Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОННИКИ
Факультет компьютерных систем и сетей
Кафедра электронных вычислительных машин
Дисциплина: Базы данных

Тема «Репетиционная база»
Лабораторная работа №1
Разработка серверной части прикладной программы

Студент:	А.С. Бригадир
Преподаватель:	С.С. Силич

МИНСК 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Данная лабораторная работа предполагает создание серверной части приложения, включая разработку спецификаций, реализацию HTTP-сервера и обеспечение взаимодействия с базой данных через стандартные методы REST API.

Работа опирается на результаты лабораторной работы №6 первого семестра, где была разработана начальная реляционная схема и реализована базовая структура базы данных.

В рамках текущего задания предполагается уточнение схемы, определение ролей пользователей, разработка технических требований и программирование серверной части. Особое внимание уделяется обеспечению безопасности доступа через разделение прав между обычными пользователями и суперпользователями, а также реализации операций резервного копирования и фильтрации данных.

1 ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

1.1 Описание реляционной модели

Реляционная схема осталась без изменений и изображена в соответствии с UML на рисунке 1.1.

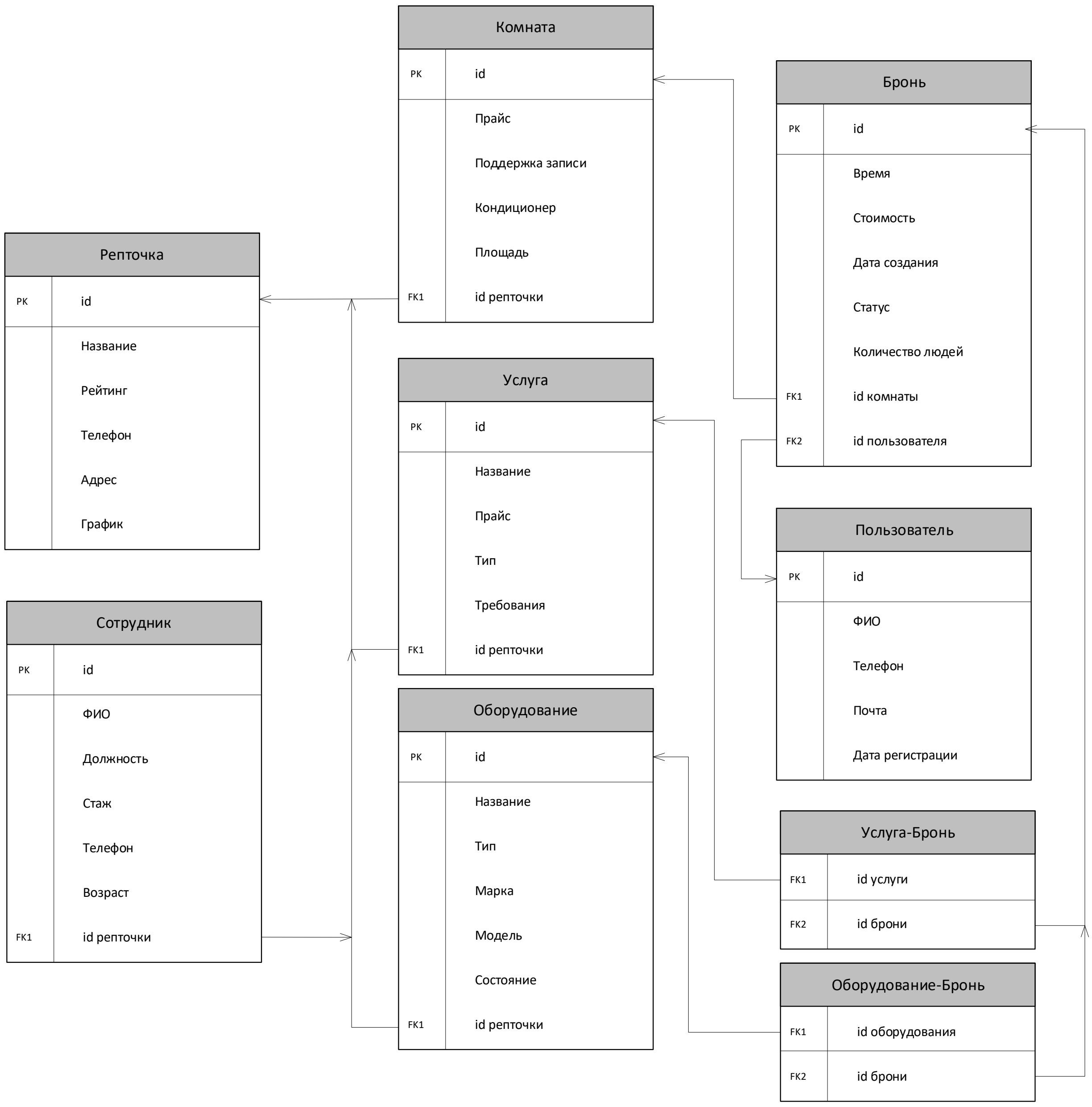


Рисунок 1.1 – Уточненная реляционная схема

1.2 Описание таблиц

Таблица `rehearsal_points` включает основные данные о репетиционных базах. Описание имен таблицы `rehearsal_points`:

– `id`: идентификатор репетиционной базы. Первичный ключ;

– rating: рейтинг базы;

– contact_number: контактный номер телефона (обязательное поле);

– schedule: расписание (JSON);

– name: название репетиционной базы (обязательное поле);

– address: адрес репетиционной базы (обязательное поле).

Таблица rooms представляет данные системы. Описание имен таблицы rooms:

– id: идентификатор комнаты. Первичный ключ;

– name: название комнаты (обязательное поле);

– air_conditioner: наличие кондиционера (по умолчанию FALSE);

– price: стоимость аренды (обязательное поле);

– recording_support: поддержка записи (по умолчанию FALSE);

– area: площадь комнаты (обязательное поле);

–id_rehearsal_point: внешний ключ на таблицу rehearsal_points.

Таблица service представляет данные о всех видах услуг. Описание имен таблицы service:

– id: идентификатор услуги. Первичный ключ;

– name: название услуги (обязательное поле);

– price: стоимость услуги (обязательное поле);

– type: тип услуги (обязательное поле);

– requirements: дополнительные требования;

–id_rehearsal_point: внешний ключ на таблицу rehearsal_points.

Таблица equipment представляет данные о всем оборудовании репетиционных точек. Описание имен таблицы equipment:

– id: идентификатор оборудования. Первичный ключ;

– name: название оборудования (обязательное поле);

– type: тип оборудования (обязательное поле);

– brand: бренд оборудования (обязательное поле);

– model: модель оборудования (обязательное поле);

– condition: состояние оборудования (обязательное поле);

-id_rehearsal_point: внешний ключ на таблицу rehearsal_points.

Таблица staff представляет данные о сотрудниках. Описание имен таблицы staff:

– id: идентификатор сотрудника. Первичный ключ;

– full_name: ФИО сотрудника (обязательное поле);

– address: адрес сотрудника;

– experience: опыт работы (в годах);

– phone: номер телефона (обязательное поле);

– age: возраст сотрудника (обязательное поле);

– id_rehearsal_point: внешний ключ на таблицу rehearsal_points.

Таблица users представляет данные о пользователях. Описание имен таблицы users:

– id: идентификатор пользователя. Первичный ключ;

– full_name: ФИО пользователя (обязательное поле);

– phone: номер телефона (обязательное поле);

– email: адрес электронной почты (обязательное поле);

– registration_date: дата регистрации (обязательное поле).

Таблица booking представляет данные о бронированиях. Описание имен таблицы booking:

– id: идентификатор бронирования. Первичный ключ;

– time: время бронирования (обязательное поле);

– duration: длительность бронирования;

– cost: стоимость (обязательное поле);

– creation_date: дата создания (обязательное поле);

– status: статус бронирования (обязательное поле);

– number_of_people: количество людей (обязательное поле);

– id_room: внешний ключ на таблицу rooms;

– id_user: внешний ключ на таблицу users.

Таблица service_booking представляет собой побочную таблицу связи many-to-many. Описание имен таблицы service_booking:

– id_service: внешний ключ на таблицу service. Первичный ключ;

– id_booking: внешний ключ на таблицу booking. Первичный ключ.

Таблица equipment_booking представляет собой побочную таблицу связи many-to-many. Описание имен таблицы equipment_booking:

– id_equipment: внешний ключ на таблицу equipment. Первичный ключ;

– id_booking: внешний ключ на таблицу booking. Первичный ключ.

1.3 Выделение справочных и основных таблиц

В данной схеме в категорию справочных таблиц должны быть выделены: service_booking и equipment_booking, так как они содержат данные для сопоставления услуг и оборудования с бронированиями и изменяются только администратором.

В качестве основной таблицы должна быть выделена таблица rehearsal_points, так как она содержит основные данные о точках репетиций, с которых начинается работа приложения.

1.4 Выделение прав доступа

Пользователь должен обладать правами просмотра, сохранения результатов запросов и редактирования всех таблиц, кроме справочных, а суперпользователь обладать теми же правами что и обычный пользователь, но с возможностью редактирования справочных таблиц и создания бэкапа базы данных. Для выполнения действий от имени суперпользователя приложение должно запрашивать пароль суперпользователя.

1.5 Определение требований к серверной части

Серверная часть прикладной программы должна быть реализована в виде HTTP-сервера. Тела ответов сервера, так же, как и тела запросов должны быть представлены в формате JSON.

Для взаимодействия с ресурсами (таблицами) должны использоваться стандартные HTTP-методы:

1) GET – получение данных о ресурсе;
2) POST – создание нового ресурса;
3) PUT – обновление существующего ресурса;
4) DELETE – удаление ресурса.

Каждый ресурс должен быть доступен по уникальному URL:

– /api/rehearsal_points: таблица `rehearsal_points`;
– /api/rooms: таблица `rooms`;
– /api/service: таблица `service`;
– /api/equipment: таблица `equipment`;
– /api/staff: таблица `staff`;
– /api/users: таблица `users`;
– /api/booking: таблица `booking`;
– /api/service_booking: таблица `service_booking`;
– /api/equipment_booking: таблица `equipment_booking`.

Серверная часть прикладной программы должна предоставлять следующие операции для работы с базой данных:

– просмотр таблиц;
– фильтрация содержимого таблиц;
– добавление записей в таблицы;
– обновление записей в таблицах;
– удаление записей из таблиц;
– выполнение специальных запросов;
– создание бэкапов базы данных;
– сохранение результатов запросов в файл.

2 ПРОГРАММИРОВАНИЕ СЕРВЕРНОЙ ЧАСТИ

2.1 Создание скриптов

Для создания таблиц в базе данных используется следующий скрипт:

```
CREATE TABLE IF NOT EXISTS main.booking (
    "Id" integer NOT NULL GENERATED BY DEFAULT AS IDENTITY,
    "Time" timestamp with time zone NOT NULL,
    "Duration" integer,
    "Cost" integer NOT NULL,
    "CreationDate" timestamp with time zone NOT NULL,
    "Status" text NOT NULL,
    "NumberOfPeople" integer NOT NULL,
    "IdRoom" integer,
    "IdUser" integer
);

CREATE TABLE IF NOT EXISTS main.equipment (
    "Id" integer NOT NULL GENERATED BY DEFAULT AS IDENTITY,
    "Name" text NOT NULL,
    "Type" text NOT NULL,
    "Brand" text NOT NULL,
    "Model" text NOT NULL,
    "Condition" text NOT NULL,
    "IdRehearsalPoint" integer
);

CREATE TABLE IF NOT EXISTS main.equipment_booking (
    "IdEquipment" integer NOT NULL,
    "IdBooking" integer NOT NULL
);

CREATE TABLE IF NOT EXISTS main.rehearsal_points (
    "Id" integer NOT NULL GENERATED BY DEFAULT AS IDENTITY,
    "Rating" real,
    "ContactNumber" text NOT NULL,
    "Schedule" text NOT NULL,
    "Name" text NOT NULL,
    "Address" text NOT NULL
);

CREATE TABLE IF NOT EXISTS main.rooms (
    "Id" integer NOT NULL GENERATED BY DEFAULT AS IDENTITY,
    "Name" text NOT NULL,
    "AirConditioner" boolean NOT NULL,
    "Price" integer NOT NULL,
    "RecordingSupport" boolean NOT NULL,
    "Area" integer NOT NULL,
    "IdRehearsalPoint" integer
);
```

