                Addressid = address.Addressid,
                Organizationid = organization.Organizationid
            };

            _context.Objectsurveys.Add(objectSurvey);
            await _context.SaveChangesAsync();

            return Ok(objectSurvey);
        }
        catch (Exception ex)
        {
            return StatusCode(500, $"Ошибка при создании объекта: {ex.Message}");
        }
    }

    // PUT: api/ObjectSurveyClient/{id}
    [HttpPut("{id}")]
```

```
    public async Task<IActionResult> UpdateObjectSurvey(int id, [FromBody]
ObjectSurveyCreateClientDto dto)
    {
        var clientId = GetClientIdFromToken();

        var objectSurvey = await _context.Objectsurveys.FindAsync(id);
        if (objectSurvey == null || objectSurvey.Clientid != clientId)
            return NotFound("Объект не найден или нет доступа");

        // Обновить или создать адрес
        var address = await _context.Addresses.FirstOrDefaultAsync(a =>
            a.Cityname == dto.Address.CityName &&
            a.Streetname == dto.Address.StreetName &&
            a.Number == dto.Address.Number);

        if (address == null)
        {
            address = new Address
            {
                Cityname = dto.Address.CityName,
                Streetname = dto.Address.StreetName,
                Number = dto.Address.Number
            };
            _context.Addresses.Add(address);
            await _context.SaveChangesAsync();
        }

        // Обновить или создать организацию
        var organization = await _context.Organizations.FirstOrDefaultAsync(o =>
            o.Organizationname == dto.Organization.OrganizationName &&
            o.Inn == dto.Organization.Inn);

        if (organization == null)
        {
            organization = new Organization
            {
                Organizationname = dto.Organization.OrganizationName,
                Inn = dto.Organization.Inn
            };
            _context.Organizations.Add(organization);
            await _context.SaveChangesAsync();
        }

        // Обновление объекта
        objectSurvey.Objectarea = dto.ObjectArea;
        objectSurvey.Addressid = address.Addressid;
        objectSurvey.Organizationid = organization.Organizationid;

        await _context.SaveChangesAsync();
        return NoContent();
    }
}
```