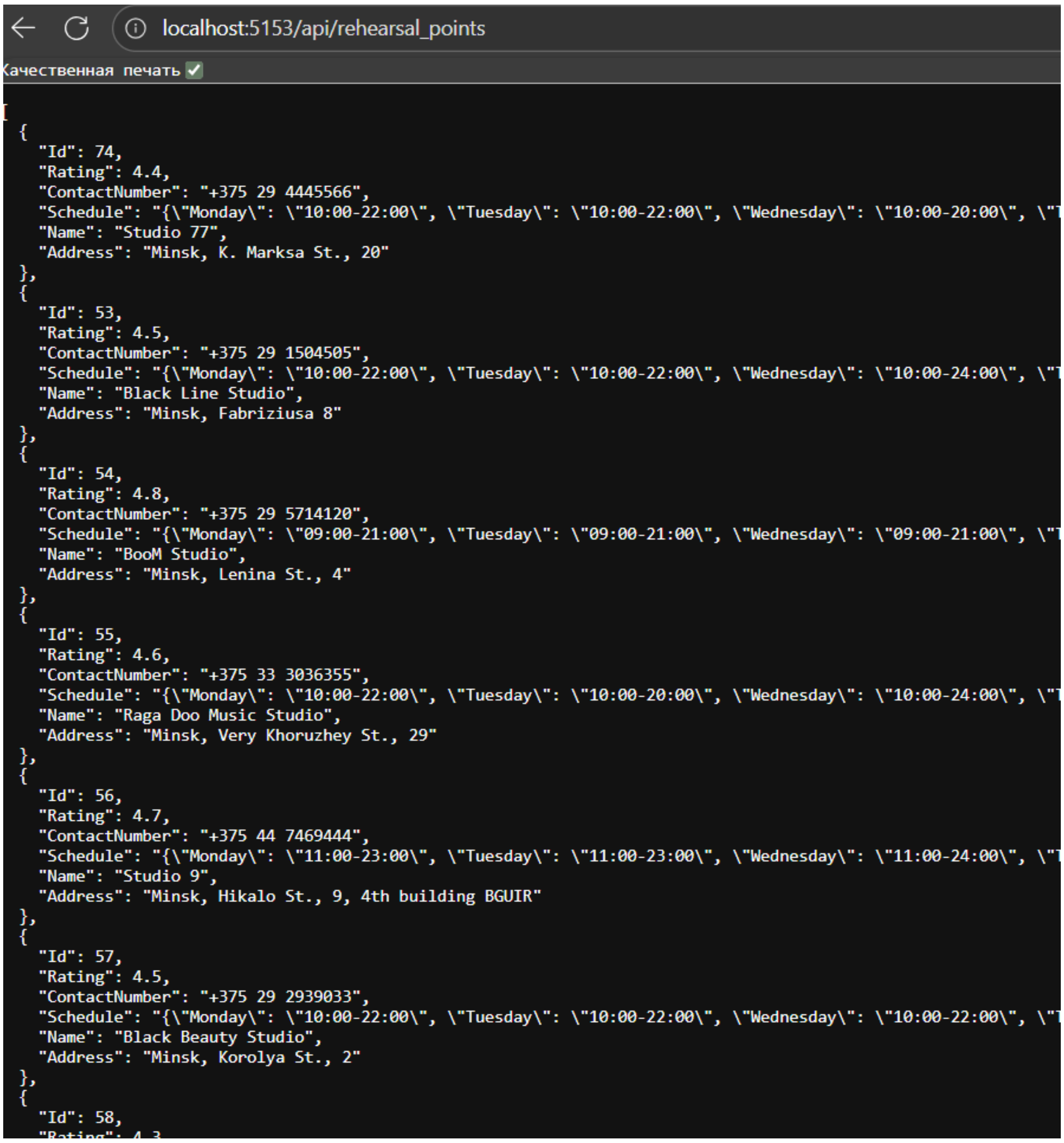
```
CREATE TABLE IF NOT EXISTS main.service (
    "Id" integer NOT NULL GENERATED BY DEFAULT AS IDENTITY,
    "Name" text NOT NULL,
    "Price" integer NOT NULL,
    "Type" text NOT NULL,
    "Requirements" text,
    "IdRehearsalPoint" integer
);

CREATE TABLE IF NOT EXISTS main.service_booking (
    "IdService" integer NOT NULL,
    "IdBooking" integer NOT NULL
);

CREATE TABLE IF NOT EXISTS main.staff (
    "Id" integer NOT NULL GENERATED BY DEFAULT AS IDENTITY,
    "FullName" text NOT NULL,
    "Address" text,
    "Experience" integer,
    "Phone" text NOT NULL,
    "Age" integer NOT NULL,
    "IdRehearsalPoint" integer
);

CREATE TABLE IF NOT EXISTS main.users (
    "Id" integer NOT NULL GENERATED BY DEFAULT AS IDENTITY,
    "FullName" text NOT NULL,
    "Phone" text NOT NULL,
    "Email" text NOT NULL,
    "RegistrationDate" timestamp with time zone NOT NULL
);

ALTER TABLE ONLY main.booking ADD CONSTRAINT PK_booking
PRIMARY KEY ("Id");

ALTER TABLE ONLY main.equipment ADD CONSTRAINT PK_equipment
PRIMARY KEY ("Id");

ALTER TABLE ONLY main.equipment_booking ADD CONSTRAINT
PK_equipment_booking PRIMARY KEY ("IdEquipment",
"IdBooking");

ALTER TABLE ONLY main.rehearsal_points ADD CONSTRAINT
PK_rehearsal_points PRIMARY KEY ("Id");

ALTER TABLE ONLY main.rooms ADD CONSTRAINT PK_rooms PRIMARY
KEY ("Id");

ALTER TABLE ONLY main.service ADD CONSTRAINT PK_service
PRIMARY KEY ("Id");
```

```
ALTER TABLE ONLY main.service_booking ADD CONSTRAINT
PK_service_booking PRIMARY KEY ("IdService", "IdBooking");

ALTER TABLE ONLY main.staff ADD CONSTRAINT PK_staff PRIMARY
KEY ("Id");

ALTER TABLE ONLY main.users ADD CONSTRAINT PK_users PRIMARY
KEY ("Id");

ALTER TABLE ONLY main.booking ADD CONSTRAINT
FK_booking_rooms_IdRoom FOREIGN KEY ("IdRoom") REFERENCES
main.rooms("Id") ON DELETE SET NULL;

ALTER TABLE ONLY main.booking ADD CONSTRAINT
FK_booking_users_IdUser FOREIGN KEY ("IdUser") REFERENCES
main.users("Id") ON DELETE CASCADE;

ALTER TABLE ONLY main.equipment_booking ADD CONSTRAINT
FK_equipment_booking_booking_IdBooking FOREIGN KEY
("IdBooking") REFERENCES main.booking("Id") ON DELETE
CASCADE;

ALTER TABLE ONLY main.equipment_booking ADD CONSTRAINT
FK_equipment_booking_equipment_IdEquipment FOREIGN KEY
("IdEquipment") REFERENCES main.equipment("Id") ON DELETE
CASCADE;

ALTER TABLE ONLY main.equipment ADD CONSTRAINT
FK_equipment_rehearsal_points_IdRehearsalPoint FOREIGN KEY
("IdRehearsalPoint") REFERENCES main.rehearsal_points("Id")
ON DELETE CASCADE;

ALTER TABLE ONLY main.rooms ADD CONSTRAINT
FK_rooms_rehearsal_points_IdRehearsalPoint FOREIGN KEY
("IdRehearsalPoint") REFERENCES main.rehearsal_points("Id")
ON DELETE CASCADE;

ALTER TABLE ONLY main.service_booking ADD CONSTRAINT
FK_service_booking_booking_IdBooking FOREIGN KEY
("IdBooking") REFERENCES main.booking("Id") ON DELETE
CASCADE;

ALTER TABLE ONLY main.service_booking ADD CONSTRAINT
FK_service_booking_service_IdService FOREIGN KEY
("IdService") REFERENCES main.service("Id") ON DELETE
CASCADE;

ALTER TABLE ONLY main.service ADD CONSTRAINT
FK_service_rehearsal_points_IdRehearsalPoint FOREIGN KEY
("IdRehearsalPoint") REFERENCES main.rehearsal_points("Id")
ON DELETE CASCADE;
```

2.2 Реализация HTTP-сервера

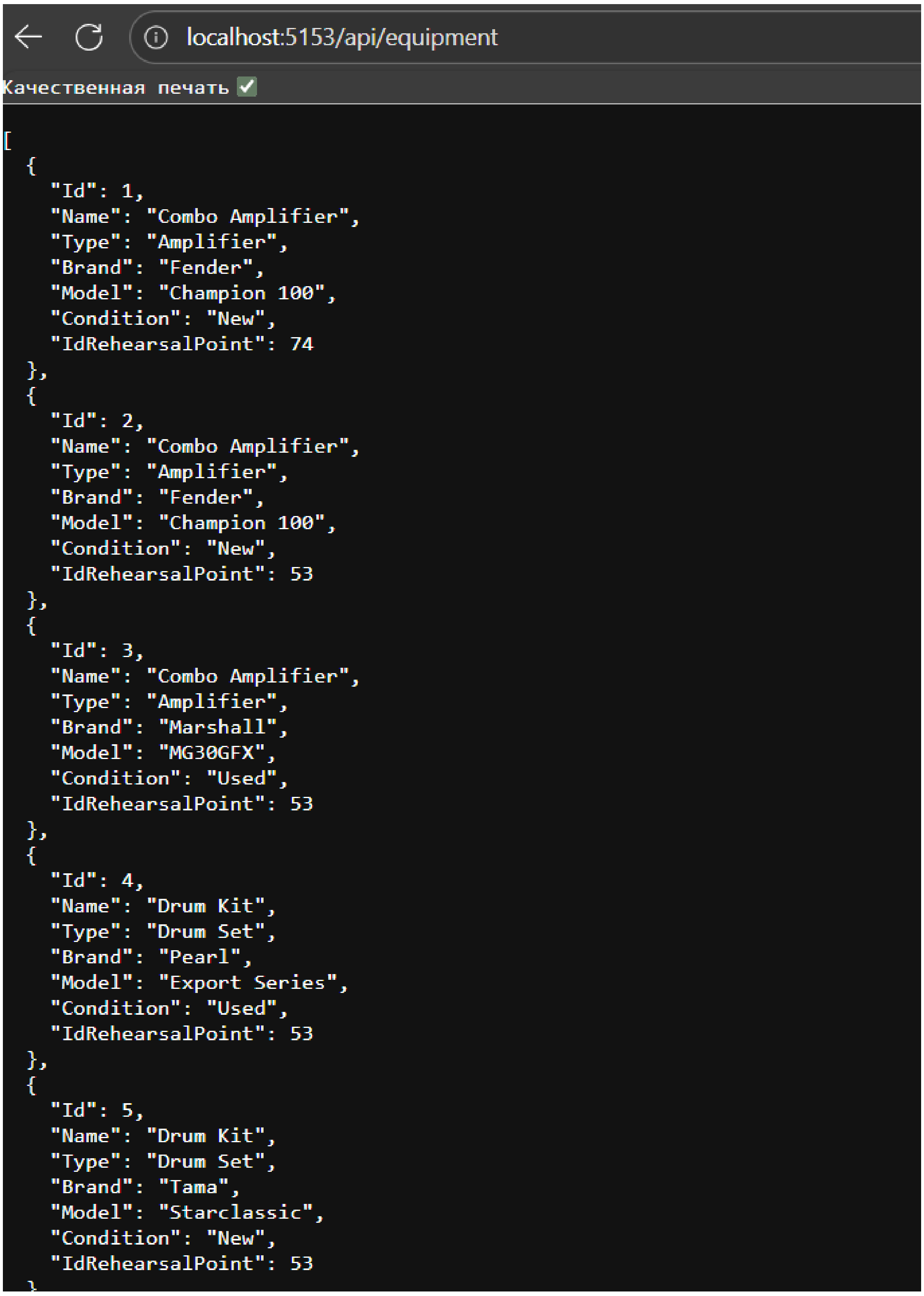
Для создания серверной части был использован DB-first подход и ORM Entity Framework Core. Для всех таблиц были описаны сущности и репозитории, которые для взаимодействия с базой данных Postgresql использовали библиотеку libpq. Также для каждой сущности и для экспорта данных были созданы соответствующие контроллеры. Листинг кода приведен в приложении А.

Для тестирования GET запросов была использована поисковая строка браузера. В качестве примера на рисунках 2.1 и 2.2 отображены результаты выполнения запросов таблиц `rehearsal_points` и `equipment` соответственно.



```
[
  {
    "Id": 74,
    "Rating": 4.4,
    "ContactNumber": "+375 29 4445566",
    "Schedule": "{\"Monday\": \"10:00-22:00\", \"Tuesday\": \"10:00-22:00\", \"Wednesday\": \"10:00-20:00\", \"T
    "Name": "Studio 77",
    "Address": "Minsk, K. Marksa St., 20"
  },
  {
    "Id": 53,
    "Rating": 4.5,
    "ContactNumber": "+375 29 1504505",
    "Schedule": "{\"Monday\": \"10:00-22:00\", \"Tuesday\": \"10:00-22:00\", \"Wednesday\": \"10:00-24:00\", \"T
    "Name": "Black Line Studio",
    "Address": "Minsk, Fabriziusa 8"
  },
  {
    "Id": 54,
    "Rating": 4.8,
    "ContactNumber": "+375 29 5714120",
    "Schedule": "{\"Monday\": \"09:00-21:00\", \"Tuesday\": \"09:00-21:00\", \"Wednesday\": \"09:00-21:00\", \"T
    "Name": "BooM Studio",
    "Address": "Minsk, Lenina St., 4"
  },
  {
    "Id": 55,
    "Rating": 4.6,
    "ContactNumber": "+375 33 3036355",
    "Schedule": "{\"Monday\": \"10:00-22:00\", \"Tuesday\": \"10:00-20:00\", \"Wednesday\": \"10:00-24:00\", \"T
    "Name": "Raga Doo Music Studio",
    "Address": "Minsk, Very Khoruzhey St., 29"
  },
  {
    "Id": 56,
    "Rating": 4.7,
    "ContactNumber": "+375 44 7469444",
    "Schedule": "{\"Monday\": \"11:00-23:00\", \"Tuesday\": \"11:00-23:00\", \"Wednesday\": \"11:00-24:00\", \"T
    "Name": "Studio 9",
    "Address": "Minsk, Hikalo St., 9, 4th building BGUIR"
  },
  {
    "Id": 57,
    "Rating": 4.5,
    "ContactNumber": "+375 29 2939033",
    "Schedule": "{\"Monday\": \"10:00-22:00\", \"Tuesday\": \"10:00-22:00\", \"Wednesday\": \"10:00-22:00\", \"T
    "Name": "Black Beauty Studio",
    "Address": "Minsk, Korolya St., 2"
  },
  {
    "Id": 58,
    "Rating": 4.3
```

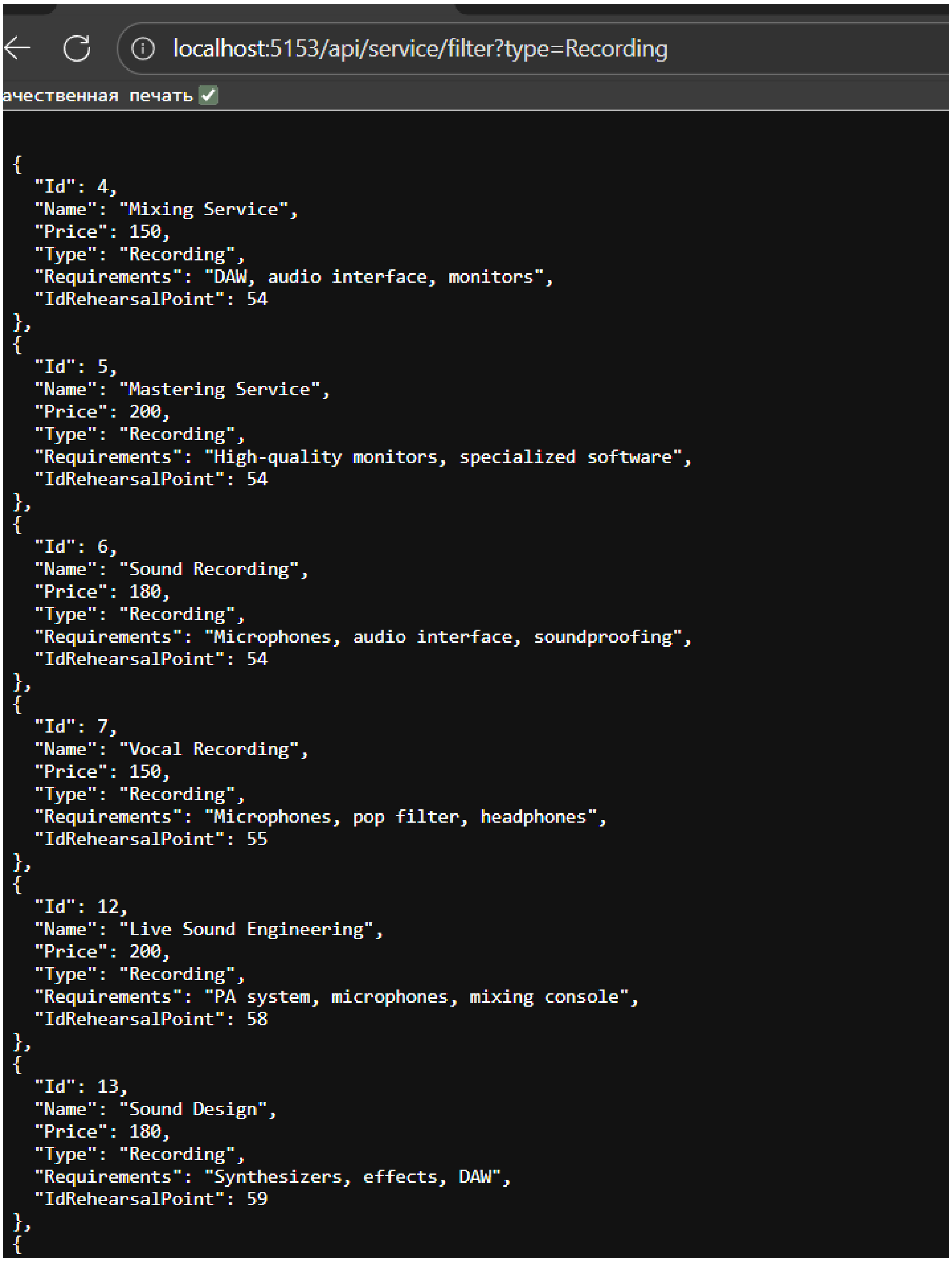
Рисунок 2.1 – Результат запроса таблицы `rehearsal_points`



```
[
  {
    "Id": 1,
    "Name": "Combo Amplifier",
    "Type": "Amplifier",
    "Brand": "Fender",
    "Model": "Champion 100",
    "Condition": "New",
    "IdRehearsalPoint": 74
  },
  {
    "Id": 2,
    "Name": "Combo Amplifier",
    "Type": "Amplifier",
    "Brand": "Fender",
    "Model": "Champion 100",
    "Condition": "New",
    "IdRehearsalPoint": 53
  },
  {
    "Id": 3,
    "Name": "Combo Amplifier",
    "Type": "Amplifier",
    "Brand": "Marshall",
    "Model": "MG30GFX",
    "Condition": "Used",
    "IdRehearsalPoint": 53
  },
  {
    "Id": 4,
    "Name": "Drum Kit",
    "Type": "Drum Set",
    "Brand": "Pearl",
    "Model": "Export Series",
    "Condition": "Used",
    "IdRehearsalPoint": 53
  },
  {
    "Id": 5,
    "Name": "Drum Kit",
    "Type": "Drum Set",
    "Brand": "Tama",
    "Model": "Starclassic",
    "Condition": "New",
    "IdRehearsalPoint": 53
```

Рисунок 2.2 – Результат запроса таблицы `equipment`

Также была реализована фильтрация по одному или нескольким параметрам. В качестве примера на рисунке 2.3 приведен результат запроса таблицы service с примененной фильтрации по типу услуги, а именно услуги записи.

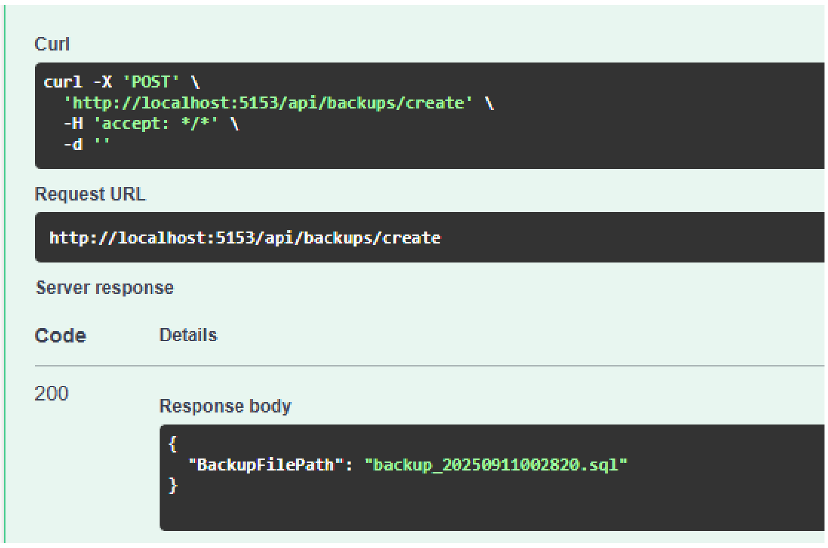

localhost:5153/api/service/filter?type=Recording

ачественная печать ✓

```
{
  "Id": 4,
  "Name": "Mixing Service",
  "Price": 150,
  "Type": "Recording",
  "Requirements": "DAW, audio interface, monitors",
  "IdRehearsalPoint": 54
},
{
  "Id": 5,
  "Name": "Mastering Service",
  "Price": 200,
  "Type": "Recording",
  "Requirements": "High-quality monitors, specialized software",
  "IdRehearsalPoint": 54
},
{
  "Id": 6,
  "Name": "Sound Recording",
  "Price": 180,
  "Type": "Recording",
  "Requirements": "Microphones, audio interface, soundproofing",
  "IdRehearsalPoint": 54
},
{
  "Id": 7,
  "Name": "Vocal Recording",
  "Price": 150,
  "Type": "Recording",
  "Requirements": "Microphones, pop filter, headphones",
  "IdRehearsalPoint": 55
},
{
  "Id": 12,
  "Name": "Live Sound Engineering",
  "Price": 200,
  "Type": "Recording",
  "Requirements": "PA system, microphones, mixing console",
  "IdRehearsalPoint": 58
},
{
  "Id": 13,
  "Name": "Sound Design",
  "Price": 180,
  "Type": "Recording",
  "Requirements": "Synthesizers, effects, DAW",
  "IdRehearsalPoint": 59
},
{
```

Рисунок 2.3 – Результат запроса таблицы service

Для тестирования POST запросов контроллера создания бэкапов был использован Swagger. В качестве примера на рисунке 2.4 отображен результат создания бэкапа.


Curl

```
curl -X 'POST' \
  'http://localhost:5153/api/backups/create' \
  -H 'accept: */*' \
  -d ''
```

Request URL

```
http://localhost:5153/api/backups/create
```

Server response

Code Details

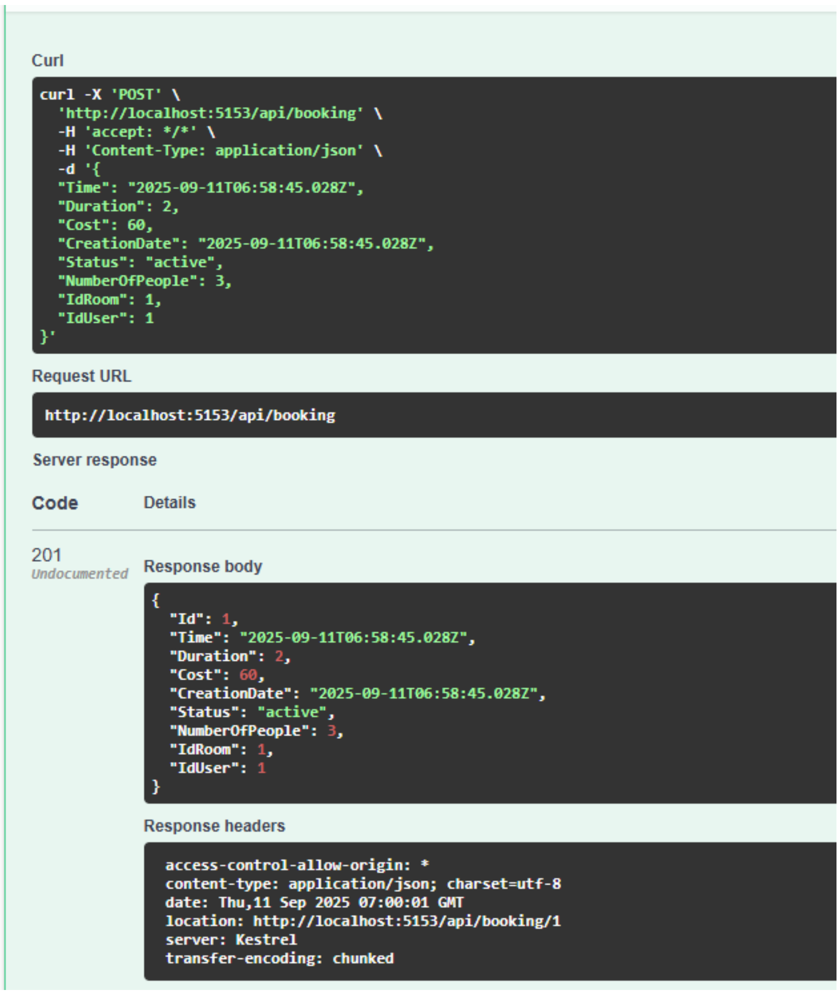
200

Response body

```
{
  "BackupFilePath": "backup_20250911002820.sql"
}
```

Рисунок 2.3 – Результат создания бэкапа базы данных

В качестве примера на рисунке 2.4 отображен результат POST запроса на создание бронирования.


Curl

```
curl -X 'POST' \
  'http://localhost:5153/api/booking' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "Time": "2025-09-11T06:58:45.028Z",
  "Duration": 2,
  "Cost": 60,
  "CreationDate": "2025-09-11T06:58:45.028Z",
  "Status": "active",
  "NumberOfPeople": 3,
  "IdRoom": 1,
  "IdUser": 1
}'
```

Request URL

```
http://localhost:5153/api/booking
```

Server response

Code Details

201
Undocumented

Response body

```
{
  "Id": 1,
  "Time": "2025-09-11T06:58:45.028Z",
  "Duration": 2,
  "Cost": 60,
  "CreationDate": "2025-09-11T06:58:45.028Z",
  "Status": "active",
  "NumberOfPeople": 3,
  "IdRoom": 1,
  "IdUser": 1
}
```

Response headers

```
access-control-allow-origin: *
content-type: application/json; charset=utf-8
date: Thu,11 Sep 2025 07:00:01 GMT
location: http://localhost:5153/api/booking/1
server: Kestrel
transfer-encoding: chunked
```

Рисунок 2.3 – Результат POST запроса на создание бронирования

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы были успешно реализованы ключевые этапы разработки серверной части прикладной программы. На основе уточненной реляционной схемы на основе лабораторной работы №6 первого семестра были разработаны технические требования, включая определение ролей пользователей, основной таблицы и справочных таблиц. Серверная часть реализована в виде HTTP-сервера с использованием формата JSON для обмена данными, что обеспечило совместимость и удобство взаимодействия.

Реализованы стандартные HTTP-методы (GET, POST, PUT, DELETE) для работы с ресурсами, доступными по уникальным URL, а также широкий спектр операций: просмотр, фильтрация, добавление, обновление и удаление записей, выполнение специальных запросов, создание бэкапов и сохранение результатов.

ПРИЛОЖЕНИЕ А

Листинг кода

Файл Program.cs:

```
using Microsoft.EntityFrameworkCore;
using RehearsalStudio.Application.Interfaces;
using RehearsalStudio.Application.Services;
using RehearsalStudio.Infrastructure.Data;
using RehearsalStudio.Infrastructure.Repositories;
using Microsoft.OpenApi.Models;
using System.Reflection;
var builder = WebApplication.CreateBuilder(args);
// Add services to the container.
// Register Entity Framework Core with PostgreSQL
builder.Services.AddDbContext<RehearsalStudioDbContext>(options =>
options.UseNpgsql(builder.Configuration.GetConnectionString("DefaultConnection")));
// Register repositories
builder.Services.AddScoped<IRehearsalPointRepository, RehearsalPointRepository>();
builder.Services.AddScoped<IRoomRepository, RoomRepository>();
builder.Services.AddScoped<IServiceRepository, ServiceRepository>();
builder.Services.AddScoped<IEquipmentRepository, EquipmentRepository>();
builder.Services.AddScoped<IStaffRepository, StaffRepository>();
builder.Services.AddScoped<IUserRepository, UserRepository>();
builder.Services.AddScoped<IBookingRepository, BookingRepository>();
builder.Services.AddScoped<IServiceBookingRepository, ServiceBookingRepository>();
builder.Services.AddScoped<IEquipmentBookingRepository, EquipmentBookingRepository>();
// Register services
builder.Services.AddScoped<IRehearsalPointService, RehearsalPointService>();
builder.Services.AddScoped<IRoomService, RoomService>();
builder.Services.AddScoped<IServiceService, ServiceService>();
builder.Services.AddScoped<IEquipmentService, EquipmentService>();
builder.Services.AddScoped<IStaffService, StaffService>();
builder.Services.AddScoped<IUserService, UserService>();
builder.Services.AddScoped<IBookingService, BookingService>();
builder.Services.AddScoped<IServiceBookingService, ServiceBookingService>();
builder.Services.AddScoped<IEquipmentBookingService, EquipmentBookingService>();
builder.Services.AddScoped<IBackupService, BackupService>();
// Add controllers
builder.Services.AddControllers()
    .AddJsonOptions(options =>
    {
        options.JsonSerializerOptions.PropertyNamingPolicy = null; // Preserve property names as-is
    });
// Configure CORS (optional, for front-end integration)
builder.Services.AddCors(options =>
{
    options.AddPolicy("AllowAll", policy =>
    {
        policy.AllowAnyOrigin()
              .AllowAnyMethod()
              .AllowAnyHeader();
    });
});
// Configure Swagger/OpenAPI
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new OpenApiInfo
    {
        Title = "RehearsalStudio API",
        Version = "v1",
        Description = "API for managing rehearsal studio resources"
    });
});
// Build the application
var app = builder.Build();
// Configure the HTTP request pipeline
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI(c =>
    {
        c.SwaggerEndpoint("/swagger/v1/swagger.json", "RehearsalStudio API v1");
        c.RoutePrefix = string.Empty; // Serve Swagger at root (/)
    });
}
app.UseHttpsRedirection();
app.UseCors("AllowAll"); // Apply CORS policy
app.UseAuthorization();
app.MapControllers();
app.Run();
```

Файл BackupsController.cs:

```
using Microsoft.AspNetCore.Mvc;
using RehearsalStudio.Application.Interfaces;
using System.Threading.Tasks;
namespace RehearsalStudio.Api.Controllers;
[Route("api/backups")]
[ApiController]
public class BackupsController : ControllerBase
{
    private readonly IBackupService _backupService;
    public BackupsController(IBackupService backupService)
    {
        _backupService = backupService;
    }
    [HttpPost("create")]
    public async Task<IActionResult> CreateBackup()
    {
        var backupPath = await _backupService.CreateDatabaseBackupAsync();
        return Ok(new { BackupFilePath = backupPath });
    }
    [HttpPost("query-save")]
    public async Task<IActionResult> SaveQueryResults([FromBody] QuerySaveRequest request)
    {
        var resultPath = await _backupService.SaveQueryResultsToFileAsync(request.SqlQuery, request.FileFormat);
        return Ok(new { ResultFilePath = resultPath });
    }
}
public class QuerySaveRequest
{
    public string SqlQuery { get; set; } = string.Empty;
    public string FileFormat { get; set; } = "json";
}
```

Файл BookingsController.cs:

```
using Microsoft.AspNetCore.Mvc;
using RehearsalStudio.Application.DTOs;
using RehearsalStudio.Application.Interfaces;
using System.Threading.Tasks;
namespace RehearsalStudio.Api.Controllers;
[Route("api/booking")]
[ApiController]
public class BookingsController : ControllerBase
{
    private readonly IBookingService _service;
    public BookingsController(IBookingService service)
    {
        _service = service;
    }
    [HttpGet]
    public async Task<IActionResult> GetAll()
    {
        var result = await _service.GetAllAsync();
        return Ok(result);
    }
    [HttpGet("{id}")]
    public async Task<IActionResult> GetById(int id)
    {
        var result = await _service.GetByIdAsync(id);
        if (result == null)
            return NotFound();
        return Ok(result);
    }
    [HttpGet("filter")]
    public async Task<IActionResult> GetFiltered([FromQuery] string? status, [FromQuery] int? idRoom, [FromQuery] int? idUser)
    {
        var result = await _service.GetFilteredAsync(status, idRoom, idUser);
        return Ok(result);
    }
    [HttpPost]
    public async Task<IActionResult> Create([FromBody] BookingDto dto)
    {
        var result = await _service.CreateAsync(dto);
        return CreatedAtAction(nameof(GetById), new { id = result.Id }, result);
    }
    [HttpPut("{id}")]
    public async Task<IActionResult> Update(int id, [FromBody] BookingDto dto)
    {
        await _service.UpdateAsync(id, dto);
        return NoContent();
    }
    [HttpDelete("{id}")]
    public async Task<IActionResult> Delete(int id)
    {
        await _service.DeleteAsync(id);
        return NoContent();
    }
}
```

Файл EquipmentBookingsController.cs:

```
using Microsoft.AspNetCore.Mvc;
using RehearsalStudio.Application.DTOs;
using RehearsalStudio.Application.Interfaces;
using System.Threading.Tasks;
namespace RehearsalStudio.Api.Controllers;
[Route("api/equipment_booking")]
[ApiController]
public class EquipmentBookingsController : ControllerBase
{
    private readonly IEquipmentBookingService _service;
    public EquipmentBookingsController(IEquipmentBookingService service)
    {
        _service = service;
    }
    [HttpGet]
    public async Task<IActionResult> GetAll()
    {
        var result = await _service.GetAllAsync();
        return Ok(result);
    }
    [HttpGet("{idEquipment}/{idBooking}")]
    public async Task<IActionResult> GetById(int idEquipment, int idBooking)
    {
        var result = await _service.GetByIdAsync(idEquipment, idBooking);
        if (result == null)
            return NotFound();
        return Ok(result);
    }
    [HttpGet("filter")]
    public async Task<IActionResult> GetFiltered([FromQuery] int? idEquipment, [FromQuery] int? idBooking)
    {
        var result = await _service.GetFilteredAsync(idEquipment, idBooking);
        return Ok(result);
    }
    [HttpPost]
    public async Task<IActionResult> Create([FromBody] EquipmentBookingDto dto)
    {
        var result = await _service.CreateAsync(dto);
        return CreatedAtAction(nameof(GetById), new { idEquipment = result.IdEquipment, idBooking = result.IdBooking }, result);
    }
    [HttpDelete("{idEquipment}/{idBooking}")]
    public async Task<IActionResult> Delete(int idEquipment, int idBooking)
    {
        await _service.DeleteAsync(idEquipment, idBooking);
        return NoContent();
    }
}
```

Файл EquipmentController.cs:

```
using Microsoft.AspNetCore.Mvc;
using RehearsalStudio.Application.DTOs;
using RehearsalStudio.Application.Interfaces;
using System.Threading.Tasks;
namespace RehearsalStudio.Api.Controllers;
[Route("api/equipment")]
[ApiController]
public class EquipmentController : ControllerBase
{
    private readonly IEquipmentService _service;
    public EquipmentController(IEquipmentService service)
    {
        _service = service;
    }
    [HttpGet]
    public async Task<IActionResult> GetAll()
    {
        var result = await _service.GetAllAsync();
        return Ok(result);
    }
    [HttpGet("{id}")]
    public async Task<IActionResult> GetById(int id)
    {
        var result = await _service.GetByIdAsync(id);
        if (result == null)
            return NotFound();
        return Ok(result);
    }
    [HttpGet("filter")]
    public async Task<IActionResult> GetFiltered([FromQuery] string? name, [FromQuery] string? type, [FromQuery] int? idRehearsalPoint)
    {
        var result = await _service.GetFilteredAsync(name, type, idRehearsalPoint);
        return Ok(result);
    }
    [HttpPost]
    public async Task<IActionResult> Create([FromBody] EquipmentDto dto)
    {
        var result = await _service.CreateAsync(dto);
        return CreatedAtAction(nameof(GetById), new { id = result.Id }, result);
    }
    [HttpPut("{id}")]
    public async Task<IActionResult> Update(int id, [FromBody] EquipmentDto dto)
    {
        await _service.UpdateAsync(id, dto);
        return NoContent();
    }
    [HttpDelete("{id}")]
    public async Task<IActionResult> Delete(int id)
    {
        await _service.DeleteAsync(id);
        return NoContent();
    }
}
```

Файл RehearsalPointsController.cs:

```
using Microsoft.AspNetCore.Mvc;
using RehearsalStudio.Application.DTOs;
using RehearsalStudio.Application.Interfaces;
using System.Threading.Tasks;
namespace RehearsalStudio.Api.Controllers;
[Route("api/rehearsal_points")]
[ApiController]
public class RehearsalPointsController : ControllerBase
{
    private readonly IRehearsalPointService _service;
    public RehearsalPointsController(IRehearsalPointService service)
    {
        _service = service;
    }
    [HttpGet]
    public async Task<IActionResult> GetAll()
    {
        var result = await _service.GetAllAsync();
        return Ok(result);
    }
    [HttpGet("{id}")]
    public async Task<IActionResult> GetById(int id)
    {
        var result = await _service.GetByIdAsync(id);
        if (result == null)
            return NotFound();
        return Ok(result);
    }
    [HttpGet("filter")]
    public async Task<IActionResult> GetFiltered([FromQuery] string? name, [FromQuery] float? minRating)
    {
        var result = await _service.GetFilteredAsync(name, minRating);
        return Ok(result);
    }
    [HttpPost]
    public async Task<IActionResult> Create([FromBody] RehearsalPointDto dto)
    {
        var result = await _service.CreateAsync(dto);
        return CreatedAtAction(nameof(GetById), new { id = result.Id }, result);
    }
    [HttpPut("{id}")]
    public async Task<IActionResult> Update(int id, [FromBody] RehearsalPointDto dto)
    {
        await _service.UpdateAsync(id, dto);
        return NoContent();
    }
    [HttpDelete("{id}")]
    public async Task<IActionResult> Delete(int id)
    {
        await _service.DeleteAsync(id);
        return NoContent();
    }
}
```

Файл RoomsController.cs:

```
using Microsoft.AspNetCore.Mvc;
using RehearsalStudio.Application.DTOs;
using RehearsalStudio.Application.Interfaces;
using System.Threading.Tasks;
namespace RehearsalStudio.Api.Controllers;
[Route("api/rooms")]
[ApiController]
public class RoomsController : ControllerBase
{
    private readonly IRoomService _service;
    public RoomsController(IRoomService service)
    {
        _service = service;
    }
    [HttpGet]
    public async Task<IActionResult> GetAll()
    {
        var result = await _service.GetAllAsync();
        return Ok(result);
    }
    [HttpGet("{id}")]
    public async Task<IActionResult> GetById(int id)
    {
        var result = await _service.GetByIdAsync(id);
        if (result == null)
            return NotFound();
        return Ok(result);
    }
    [HttpGet("filter")]
    public async Task<IActionResult> GetFiltered([FromQuery] string? name, [FromQuery] int? minPrice, [FromQuery] int? idRehearsalPoint)
    {
        var result = await _service.GetFilteredAsync(name, minPrice, idRehearsalPoint);
        return Ok(result);
    }
    [HttpPost]
    public async Task<IActionResult> Create([FromBody] RoomDto dto)
    {
        var result = await _service.CreateAsync(dto);
        return CreatedAtAction(nameof(GetById), new { id = result.Id }, result);
    }
    [HttpPut("{id}")]
    public async Task<IActionResult> Update(int id, [FromBody] RoomDto dto)
    {
        await _service.UpdateAsync(id, dto);
        return NoContent();
    }
    [HttpDelete("{id}")]
    public async Task<IActionResult> Delete(int id)
    {
        await _service.DeleteAsync(id);
        return NoContent();
    }
}
```

Файл ServiceBookingsController.cs:

```
using Microsoft.AspNetCore.Mvc;
using RehearsalStudio.Application.DTOs;
using RehearsalStudio.Application.Interfaces;
using System.Threading.Tasks;
namespace RehearsalStudio.Api.Controllers;
[Route("api/service_booking")]
[ApiController]
public class ServiceBookingsController : ControllerBase
{
    private readonly IServiceBookingService _service;
    public ServiceBookingsController(IServiceBookingService service)
    {
        _service = service;
    }
    [HttpGet]
    public async Task<IActionResult> GetAll()
    {
        var result = await _service.GetAllAsync();
        return Ok(result);
    }
    [HttpGet("{idService}/{idBooking}")]
    public async Task<IActionResult> GetById(int idService, int idBooking)
    {
        var result = await _service.GetByIdAsync(idService, idBooking);
        if (result == null)
            return NotFound();
        return Ok(result);
    }
    [HttpGet("filter")]
    public async Task<IActionResult> GetFiltered([FromQuery] int? idService, [FromQuery] int? idBooking)
    {
        var result = await _service.GetFilteredAsync(idService, idBooking);
        return Ok(result);
    }
    [HttpPost]
    public async Task<IActionResult> Create([FromBody] ServiceBookingDto dto)
    {
        var result = await _service.CreateAsync(dto);
        return CreatedAtAction(nameof(GetById), new { idService = result.IdService, idBooking = result.IdBooking }, result);
    }
    [HttpDelete("{idService}/{idBooking}")]
    public async Task<IActionResult> Delete(int idService, int idBooking)
    {
        await _service.DeleteAsync(idService, idBooking);
        return NoContent();
    }
}
```

Файл ServicesController.cs:

```
using Microsoft.AspNetCore.Mvc;
using RehearsalStudio.Application.DTOs;
using RehearsalStudio.Application.Interfaces;
using System.Threading.Tasks;
namespace RehearsalStudio.Api.Controllers;
[Route("api/service")]
[ApiController]
public class ServicesController : ControllerBase
{
    private readonly IServiceService _service;
    public ServicesController(IServiceService service)
    {
        _service = service;
    }
    [HttpGet]
    public async Task<IActionResult> GetAll()
    {
        var result = await _service.GetAllAsync();
        return Ok(result);
    }
    [HttpGet("{id}")]
    public async Task<IActionResult> GetById(int id)
    {
        var result = await _service.GetByIdAsync(id);
        if (result == null)
            return NotFound();
        return Ok(result);
    }
    [HttpGet("filter")]
    public async Task<IActionResult> GetFiltered([FromQuery] string? name, [FromQuery] string? type, [FromQuery] int? idRehearsalPoint)
    {
        var result = await _service.GetFilteredAsync(name, type, idRehearsalPoint);
        return Ok(result);
    }
    [HttpPost]
    public async Task<IActionResult> Create([FromBody] ServiceDto dto)
    {
        var result = await _service.CreateAsync(dto);
        return CreatedAtAction(nameof(GetById), new { id = result.Id }, result);
    }
    [HttpPut("{id}")]
    public async Task<IActionResult> Update(int id, [FromBody] ServiceDto dto)
    {
        await _service.UpdateAsync(id, dto);
        return NoContent();
    }
    [HttpDelete("{id}")]
    public async Task<IActionResult> Delete(int id)
    {
        await _service.DeleteAsync(id);
        return NoContent();
    }
}
```

Файл UsersController.cs:

```
using Microsoft.AspNetCore.Mvc;
using RehearsalStudio.Application.DTOs;
using RehearsalStudio.Application.Interfaces;
using System.Threading.Tasks;
namespace RehearsalStudio.Api.Controllers;
[Route("api/users")]
[ApiController]
public class UsersController : ControllerBase
{
    private readonly IUserService _service;
    public UsersController(IUserService service)
    {
        _service = service;
    }
    [HttpGet]
    public async Task<IActionResult> GetAll()
    {
        var result = await _service.GetAllAsync();
        return Ok(result);
    }
    [HttpGet("{id}")]
    public async Task<IActionResult> GetById(int id)
    {
        var result = await _service.GetByIdAsync(id);
        if (result == null)
            return NotFound();
        return Ok(result);
    }
    [HttpGet("filter")]
    public async Task<IActionResult> GetFiltered([FromQuery] string? fullName, [FromQuery] string? email)
    {
        var result = await _service.GetFilteredAsync(fullName, email);
        return Ok(result);
    }
    [HttpPost]
    public async Task<IActionResult> Create([FromBody] UserDto dto)
    {
        var result = await _service.CreateAsync(dto);
        return CreatedAtAction(nameof(GetById), new { id = result.Id }, result);
    }
    [HttpPut("{id}")]
    public async Task<IActionResult> Update(int id, [FromBody] UserDto dto)
    {
        await _service.UpdateAsync(id, dto);
        return NoContent();
    }
    [HttpDelete("{id}")]
    public async Task<IActionResult> Delete(int id)
    {
        await _service.DeleteAsync(id);
        return NoContent();
    }
}
```

Файл .NETCoreApp,Version=v9.0.AssemblyAttributes.cs:

```
// <autogenerated />
using System;
using System.Reflection;
[assembly: global::System.Runtime.Versioning.TargetFrameworkAttribute(".NETCoreApp,Version=v9.0", FrameworkDisplayName = ".NET 9.0")]
```

Файл RehearsalStudio.Api.AssemblyInfo.cs:

```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//
```

```
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------
using System;
using System.Reflection;
[assembly: System.Reflection.AssemblyCompanyAttribute("RehearsalStudio.Api")]
[assembly: System.Reflection.AssemblyConfigurationAttribute("Debug")]
[assembly: System.Reflection.AssemblyFileVersionAttribute("1.0.0.0")]
[assembly: System.Reflection.AssemblyInformationalVersionAttribute("1.0.0")]
[assembly: System.Reflection.AssemblyProductAttribute("RehearsalStudio.Api")]
[assembly: System.Reflection.AssemblyTitleAttribute("RehearsalStudio.Api")]
[assembly: System.Reflection.AssemblyVersionAttribute("1.0.0.0")]
// Generated by the MSBuild WriteCodeFragment class.
```

Файл RehearsalStudio.Api.GlobalUsings.g.cs:

```
// <auto-generated/>
global using global::Microsoft.AspNetCore.Builder;
global using global::Microsoft.AspNetCore.Hosting;
global using global::Microsoft.AspNetCore.Http;
global using global::Microsoft.AspNetCore.Routing;
global using global::Microsoft.Extensions.Configuration;
global using global::Microsoft.Extensions.DependencyInjection;
global using global::Microsoft.Extensions.Hosting;
global using global::Microsoft.Extensions.Logging;
global using global::System;
global using global::System.Collections.Generic;
global using global::System.IO;
global using global::System.Linq;
global using global::System.Net.Http;
global using global::System.Net.Http.Json;
global using global::System.Threading;
global using global::System.Threading.Tasks;
```

Файл RehearsalStudio.Api.MvcApplicationPartsAssemblyInfo.cs:

```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------
using System;
using System.Reflection;
[assembly: Microsoft.AspNetCore.Mvc.ApplicationParts.ApplicationPartAttribute("Microsoft.AspNetCore.OpenApi")]
```

```
[assembly: Microsoft.AspNetCore.Mvc.ApplicationParts.ApplicationPartAttribute("Swashbuckle.AspNetCore.SwaggerGen")]
// Создано классом WriteCodeFragment MSBuild.
```

Файл RehearsalStudioDbContext.cs:

```
using Microsoft.EntityFrameworkCore;
using RehearsalStudio.Domain.Entities;
using Npgsql.EntityFrameworkCore.PostgreSQL;
namespace RehearsalStudio.Infrastructure.Data;
public class RehearsalStudioDbContext : DbContext
{
    public RehearsalStudioDbContext(DbContextOptions<RehearsalStudioDbContext> options)
        : base(options)
    {
    }
    public DbSet<RehearsalPoint> RehearsalPoints { get; set; }
    public DbSet<Room> Rooms { get; set; }
    public DbSet<Service> Services { get; set; }
    public DbSet<Equipment> Equipment { get; set; }
    public DbSet<Staff> Staff { get; set; }
    public DbSet<User> Users { get; set; }
    public DbSet<Booking> Bookings { get; set; }
    public DbSet<ServiceBooking> ServiceBookings { get; set; }
    public DbSet<EquipmentBooking> EquipmentBookings { get; set; }
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.HasDefaultSchema("main");
        // Table names
        modelBuilder.Entity<RehearsalPoint>().ToTable("rehearsal_points");
        modelBuilder.Entity<Room>().ToTable("rooms");
        modelBuilder.Entity<Service>().ToTable("service");
        modelBuilder.Entity<Equipment>().ToTable("equipment");
        modelBuilder.Entity<Staff>().ToTable("staff");
        modelBuilder.Entity<User>().ToTable("users");
        modelBuilder.Entity<Booking>().ToTable("booking");
        modelBuilder.Entity<ServiceBooking>().ToTable("service_booking");
        modelBuilder.Entity<EquipmentBooking>().ToTable("equipment_booking");
        // Primary keys
        modelBuilder.Entity<RehearsalPoint>().HasKey(rp => rp.Id);
        modelBuilder.Entity<Room>().HasKey(r => r.Id);
        modelBuilder.Entity<Service>().HasKey(s => s.Id);
        modelBuilder.Entity<Equipment>().HasKey(e => e.Id);
        modelBuilder.Entity<Staff>().HasKey(s => s.Id);
        modelBuilder.Entity<User>().HasKey(u => u.Id);
        modelBuilder.Entity<Booking>().HasKey(b => b.Id);
        modelBuilder.Entity<ServiceBooking>().HasKey(sb => new { sb.IdService, sb.IdBooking });
        modelBuilder.Entity<EquipmentBooking>().HasKey(eb => new { eb.IdEquipment, eb.IdBooking });
        // Auto-increment for IDs
        modelBuilder.Entity<RehearsalPoint>().Property(rp => rp.Id).ValueGeneratedOnAdd();
        modelBuilder.Entity<Room>().Property(r => r.Id).ValueGeneratedOnAdd();
        modelBuilder.Entity<Service>().Property(s => s.Id).ValueGeneratedOnAdd();
        modelBuilder.Entity<Equipment>().Property(e => e.Id).ValueGeneratedOnAdd();
        modelBuilder.Entity<Staff>().Property(s => s.Id).ValueGeneratedOnAdd();
        modelBuilder.Entity<User>().Property(u => u.Id).ValueGeneratedOnAdd();
        modelBuilder.Entity<Booking>().Property(b => b.Id).ValueGeneratedOnAdd();
        // No JSON type for Schedule
        modelBuilder.Entity<RehearsalPoint>()
            .Property(rp => rp.Schedule)
            .HasColumnType("text");
        // Foreign keys with inverse navigation
        modelBuilder.Entity<Room>()
            .HasOne(r => r.RehearsalPoint)
            .WithMany(rp => rp.Rooms)
            .HasForeignKey(r => r.IdRehearsalPoint)
            .OnDelete(DeleteBehavior.Cascade);
        modelBuilder.Entity<Service>()
            .HasOne(s => s.RehearsalPoint)
            .WithMany(rp => rp.Services)
            .HasForeignKey(s => s.IdRehearsalPoint)
            .OnDelete(DeleteBehavior.Cascade);
        modelBuilder.Entity<Equipment>()
            .HasOne(e => e.RehearsalPoint)
            .WithMany(rp => rp.Equipment)
            .HasForeignKey(e => e.IdRehearsalPoint)
            .OnDelete(DeleteBehavior.Cascade);
        modelBuilder.Entity<Staff>()
            .HasOne(s => s.RehearsalPoint)
            .WithMany(rp => rp.Staff)
            .HasForeignKey(s => s.IdRehearsalPoint)
            .OnDelete(DeleteBehavior.Cascade);
        modelBuilder.Entity<Booking>()
            .HasOne(b => b.Room)
            .WithMany(r => r.Bookings)
            .HasForeignKey(b => b.IdRoom)
            .OnDelete(DeleteBehavior.SetNull);
        modelBuilder.Entity<Booking>()
            .HasOne(b => b.User)
            .WithMany(u => u.Bookings)
            .HasForeignKey(b => b.IdUser)
            .OnDelete(DeleteBehavior.Cascade);
        modelBuilder.Entity<ServiceBooking>()
            .HasOne(sb => sb.Service)
            .WithMany(s => s.ServiceBookings)
            .HasForeignKey(sb => sb.IdService)
            .OnDelete(DeleteBehavior.Cascade);
        modelBuilder.Entity<ServiceBooking>()
            .HasOne(sb => sb.Booking)
            .WithMany(b => b.ServiceBookings)
            .HasForeignKey(sb => sb.IdBooking)
            .OnDelete(DeleteBehavior.Cascade);
        modelBuilder.Entity<EquipmentBooking>()
            .HasOne(eb => eb.Equipment)
            .WithMany(e => e.EquipmentBookings)
            .HasForeignKey(eb => eb.IdEquipment)
            .OnDelete(DeleteBehavior.Cascade);
        modelBuilder.Entity<EquipmentBooking>()
            .HasOne(eb => eb.Booking)
            .WithMany(b => b.EquipmentBookings)
            .HasForeignKey(eb => eb.IdBooking)
            .OnDelete(DeleteBehavior.Cascade);
```

```
    }
}
```

Файл BookingDto.cs:

```
using System;
namespace RehearsalStudio.Application.DTOs;
public class BookingDto
{
    public int Id { get; set; }
    public DateTime Time { get; set; }
    public int? Duration { get; set; }
    public int Cost { get; set; }
    public DateTime CreationDate { get; set; }
    public string Status { get; set; } = string.Empty;
    public int NumberOfPeople { get; set; }
    public int? IdRoom { get; set; }
    public int? IdUser { get; set; }
}
```

Файл EquipmentBookingDto.cs:

```
using System;
namespace RehearsalStudio.Application.DTOs;
public class EquipmentBookingDto
{
    public int IdEquipment { get; set; }
    public int IdBooking { get; set; }
}
```

Файл EquipmentDto.cs:

```
using System;
namespace RehearsalStudio.Application.DTOs;
public class EquipmentDto
{
    public int Id { get; set; }
    public string Name { get; set; } = string.Empty;
    public string Type { get; set; } = string.Empty;
    public string Brand { get; set; } = string.Empty;
    public string Model { get; set; } = string.Empty;
    public string Condition { get; set; } = string.Empty;
    public int? IdRehearsalPoint { get; set; }
}
```

Файл RehearsalPointDto.cs:

```
using System;
namespace RehearsalStudio.Application.DTOs;
public class RehearsalPointDto
{
    public int Id { get; set; }
    public float? Rating { get; set; }
    public string ContactNumber { get; set; } = string.Empty;
    public string Schedule { get; set; } = string.Empty;
    public string Name { get; set; } = string.Empty;
    public string Address { get; set; } = string.Empty;
}
```

Файл RoomDto.cs:

```
using System;
namespace RehearsalStudio.Application.DTOs;
public class RoomDto
```

```
{
    public int Id { get; set; }
    public string Name { get; set; } = string.Empty;
    public bool AirConditioner { get; set; }
    public int Price { get; set; }
    public bool RecordingSupport { get; set; }
    public int Area { get; set; }
    public int? IdRehearsalPoint { get; set; }
}
```

Файл ServiceBookingDto.cs:

```
using System;
namespace RehearsalStudio.Application.DTOs;
public class ServiceBookingDto
{
    public int IdService { get; set; }
    public int IdBooking { get; set; }
}
```

Файл ServiceDto.cs:

```
using System;
namespace RehearsalStudio.Application.DTOs;
public class ServiceDto
{
    public int Id { get; set; }
    public string Name { get; set; } = string.Empty;
    public int Price { get; set; }
    public string Type { get; set; } = string.Empty;
    public string? Requirements { get; set; }
    public int? IdRehearsalPoint { get; set; }
}
```

Файл StaffDto.cs:

```
using System;
namespace RehearsalStudio.Application.DTOs;
public class StaffDto
{
    public int Id { get; set; }
    public string FullName { get; set; } = string.Empty;
    public string? Address { get; set; }
    public int? Experience { get; set; }
    public string Phone { get; set; } = string.Empty;
    public int Age { get; set; }
    public int? IdRehearsalPoint { get; set; }
}
```

Файл UserDto.cs:

```
using System;
namespace RehearsalStudio.Application.DTOs;
public class UserDto
{
    public int Id { get; set; }
    public string FullName { get; set; } = string.Empty;
    public string Phone { get; set; } = string.Empty;
    public string Email { get; set; } = string.Empty;
    public DateTime RegistrationDate { get; set; }
}
```

Файл IBackupService.cs:

```
using System.Threading.Tasks;
namespace RehearsalStudio.Application.Interfaces;
public interface IBackupService
{
    Task<string> CreateDatabaseBackupAsync();
    Task<string> SaveQueryResultsToFileAsync(string sqlQuery, string fileFormat = "json");
}
```

Файл IBookingRepository.cs:

```
using RehearsalStudio.Domain.Entities;
namespace RehearsalStudio.Application.Interfaces;
public interface IBookingRepository
{
    Task<IEnumerable<Booking>> GetAllAsync();
    Task<Booking?> GetByIdAsync(int id);
    Task<IEnumerable<Booking>> GetFilteredAsync(string? status, int? idRoom, int? idUser);
    Task<Booking> AddAsync(Booking booking);
    Task UpdateAsync(Booking booking);
    Task DeleteAsync(int id);
}
```

Файл IBookingService.cs:

```
using RehearsalStudio.Application.DTOs;
using System.Threading.Tasks;
namespace RehearsalStudio.Application.Interfaces;
public interface IBookingService
{
    Task<IEnumerable<BookingDto>> GetAllAsync();
    Task<BookingDto?> GetByIdAsync(int id);
    Task<IEnumerable<BookingDto>> GetFilteredAsync(string? status, int? idRoom, int? idUser);
    Task<BookingDto> CreateAsync(BookingDto dto);
    Task UpdateAsync(int id, BookingDto dto);
    Task DeleteAsync(int id);
}
```

Файл IEquipmentBookingRepository.cs:

```
using RehearsalStudio.Domain.Entities;
namespace RehearsalStudio.Application.Interfaces;
public interface IEquipmentBookingRepository
{
    Task<IEnumerable<EquipmentBooking>> GetAllAsync();
    Task<EquipmentBooking?> GetByIdAsync(int idEquipment, int idBooking);
    Task<IEnumerable<EquipmentBooking>> GetFilteredAsync(int? idEquipment, int? idBooking);
    Task<EquipmentBooking> AddAsync(EquipmentBooking equipmentBooking);
    Task DeleteAsync(int idEquipment, int idBooking);
}
```

Файл IEquipmentBookingService.cs:

```
using RehearsalStudio.Application.DTOs;
using System.Threading.Tasks;
namespace RehearsalStudio.Application.Interfaces;
public interface IEquipmentBookingService
{
    Task<IEnumerable<EquipmentBookingDto>> GetAllAsync();
    Task<EquipmentBookingDto?> GetByIdAsync(int idEquipment, int idBooking);
    Task<IEnumerable<EquipmentBookingDto>> GetFilteredAsync(int? idEquipment, int? idBooking);
    Task<EquipmentBookingDto> CreateAsync(EquipmentBookingDto dto);
    Task DeleteAsync(int idEquipment, int idBooking);
}
```

Файл IEquipmentRepository.cs:

```
using RehearsalStudio.Domain.Entities;
namespace RehearsalStudio.Application.Interfaces;
public interface IEquipmentRepository
{
    Task<IEnumerable<Equipment>> GetAllAsync();
    Task<Equipment?> GetByIdAsync(int id);
    Task<IEnumerable<Equipment>> GetFilteredAsync(string? name, string? type, int? idRehearsalPoint);
    Task<Equipment> AddAsync(Equipment equipment);
    Task UpdateAsync(Equipment equipment);
    Task DeleteAsync(int id);
}
```

Файл IEquipmentService.cs:

```
using RehearsalStudio.Application.DTOs;
using System.Threading.Tasks;
namespace RehearsalStudio.Application.Interfaces;
public interface IEquipmentService
{
    Task<IEnumerable<EquipmentDto>> GetAllAsync();
    Task<EquipmentDto?> GetByIdAsync(int id);
    Task<IEnumerable<EquipmentDto>> GetFilteredAsync(string? name, string? type, int? idRehearsalPoint);
    Task<EquipmentDto> CreateAsync(EquipmentDto dto);
    Task UpdateAsync(int id, EquipmentDto dto);
    Task DeleteAsync(int id);
}
```

Файл IRehearsalPointRepository.cs:

```
using RehearsalStudio.Domain.Entities;
namespace RehearsalStudio.Application.Interfaces;
public interface IRehearsalPointRepository
{
    Task<IEnumerable<RehearsalPoint>> GetAllAsync();
    Task<RehearsalPoint?> GetByIdAsync(int id);
    Task<IEnumerable<RehearsalPoint>> GetFilteredAsync(string? name, float? minRating);
    Task<RehearsalPoint> AddAsync(RehearsalPoint rehearsalPoint);
    Task UpdateAsync(RehearsalPoint rehearsalPoint);
    Task DeleteAsync(int id);
}
```

Файл IRehearsalPointService.cs:

```
using RehearsalStudio.Application.DTOs;
using System.Threading.Tasks;
namespace RehearsalStudio.Application.Interfaces;
public interface IRehearsalPointService
{
    Task<IEnumerable<RehearsalPointDto>> GetAllAsync();
    Task<RehearsalPointDto?> GetByIdAsync(int id);
    Task<IEnumerable<RehearsalPointDto>> GetFilteredAsync(string? name, float? minRating);
    Task<RehearsalPointDto> CreateAsync(RehearsalPointDto dto);
    Task UpdateAsync(int id, RehearsalPointDto dto);
    Task DeleteAsync(int id);
}
```

Файл IRoomRepository.cs:

```
using RehearsalStudio.Domain.Entities;
namespace RehearsalStudio.Application.Interfaces;
public interface IRoomRepository
{
    Task<IEnumerable<Room>> GetAllAsync();
    Task<Room?> GetByIdAsync(int id);
    Task<IEnumerable<Room>> GetFilteredAsync(string? name, int? minPrice, int? idRehearsalPoint);
    Task<Room> AddAsync(Room room);
    Task UpdateAsync(Room room);
    Task DeleteAsync(int id);
}
```

Файл IRoomService.cs:

```
using RehearsalStudio.Application.DTOs;
using System.Threading.Tasks;
namespace RehearsalStudio.Application.Interfaces;
public interface IRoomService
{
    Task<IEnumerable<RoomDto>> GetAllAsync();
    Task<RoomDto?> GetByIdAsync(int id);
    Task<IEnumerable<RoomDto>> GetFilteredAsync(string? name, int? minPrice, int? idRehearsalPoint);
    Task<RoomDto> CreateAsync(RoomDto dto);
    Task UpdateAsync(int id, RoomDto dto);
    Task DeleteAsync(int id);
}
```

Файл IServiceBookingRepository.cs:

```
using RehearsalStudio.Domain.Entities;
namespace RehearsalStudio.Application.Interfaces;
public interface IServiceBookingRepository
{
    Task<IEnumerable<ServiceBooking>> GetAllAsync();
    Task<ServiceBooking?> GetByIdAsync(int idService, int idBooking);
    Task<IEnumerable<ServiceBooking>> GetFilteredAsync(int? idService, int? idBooking);
    Task<ServiceBooking> AddAsync(ServiceBooking serviceBooking);
    Task DeleteAsync(int idService, int idBooking);
}
```

Файл IServiceBookingService.cs:

```
using RehearsalStudio.Application.DTOs;
using System.Threading.Tasks;
namespace RehearsalStudio.Application.Interfaces;
public interface IServiceBookingService
{
    Task<IEnumerable<ServiceBookingDto>> GetAllAsync();
    Task<ServiceBookingDto?> GetByIdAsync(int idService, int idBooking);
    Task<IEnumerable<ServiceBookingDto>> GetFilteredAsync(int? idService, int? idBooking);
    Task<ServiceBookingDto> CreateAsync(ServiceBookingDto dto);
    Task DeleteAsync(int idService, int idBooking);
}
```

Файл IServiceRepository.cs:

```
using RehearsalStudio.Domain.Entities;
namespace RehearsalStudio.Application.Interfaces;
public interface IServiceRepository
{
    Task<IEnumerable<Service>> GetAllAsync();
    Task<Service?> GetByIdAsync(int id);
    Task<IEnumerable<Service>> GetFilteredAsync(string? name, string? type, int? idRehearsalPoint);
    Task<Service> AddAsync(Service service);
    Task UpdateAsync(Service service);
    Task DeleteAsync(int id);
}
```

Файл IServiceService.cs:

```
using RehearsalStudio.Application.DTOs;
using System.Threading.Tasks;
namespace RehearsalStudio.Application.Interfaces;
public interface IServiceService
{
    Task<IEnumerable<ServiceDto>> GetAllAsync();
    Task<ServiceDto?> GetByIdAsync(int id);
    Task<IEnumerable<ServiceDto>> GetFilteredAsync(string? name, string? type, int? idRehearsalPoint);
    Task<ServiceDto> CreateAsync(ServiceDto dto);
    Task UpdateAsync(int id, ServiceDto dto);
    Task DeleteAsync(int id);
}
```

Файл IStaffRepository.cs:

```
using RehearsalStudio.Domain.Entities;
namespace RehearsalStudio.Application.Interfaces;
public interface IStaffRepository
{
    Task<IEnumerable<Staff>> GetAllAsync();
    Task<Staff?> GetByIdAsync(int id);
    Task<IEnumerable<Staff>> GetFilteredAsync(string? fullName, int? minAge, int? idRehearsalPoint);
    Task<Staff> AddAsync(Staff staff);
    Task UpdateAsync(Staff staff);
    Task DeleteAsync(int id);
}
```

Файл IStaffService.cs:

```
using RehearsalStudio.Application.DTOs;
using System.Threading.Tasks;
namespace RehearsalStudio.Application.Interfaces;
public interface IStaffService
{
    Task<IEnumerable<StaffDto>> GetAllAsync();
    Task<StaffDto?> GetByIdAsync(int id);
    Task<IEnumerable<StaffDto>> GetFilteredAsync(string? fullName, int? minAge, int? idRehearsalPoint);
    Task<StaffDto> CreateAsync(StaffDto dto);
    Task UpdateAsync(int id, StaffDto dto);
    Task DeleteAsync(int id);
}
```

Файл IUserRepository.cs:

```
using RehearsalStudio.Domain.Entities;
namespace RehearsalStudio.Application.Interfaces;
public interface IUserRepository
{
    Task<IEnumerable<User>> GetAllAsync();
    Task<User?> GetByIdAsync(int id);
    Task<IEnumerable<User>> GetFilteredAsync(string? fullName, string? email);
    Task<User> AddAsync(User user);
    Task UpdateAsync(User user);
    Task DeleteAsync(int id);
}
```

Файл IUserService.cs:

```
using RehearsalStudio.Application.DTOs;
using System.Threading.Tasks;
namespace RehearsalStudio.Application.Interfaces;
public interface IUserService
{
    Task<IEnumerable<UserDto>> GetAllAsync();
    Task<UserDto?> GetByIdAsync(int id);
    Task<IEnumerable<UserDto>> GetFilteredAsync(string? fullName, string? email);
    Task<UserDto> CreateAsync(UserDto dto);
    Task UpdateAsync(int id, UserDto dto);
    Task DeleteAsync(int id);
}
```

Файл 20250910175704_InitialCreate.cs:

```
using System;
using Microsoft.EntityFrameworkCore.Migrations;
using Npgsql.EntityFrameworkCore.PostgreSQL.Metadata;
#nullable disable
namespace RehearsalStudio.Application.Migrations
{
    /// <inheritdoc />
    public partial class InitialCreate : Migration
    {
        /// <inheritdoc />
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.EnsureSchema(
                name: "main");
            migrationBuilder.CreateTable(
                name: "rehearsal_points",
                schema: "main",
                columns: table => new
                {
                    Id = table.Column<int>(type: "integer", nullable: false)
.Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),
                    Rating = table.Column<float>(type: "real", nullable: true),
                    ContactNumber = table.Column<string>(type: "text", nullable: false),
                    Schedule = table.Column<string>(type: "text", nullable: false),
                    Name = table.Column<string>(type: "text", nullable: false),
                    Address = table.Column<string>(type: "text", nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_rehearsal_points", x => x.Id);
                });
            migrationBuilder.CreateTable(
                name: "users",
                schema: "main",
                columns: table => new
                {
                    Id = table.Column<int>(type: "integer", nullable: false)
                        .Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),
                    FullName = table.Column<string>(type: "text", nullable: false),
                    Phone = table.Column<string>(type: "text", nullable: false),
                    Email = table.Column<string>(type: "text", nullable: false),
                    RegistrationDate = table.Column<DateTime>(type: "timestamp with time zone", nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_users", x => x.Id);
                });
            migrationBuilder.CreateTable(
                name: "equipment",
                schema: "main",
                columns: table => new
                {
                    Id = table.Column<int>(type: "integer", nullable: false)
                        .Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),
                    Name = table.Column<string>(type: "text", nullable: false),
                    Type = table.Column<string>(type: "text", nullable: false),
                    Brand = table.Column<string>(type: "text", nullable: false),
                    Model = table.Column<string>(type: "text", nullable: false),
                    Condition = table.Column<string>(type: "text", nullable: false),
                    IdRehearsalPoint = table.Column<int>(type: "integer", nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_equipment", x => x.Id);
                    table.ForeignKey(
                        name: "FK_equipment_rehearsal_points_IdRehearsalPoint",
                        column: x => x.IdRehearsalPoint,
                        principalSchema: "main",
                        principalTable: "rehearsal_points",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });
            migrationBuilder.CreateTable(
                name: "rooms",
                schema: "main",
                columns: table => new
                {
                    Id = table.Column<int>(type: "integer", nullable: false)
                        .Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),
                    Name = table.Column<string>(type: "text", nullable: false),
                    AirConditioner = table.Column<bool>(type: "boolean", nullable: false),
                    Price = table.Column<int>(type: "integer", nullable: false),
                    RecordingSupport = table.Column<bool>(type: "boolean", nullable: false),
                    Area = table.Column<int>(type: "integer", nullable: false),
                    IdRehearsalPoint = table.Column<int>(type: "integer", nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_rooms", x => x.Id);
                    table.ForeignKey(
                        name: "FK_rooms_rehearsal_points_IdRehearsalPoint",
                        column: x => x.IdRehearsalPoint,
                        principalSchema: "main",
                        principalTable: "rehearsal_points",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });
            migrationBuilder.CreateTable(
                name: "service",
                schema: "main",
                columns: table => new
                {
                    Id = table.Column<int>(type: "integer", nullable: false)
                        .Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),
                    Name = table.Column<string>(type: "text", nullable: false),
                    Price = table.Column<int>(type: "integer", nullable: false),
                    Type = table.Column<string>(type: "text", nullable: false),
                    Requirements = table.Column<string>(type: "text", nullable: true),
                    IdRehearsalPoint = table.Column<int>(type: "integer", nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_service", x => x.Id);
                    table.ForeignKey(
                        name: "FK_service_rehearsal_points_IdRehearsalPoint",
                        column: x => x.IdRehearsalPoint,
                        principalSchema: "main",
                        principalTable: "rehearsal_points",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });
            migrationBuilder.CreateTable(
                name: "staff",
                schema: "main",
                columns: table => new
                {
                    Id = table.Column<int>(type: "integer", nullable: false)
                        .Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),
                    FullName = table.Column<string>(type: "text", nullable: false),
                    Address = table.Column<string>(type: "text", nullable: true),
                    Experience = table.Column<int>(type: "integer", nullable: true),
                    Phone = table.Column<string>(type: "text", nullable: false),
                    Age = table.Column<int>(type: "integer", nullable: false),
                    IdRehearsalPoint = table.Column<int>(type: "integer", nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_staff", x => x.Id);
                    table.ForeignKey(
                        name: "FK_staff_rehearsal_points_IdRehearsalPoint",
                        column: x => x.IdRehearsalPoint,
                        principalSchema: "main",
                        principalTable: "rehearsal_points",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });
            migrationBuilder.CreateTable(
                name: "booking",
                schema: "main",
                columns: table => new
                {
                    Id = table.Column<int>(type: "integer", nullable: false)
                        .Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),
                    Time = table.Column<DateTime>(type: "timestamp with time zone", nullable: false),
                    Duration = table.Column<int>(type: "integer", nullable: true),
                    Cost = table.Column<int>(type: "integer", nullable: false),
                    CreationDate = table.Column<DateTime>(type: "timestamp with time zone", nullable: false),
                    Status = table.Column<string>(type: "text", nullable: false),
                    NumberOfPeople = table.Column<int>(type: "integer", nullable: false),
                    IdRoom = table.Column<int>(type: "integer", nullable: true),
                    IdUser = table.Column<int>(type: "integer", nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_booking", x => x.Id);
                    table.ForeignKey(
                        name: "FK_booking_rooms_IdRoom",
                        column: x => x.IdRoom,
                        principalSchema: "main",
                        principalTable: "rooms",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.SetNull);
                    table.ForeignKey(
                        name: "FK_booking_users_IdUser",
                        column: x => x.IdUser,
                        principalSchema: "main",
                        principalTable: "users",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });
            migrationBuilder.CreateTable(
                name: "equipment_booking",
                schema: "main",
                columns: table => new
                {
                    IdEquipment = table.Column<int>(type: "integer", nullable: false),
                    IdBooking = table.Column<int>(type: "integer", nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_equipment_booking", x => new { x.IdEquipment, x.IdBooking });
                    table.ForeignKey(
                        name: "FK_equipment_booking_booking_IdBooking",
                        column: x => x.IdBooking,
                        principalSchema: "main",
                        principalTable: "booking",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                    table.ForeignKey(
                        name: "FK_equipment_booking_equipment_IdEquipment",
                        column: x => x.IdEquipment,
                        principalSchema: "main",
                        principalTable: "equipment",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });
            migrationBuilder.CreateTable(
                name: "service_booking",
                schema: "main",
                columns: table => new
                {
                    IdService = table.Column<int>(type: "integer", nullable: false),
                    IdBooking = table.Column<int>(type: "integer", nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_service_booking", x => new { x.IdService, x.IdBooking });
                    table.ForeignKey(
                        name: "FK_service_booking_booking_IdBooking",
                        column: x => x.IdBooking,
                        principalSchema: "main",
                        principalTable: "booking",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                    table.ForeignKey(
                        name: "FK_service_booking_service_IdService",
                        column: x => x.IdService,
                        principalSchema: "main",
                        principalTable: "service",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });
            migrationBuilder.CreateIndex(
                name: "IX_booking_IdRoom",
                schema: "main",
                table: "booking",
                column: "IdRoom");
            migrationBuilder.CreateIndex(
                name: "IX_booking_IdUser",
                schema: "main",
                table: "booking",
                column: "IdUser");
            migrationBuilder.CreateIndex(
                name: "IX_equipment_IdRehearsalPoint",
                schema: "main",
                table: "equipment",
                column: "IdRehearsalPoint");
            migrationBuilder.CreateIndex(
                name: "IX_equipment_booking_IdBooking",
                schema: "main",
                table: "equipment_booking",
                column: "IdBooking");
            migrationBuilder.CreateIndex(
                name: "IX_rooms_IdRehearsalPoint",
                schema: "main",
                table: "rooms",
                column: "IdRehearsalPoint");
            migrationBuilder.CreateIndex(
                name: "IX_service_IdRehearsalPoint",
                schema: "main",
                table: "service",
                column: "IdRehearsalPoint");
            migrationBuilder.CreateIndex(
                name: "IX_service_booking_IdBooking",
                schema: "main",
                table: "service_booking",
                column: "IdBooking");
            migrationBuilder.CreateIndex(
                name: "IX_staff_IdRehearsalPoint",
                schema: "main",
                table: "staff",
                column: "IdRehearsalPoint");
        }
        /// <inheritdoc />
        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropTable(
                name: "equipment_booking",
                schema: "main");
            migrationBuilder.DropTable(
                name: "service_booking",
                schema: "main");
            migrationBuilder.DropTable(
                name: "staff",
                schema: "main");
            migrationBuilder.DropTable(
                name: "equipment",
                schema: "main");
            migrationBuilder.DropTable(
                name: "booking",
                schema: "main");
            migrationBuilder.DropTable(
                name: "service",
                schema: "main");
            migrationBuilder.DropTable(
                name: "rooms",
                schema: "main");
            migrationBuilder.DropTable(
                name: "users",
                schema: "main");
            migrationBuilder.DropTable(
                name: "rehearsal_points",
                schema: "main");
        }
    }
}
```

Файл 20250910175704_InitialCreate.Designer.cs:

```
// <auto-generated />
using System;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Infrastructure;
using Microsoft.EntityFrameworkCore.Migrations;
using Microsoft.EntityFrameworkCore.Storage.ValueConversion;
using Npgsql.EntityFrameworkCore.PostgreSQL.Metadata;
using RehearsalStudio.Infrastructure.Data;
#nullable disable
namespace RehearsalStudio.Application.Migrations
{
    [DbContext(typeof(RehearsalStudioDbContext))]
    [Migration("20250910175704_InitialCreate")]
    partial class InitialCreate
    {
        /// <inheritdoc />
        protected override void BuildTargetModel(ModelBuilder modelBuilder)
        {
#pragma warning disable 612, 618
            modelBuilder
                .HasDefaultSchema("main")
                .HasAnnotation("ProductVersion", "9.0.9")
                .HasAnnotation("Relational:MaxIdentifierLength", 63);
            NpgsqlModelBuilderExtensions.UseIdentityByDefaultColumns(modelBuilder);
            modelBuilder.Entity("RehearsalStudio.Domain.Entities.Booking", b =>
                {
                    b.Property<int>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("integer");
                    NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));
                    b.Property<int>("Cost")
                        .HasColumnType("integer");
                    b.Property<DateTime>("CreationDate")
                        .HasColumnType("timestamp with time zone");
                    b.Property<int?>("Duration")
                        .HasColumnType("integer");
                    b.Property<int?>("IdRoom")
                        .HasColumnType("integer");
                    b.Property<int?>("IdUser")
                        .HasColumnType("integer");
                    b.Property<int>("NumberOfPeople")
                        .HasColumnType("integer");
                    b.Property<string>("Status")
                        .IsRequired()
                        .HasColumnType("text");
                    b.Property<DateTime>("Time")
                        .HasColumnType("timestamp with time zone");
                    b.HasKey("Id");
                    b.HasIndex("IdRoom");
                    b.HasIndex("IdUser");
                    b.ToTable("booking", "main");
                });
            modelBuilder.Entity("RehearsalStudio.Domain.Entities.Equipment", b =>
                {
                    b.Property<int>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("integer");
                    NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));
                    b.Property<string>("Brand")
                        .IsRequired()
                        .HasColumnType("text");
                    b.Property<string>("Condition")
                        .IsRequired()
                        .HasColumnType("text");
                    b.Property<int?>("IdRehearsalPoint")
                        .HasColumnType("integer");
                    b.Property<string>("Model")
                        .IsRequired()
                        .HasColumnType("text");
                    b.Property<string>("Name")
                        .IsRequired()
                        .HasColumnType("text");
                    b.Property<string>("Type")
                        .IsRequired()
                        .HasColumnType("text");
                    b.HasKey("Id");
                    b.HasIndex("IdRehearsalPoint");
                    b.ToTable("equipment", "main");
                });
                modelBuilder.Entity("RehearsalStudio.Domain.Entities.EquipmentBooking", b =>
                {
                    b.Property<int>("IdEquipment")
                        .HasColumnType("integer")
                        .HasColumnOrder(0);
                    b.Property<int>("IdBooking")
                        .HasColumnType("integer")
                        .HasColumnOrder(1);
                    b.HasKey("IdEquipment", "IdBooking");
                    b.HasIndex("IdBooking");
                    b.ToTable("equipment_booking", "main");
                });
                modelBuilder.Entity("RehearsalStudio.Domain.Entities.RehearsalPoint", b =>
                {
                    b.Property<int>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("integer");
                    NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));
                    b.Property<string>("Address")
                        .IsRequired()
                        .HasColumnType("text");
                    b.Property<string>("ContactNumber")
                        .IsRequired()
                        .HasColumnType("text");
                    b.Property<string>("Name")
                        .IsRequired()
                        .HasColumnType("text");
                    b.Property<float?>("Rating")
                        .HasColumnType("real");
                    b.Property<string>("Schedule")
                        .IsRequired()
                        .HasColumnType("text");
                    b.HasKey("Id");
                    b.ToTable("rehearsal_points", "main");
                });
                modelBuilder.Entity("RehearsalStudio.Domain.Entities.Room", b =>
                {
                    b.Property<int>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("integer");
                    NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));
                    b.Property<bool>("AirConditioner")
                        .HasColumnType("boolean");
                    b.Property<int>("Area")
                        .HasColumnType("integer");
                    b.Property<int?>("IdRehearsalPoint")
                        .HasColumnType("integer");
                    b.Property<string>("Name")
                        .IsRequired()
                        .HasColumnType("text");
                    b.Property<int>("Price")
                        .HasColumnType("integer");
                    b.Property<bool>("RecordingSupport")
                        .HasColumnType("boolean");
                    b.HasKey("Id");
                    b.HasIndex("IdRehearsalPoint");
                    b.ToTable("rooms", "main");
                });
                modelBuilder.Entity("RehearsalStudio.Domain.Entities.Service", b =>
                {
                    b.Property<int>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("integer");
                    NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));
                    b.Property<int?>("IdRehearsalPoint")
                        .HasColumnType("integer");
                    b.Property<string>("Name")
                        .IsRequired()
                        .HasColumnType("text");
                    b.Property<int>("Price")
                        .HasColumnType("integer");
                    b.Property<string>("Requirements")
                        .HasColumnType("text");
                    b.Property<string>("Type")
                        .IsRequired()
                        .HasColumnType("text");
                    b.HasKey("Id");
                    b.HasIndex("IdRehearsalPoint");
                    b.ToTable("service", "main");
                });
                modelBuilder.Entity("RehearsalStudio.Domain.Entities.ServiceBooking", b =>
                {
                    b.Property<int>("IdService")
                        .HasColumnType("integer")
                        .HasColumnOrder(0);
                    b.Property<int>("IdBooking")
                        .HasColumnType("integer")
                        .HasColumnOrder(1);
                    b.HasKey("IdService", "IdBooking");
                    b.HasIndex("IdBooking");
                    b.ToTable("service_booking", "main");
                });
                modelBuilder.Entity("RehearsalStudio.Domain.Entities.Staff", b =>
                {
                    b.Property<int>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("integer");
                    NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));
                    b.Property<string>("Address")
                        .HasColumnType("text");
                    b.Property<int>("Age")
```

```
                    .HasColumnType("integer");
                b.Property<int?>("Experience")
                    .HasColumnType("integer");
                b.Property<string>("FullName")
                    .IsRequired()
                    .HasColumnType("text");
                b.Property<int?>("IdRehearsalPoint")
                    .HasColumnType("integer");
                b.Property<string>("Phone")
                    .IsRequired()
                    .HasColumnType("text");
                b.HasKey("Id");
                b.HasIndex("IdRehearsalPoint");
                b.ToTable("staff", "main");
            });
            modelBuilder.Entity("RehearsalStudio.Domain.Entities.User", b =>
            {
                b.Property<int>("Id")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("integer");
                NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));
                b.Property<string>("Email")
                    .IsRequired()
                    .HasColumnType("text");
                b.Property<string>("FullName")
                    .IsRequired()
                    .HasColumnType("text");
                b.Property<string>("Phone")
                    .IsRequired()
                    .HasColumnType("text");
                b.Property<DateTime>("RegistrationDate")
                    .HasColumnType("timestamp with time zone");
                b.HasKey("Id");
                b.ToTable("users", "main");
            });
            modelBuilder.Entity("RehearsalStudio.Domain.Entities.Booking", b =>
            {
                b.HasOne("RehearsalStudio.Domain.Entities.Room", "Room")
                    .WithMany("Bookings")
                    .HasForeignKey("IdRoom")
                    .OnDelete(DeleteBehavior.SetNull);
                b.HasOne("RehearsalStudio.Domain.Entities.User", "User")
                    .WithMany("Bookings")
                    .HasForeignKey("IdUser")
                    .OnDelete(DeleteBehavior.Cascade);
                b.Navigation("Room");
                b.Navigation("User");
            });
            modelBuilder.Entity("RehearsalStudio.Domain.Entities.Equipment", b =>
            {
                b.HasOne("RehearsalStudio.Domain.Entities.RehearsalPoint", "RehearsalPoint")
                    .WithMany("Equipment")
                    .HasForeignKey("IdRehearsalPoint")
                    .OnDelete(DeleteBehavior.Cascade);
                b.Navigation("RehearsalPoint");
            });
        modelBuilder.Entity("RehearsalStudio.Domain.Entities.EquipmentBooking", b =>
            {
                b.HasOne("RehearsalStudio.Domain.Entities.Booking", "Booking")
                    .WithMany("EquipmentBookings")
                    .HasForeignKey("IdBooking")
                    .OnDelete(DeleteBehavior.Cascade)
                    .IsRequired();
                b.HasOne("RehearsalStudio.Domain.Entities.Equipment", "Equipment")
                    .WithMany("EquipmentBookings")
                    .HasForeignKey("IdEquipment")
                    .OnDelete(DeleteBehavior.Cascade)
                    .IsRequired();
                b.Navigation("Booking");
                b.Navigation("Equipment");
            });
        modelBuilder.Entity("RehearsalStudio.Domain.Entities.Room", b =>
            {
                b.HasOne("RehearsalStudio.Domain.Entities.RehearsalPoint", "RehearsalPoint")
                    .WithMany("Rooms")
                    .HasForeignKey("IdRehearsalPoint")
                    .OnDelete(DeleteBehavior.Cascade);
                b.Navigation("RehearsalPoint");
            });
        modelBuilder.Entity("RehearsalStudio.Domain.Entities.Service", b =>
            {
                b.HasOne("RehearsalStudio.Domain.Entities.RehearsalPoint", "RehearsalPoint")
                    .WithMany("Services")
                    .HasForeignKey("IdRehearsalPoint")
                    .OnDelete(DeleteBehavior.Cascade);
                b.Navigation("RehearsalPoint");
            });
        modelBuilder.Entity("RehearsalStudio.Domain.Entities.ServiceBooking", b =>
            {
                b.HasOne("RehearsalStudio.Domain.Entities.Booking", "Booking")
                    .WithMany("ServiceBookings")
                    .HasForeignKey("IdBooking")
                    .OnDelete(DeleteBehavior.Cascade)
                    .IsRequired();
                b.HasOne("RehearsalStudio.Domain.Entities.Service", "Service")
                    .WithMany("ServiceBookings")
                    .HasForeignKey("IdService")
                    .OnDelete(DeleteBehavior.Cascade)
                    .IsRequired();
                b.Navigation("Booking");
                b.Navigation("Service");
            });
        modelBuilder.Entity("RehearsalStudio.Domain.Entities.Staff", b =>
            {
                b.HasOne("RehearsalStudio.Domain.Entities.RehearsalPoint", "RehearsalPoint")
                    .WithMany("Staff")
                    .HasForeignKey("IdRehearsalPoint")
                    .OnDelete(DeleteBehavior.Cascade);
                b.Navigation("RehearsalPoint");
            });
        modelBuilder.Entity("RehearsalStudio.Domain.Entities.Booking", b =>
            {
                b.Navigation("EquipmentBookings");
                b.Navigation("ServiceBookings");
            });
        modelBuilder.Entity("RehearsalStudio.Domain.Entities.Equipment", b =>
            {
                b.Navigation("EquipmentBookings");
            });
        modelBuilder.Entity("RehearsalStudio.Domain.Entities.RehearsalPoint", b =>
            {
                b.Navigation("Equipment");
                b.Navigation("Rooms");
                b.Navigation("Services");
                b.Navigation("Staff");
            });
        modelBuilder.Entity("RehearsalStudio.Domain.Entities.Room", b =>
            {
                b.Navigation("Bookings");
            });
        modelBuilder.Entity("RehearsalStudio.Domain.Entities.Service", b =>
            {
                b.Navigation("ServiceBookings");
            });
        modelBuilder.Entity("RehearsalStudio.Domain.Entities.User", b =>
            {
                b.Navigation("Bookings");
            });
#pragma warning restore 612, 618
    }
}
}
```

Файл RehearsalStudioDbContextModelSnapshot.cs:

```
// <auto-generated />
using System;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Infrastructure;
using Microsoft.EntityFrameworkCore.Storage.ValueConversion;
using Npgsql.EntityFrameworkCore.PostgreSQL.Metadata;
using RehearsalStudio.Infrastructure.Data;
#nullable disable
namespace RehearsalStudio.Application.Migrations
{
    [DbContext(typeof(RehearsalStudioDbContext))]
    partial class RehearsalStudioDbContextModelSnapshot : ModelSnapshot
    {
        protected override void BuildModel(ModelBuilder modelBuilder)
        {
#pragma warning disable 612, 618
```

```
            modelBuilder
                .HasDefaultSchema("main")
                .HasAnnotation("ProductVersion", "9.0.9")
                .HasAnnotation("Relational:MaxIdentifierLength", 63);
            NpgsqlModelBuilderExtensions.UseIdentityByDefaultColumns(modelBuilder);
            modelBuilder.Entity("RehearsalStudio.Domain.Entities.Booking", b =>
                {
                    b.Property<int>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("integer");
                    NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));
                    b.Property<int>("Cost")
                        .HasColumnType("integer");
                    b.Property<DateTime>("CreationDate")
                        .HasColumnType("timestamp with time zone");
                    b.Property<int?>("Duration")
                        .HasColumnType("integer");
                    b.Property<int?>("IdRoom")
                        .HasColumnType("integer");
                    b.Property<int?>("IdUser")
                        .HasColumnType("integer");
                    b.Property<int>("NumberOfPeople")
                        .HasColumnType("integer");
                    b.Property<string>("Status")
                        .IsRequired()
                        .HasColumnType("text");
                    b.Property<DateTime>("Time")
                        .HasColumnType("timestamp with time zone");
                    b.HasKey("Id");
                    b.HasIndex("IdRoom");
                    b.HasIndex("IdUser");
                    b.ToTable("booking", "main");
                });
            modelBuilder.Entity("RehearsalStudio.Domain.Entities.Equipment", b =>
                {
                    b.Property<int>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("integer");
                    NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));
                    b.Property<string>("Brand")
                        .IsRequired()
                        .HasColumnType("text");
                    b.Property<string>("Condition")
                        .IsRequired()
                        .HasColumnType("text");
                    b.Property<int?>("IdRehearsalPoint")
                        .HasColumnType("integer");
                    b.Property<string>("Model")
                        .IsRequired()
                        .HasColumnType("text");
                    b.Property<string>("Name")
                        .IsRequired()
                        .HasColumnType("text");
                    b.Property<string>("Type")
                        .IsRequired()
                        .HasColumnType("text");
                    b.HasKey("Id");
                    b.HasIndex("IdRehearsalPoint");
                    b.ToTable("equipment", "main");
                });
                modelBuilder.Entity("RehearsalStudio.Domain.Entities.EquipmentBooking", b =>
                {
                    b.Property<int>("IdEquipment")
                        .HasColumnType("integer")
                        .HasColumnOrder(0);
                    b.Property<int>("IdBooking")
                        .HasColumnType("integer")
                        .HasColumnOrder(1);
                    b.HasKey("IdEquipment", "IdBooking");
                    b.HasIndex("IdBooking");
                    b.ToTable("equipment_booking", "main");
                });
                modelBuilder.Entity("RehearsalStudio.Domain.Entities.RehearsalPoint", b =>
                {
                    b.Property<int>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("integer");
                    NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));
                    b.Property<string>("Address")
                        .IsRequired()
                        .HasColumnType("text");
                    b.Property<string>("ContactNumber")
                        .IsRequired()
                        .HasColumnType("text");
                    b.Property<string>("Name")
                        .IsRequired()
                        .HasColumnType("text");
                    b.Property<float?>("Rating")
                        .HasColumnType("real");
                    b.Property<string>("Schedule")
                        .IsRequired()
                        .HasColumnType("text");
                    b.HasKey("Id");
                    b.ToTable("rehearsal_points", "main");
                });
                modelBuilder.Entity("RehearsalStudio.Domain.Entities.Room", b =>
                {
                    b.Property<int>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("integer");
                    NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));
                    b.Property<bool>("AirConditioner")
                        .HasColumnType("boolean");
                    b.Property<int>("Area")
                        .HasColumnType("integer");
                    b.Property<int?>("IdRehearsalPoint")
                        .HasColumnType("integer");
                    b.Property<string>("Name")
                        .IsRequired()
                        .HasColumnType("text");
                    b.Property<int>("Price")
```

```
                    .HasColumnType("integer");
                b.Property<bool>("RecordingSupport")
                    .HasColumnType("boolean");
                b.HasKey("Id");
                b.HasIndex("IdRehearsalPoint");
                b.ToTable("rooms", "main");
            });
            modelBuilder.Entity("RehearsalStudio.Domain.Entities.Service", b =>
            {
                b.Property<int>("Id")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("integer");
                NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));
                b.Property<int?>("IdRehearsalPoint")
                    .HasColumnType("integer");
                b.Property<string>("Name")
                    .IsRequired()
                    .HasColumnType("text");
                b.Property<int>("Price")
                    .HasColumnType("integer");
                b.Property<string>("Requirements")
                    .HasColumnType("text");
                b.Property<string>("Type")
                    .IsRequired()
                    .HasColumnType("text");
                b.HasKey("Id");
                b.HasIndex("IdRehearsalPoint");
                b.ToTable("service", "main");
            });
            modelBuilder.Entity("RehearsalStudio.Domain.Entities.ServiceBooking", b =>
            {
                b.Property<int>("IdService")
                    .HasColumnType("integer")
                    .HasColumnOrder(0);
                b.Property<int>("IdBooking")
                    .HasColumnType("integer")
                    .HasColumnOrder(1);
                b.HasKey("IdService", "IdBooking");
                b.HasIndex("IdBooking");
                b.ToTable("service_booking", "main");
            });
            modelBuilder.Entity("RehearsalStudio.Domain.Entities.Staff", b =>
            {
                b.Property<int>("Id")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("integer");
                NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));
                b.Property<string>("Address")
                    .HasColumnType("text");
                b.Property<int>("Age")
                    .HasColumnType("integer");
                b.Property<int?>("Experience")
                    .HasColumnType("integer");
                b.Property<string>("FullName")
                    .IsRequired()
                    .HasColumnType("text");
                b.Property<int?>("IdRehearsalPoint")
                    .HasColumnType("integer");
                b.Property<string>("Phone")
                    .IsRequired()
                    .HasColumnType("text");
                b.HasKey("Id");
                b.HasIndex("IdRehearsalPoint");
                b.ToTable("staff", "main");
            });
            modelBuilder.Entity("RehearsalStudio.Domain.Entities.User", b =>
            {
                b.Property<int>("Id")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("integer");
                NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));
                b.Property<string>("Email")
                    .IsRequired()
                    .HasColumnType("text");
                b.Property<string>("FullName")
                    .IsRequired()
                    .HasColumnType("text");
                b.Property<string>("Phone")
                    .IsRequired()
                    .HasColumnType("text");
                b.Property<DateTime>("RegistrationDate")
                    .HasColumnType("timestamp with time zone");
                b.HasKey("Id");
                b.ToTable("users", "main");
            });
            modelBuilder.Entity("RehearsalStudio.Domain.Entities.Booking", b =>
            {
                b.HasOne("RehearsalStudio.Domain.Entities.Room", "Room")
                    .WithMany("Bookings")
                    .HasForeignKey("IdRoom")
                    .OnDelete(DeleteBehavior.SetNull);
                b.HasOne("RehearsalStudio.Domain.Entities.User", "User")
                    .WithMany("Bookings")
                    .HasForeignKey("IdUser")
                    .OnDelete(DeleteBehavior.Cascade);
                b.Navigation("Room");
                b.Navigation("User");
            });
            modelBuilder.Entity("RehearsalStudio.Domain.Entities.Equipment", b =>
            {
                b.HasOne("RehearsalStudio.Domain.Entities.RehearsalPoint", "RehearsalPoint")
                    .WithMany("Equipment")
                    .HasForeignKey("IdRehearsalPoint")
                    .OnDelete(DeleteBehavior.Cascade);
                b.Navigation("RehearsalPoint");
            });
            modelBuilder.Entity("RehearsalStudio.Domain.Entities.EquipmentBooking", b =>
            {
                b.HasOne("RehearsalStudio.Domain.Entities.Booking", "Booking")
```

```
                        .WithMany("EquipmentBookings")
                        .HasForeignKey("IdBooking")
                        .OnDelete(DeleteBehavior.Cascade)
                        .IsRequired();
                    b.HasOne("RehearsalStudio.Domain.Entities.Equipment", "Equipment")
                        .WithMany("EquipmentBookings")
                        .HasForeignKey("IdEquipment")
                        .OnDelete(DeleteBehavior.Cascade)
                        .IsRequired();
                    b.Navigation("Booking");
                    b.Navigation("Equipment");
                });
            modelBuilder.Entity("RehearsalStudio.Domain.Entities.Room", b =>
                {
                    b.HasOne("RehearsalStudio.Domain.Entities.RehearsalPoint", "RehearsalPoint")
                        .WithMany("Rooms")
                        .HasForeignKey("IdRehearsalPoint")
                        .OnDelete(DeleteBehavior.Cascade);
                    b.Navigation("RehearsalPoint");
                });
            modelBuilder.Entity("RehearsalStudio.Domain.Entities.Service", b =>
                {
                    b.HasOne("RehearsalStudio.Domain.Entities.RehearsalPoint", "RehearsalPoint")
                        .WithMany("Services")
                        .HasForeignKey("IdRehearsalPoint")
                        .OnDelete(DeleteBehavior.Cascade);
                    b.Navigation("RehearsalPoint");
                });
            modelBuilder.Entity("RehearsalStudio.Domain.Entities.ServiceBooking", b =>
                {
                    b.HasOne("RehearsalStudio.Domain.Entities.Booking", "Booking")
                        .WithMany("ServiceBookings")
                        .HasForeignKey("IdBooking")
                        .OnDelete(DeleteBehavior.Cascade)
                        .IsRequired();
                    b.HasOne("RehearsalStudio.Domain.Entities.Service", "Service")
                        .WithMany("ServiceBookings")
                        .HasForeignKey("IdService")
                        .OnDelete(DeleteBehavior.Cascade)
                        .IsRequired();
                    b.Navigation("Booking");
                    b.Navigation("Service");
                });
            modelBuilder.Entity("RehearsalStudio.Domain.Entities.Staff", b =>
                {
                    b.HasOne("RehearsalStudio.Domain.Entities.RehearsalPoint", "RehearsalPoint")
                        .WithMany("Staff")
                        .HasForeignKey("IdRehearsalPoint")
                        .OnDelete(DeleteBehavior.Cascade);
                    b.Navigation("RehearsalPoint");
                });
            modelBuilder.Entity("RehearsalStudio.Domain.Entities.Booking", b =>
        {
          b.Navigation("EquipmentBookings");
          b.Navigation("ServiceBookings");
        });
        modelBuilder.Entity("RehearsalStudio.Domain.Entities.Equipment", b =>
        {
          b.Navigation("EquipmentBookings");
        });
        modelBuilder.Entity("RehearsalStudio.Domain.Entities.RehearsalPoint", b =>
        {
          b.Navigation("Equipment");
          b.Navigation("Rooms");
          b.Navigation("Services");
          b.Navigation("Staff");
        });
        modelBuilder.Entity("RehearsalStudio.Domain.Entities.Room", b =>
        {
          b.Navigation("Bookings");
        });
        modelBuilder.Entity("RehearsalStudio.Domain.Entities.Service", b =>
        {
          b.Navigation("ServiceBookings");
        });
        modelBuilder.Entity("RehearsalStudio.Domain.Entities.User", b =>
        {
          b.Navigation("Bookings");
        });
#pragma warning restore 612, 618
    }
  }
}
```

Файл .NETCoreApp,Version=v9.0.AssemblyAttributes.cs:

```
// <autogenerated />
using System;
using System.Reflection;
[assembly: global::System.Runtime.Versioning.TargetFrameworkAttribute(".NETCoreApp,Version=v9.0", FrameworkDisplayName = ".NET 9.0")]
```

Файл RehearsalStudio.Application.AssemblyInfo.cs:

```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------
using System;
using System.Reflection;
[assembly: System.Reflection.AssemblyCompanyAttribute("RehearsalStudio.Application")]
```

```
[assembly: System.Reflection.AssemblyConfigurationAttribute("Debug")]
[assembly: System.Reflection.AssemblyFileVersionAttribute("1.0.0.0")]
[assembly: System.Reflection.AssemblyInformationalVersionAttribute("1.0.0")]
[assembly: System.Reflection.AssemblyProductAttribute("RehearsalStudio.Application")]
[assembly: System.Reflection.AssemblyTitleAttribute("RehearsalStudio.Application")]
[assembly: System.Reflection.AssemblyVersionAttribute("1.0.0.0")]
// Generated by the MSBuild WriteCodeFragment class.
```

Файл RehearsalStudio.Application.GlobalUsings.g.cs:

```
// <auto-generated/>
global using global::System;
global using global::System.Collections.Generic;
global using global::System.IO;
global using global::System.Linq;
global using global::System.Net.Http;
global using global::System.Threading;
global using global::System.Threading.Tasks;
```

Файл BookingRepository.cs:

```
using Microsoft.EntityFrameworkCore;
using RehearsalStudio.Application.Interfaces;
using RehearsalStudio.Domain.Entities;
using RehearsalStudio.Infrastructure.Data;
namespace RehearsalStudio.Infrastructure.Repositories;
public class BookingRepository : IBookingRepository
{
    private readonly RehearsalStudioDbContext _context;
    public BookingRepository(RehearsalStudioDbContext context)
    {
        _context = context;
    }
    public async Task<IEnumerable<Booking>> GetAllAsync()
    {
        return await _context.Bookings.ToListAsync();
    }
    public async Task<Booking?> GetByIdAsync(int id)
    {
        return await _context.Bookings.FindAsync(id);
    }
    public async Task<IEnumerable<Booking>> GetFilteredAsync(string? status, int? idRoom, int? idUser)
    {
        var query = _context.Bookings.AsQueryable();
        if (!string.IsNullOrEmpty(status))
            query = query.Where(b => b.Status.Contains(status));
        if (idRoom.HasValue)
            query = query.Where(b => b.IdRoom == idRoom.Value);
        if (idUser.HasValue)
            query = query.Where(b => b.IdUser == idUser.Value);
        return await query.ToListAsync();
    }
    public async Task<Booking> AddAsync(Booking booking)
    {
        _context.Bookings.Add(booking);
        await _context.SaveChangesAsync();
        return booking;
    }
    public async Task UpdateAsync(Booking booking)
    {
        _context.Bookings.Update(booking);
        await _context.SaveChangesAsync();
    }
    public async Task DeleteAsync(int id)
    {
        var booking = await _context.Bookings.FindAsync(id);
        if (booking != null)
        {
            _context.Bookings.Remove(booking);
            await _context.SaveChangesAsync();
        }
    }
}
```

Файл EquipmentBookingRepository.cs:

```
using Microsoft.EntityFrameworkCore;
using RehearsalStudio.Application.Interfaces;
using RehearsalStudio.Domain.Entities;
using RehearsalStudio.Infrastructure.Data;
namespace RehearsalStudio.Infrastructure.Repositories;
public class EquipmentBookingRepository : IEquipmentBookingRepository
{
    private readonly RehearsalStudioDbContext _context;
    public EquipmentBookingRepository(RehearsalStudioDbContext context)
    {
        _context = context;
    }
    public async Task<IEnumerable<EquipmentBooking>> GetAllAsync()
    {
        return await _context.EquipmentBookings.ToListAsync();
    }
    public async Task<EquipmentBooking?> GetByIdAsync(int idEquipment, int idBooking)
    {
        return await _context.EquipmentBookings.FindAsync(idEquipment, idBooking);
    }
    public async Task<IEnumerable<EquipmentBooking>> GetFilteredAsync(int? idEquipment, int? idBooking)
    {
        var query = _context.EquipmentBookings.AsQueryable();
        if (idEquipment.HasValue)
            query = query.Where(eb => eb.IdEquipment == idEquipment.Value);
        if (idBooking.HasValue)
            query = query.Where(eb => eb.IdBooking == idBooking.Value);
        return await query.ToListAsync();
    }
    public async Task<EquipmentBooking> AddAsync(EquipmentBooking equipmentBooking)
    {
        _context.EquipmentBookings.Add(equipmentBooking);
        await _context.SaveChangesAsync();
        return equipmentBooking;
    }
    public async Task DeleteAsync(int idEquipment, int idBooking)
    {
        var equipmentBooking = await
_context.EquipmentBookings.FindAsync(idEquipment, idBooking);
        if (equipmentBooking != null)
        {
            _context.EquipmentBookings.Remove(equipmentBooking);
            await _context.SaveChangesAsync();
        }
    }
}
```

Файл EquipmentRepository.cs:

```
using Microsoft.EntityFrameworkCore;
using RehearsalStudio.Application.Interfaces;
using RehearsalStudio.Domain.Entities;
using RehearsalStudio.Infrastructure.Data;
namespace RehearsalStudio.Infrastructure.Repositories;
public class EquipmentRepository : IEquipmentRepository
{
    private readonly RehearsalStudioDbContext _context;
    public EquipmentRepository(RehearsalStudioDbContext context)
    {
        _context = context;
    }
    public async Task<IEnumerable<Equipment>> GetAllAsync()
    {
        return await _context.Equipment.ToListAsync();
    }
    public async Task<Equipment?> GetByIdAsync(int id)
    {
        return await _context.Equipment.FindAsync(id);
    }
    public async Task<IEnumerable<Equipment>>
GetFilteredAsync(string? name, string? type, int? idRehearsalPoint)
    {
        var query = _context.Equipment.AsQueryable();
        if (!string.IsNullOrEmpty(name))
            query = query.Where(e => e.Name.Contains(name));
        if (!string.IsNullOrEmpty(type))
            query = query.Where(e => e.Type.Contains(type));
        if (idRehearsalPoint.HasValue)
            query = query.Where(e => e.IdRehearsalPoint ==
idRehearsalPoint.Value);
        return await query.ToListAsync();
    }
    public async Task<Equipment> AddAsync(Equipment equipment)
    {
        _context.Equipment.Add(equipment);
        await _context.SaveChangesAsync();
        return equipment;
    }
    public async Task UpdateAsync(Equipment equipment)
    {
        _context.Equipment.Update(equipment);
        await _context.SaveChangesAsync();
    }
    public async Task DeleteAsync(int id)
    {
        var equipment = await _context.Equipment.FindAsync(id);
        if (equipment != null)
        {
            _context.Equipment.Remove(equipment);
            await _context.SaveChangesAsync();
        }
    }
}
```

Файл RehearsalPointRepository.cs:

```
using Microsoft.EntityFrameworkCore;
using RehearsalStudio.Application.Interfaces;
using RehearsalStudio.Domain.Entities;
using RehearsalStudio.Infrastructure.Data;
namespace RehearsalStudio.Infrastructure.Repositories;
public class RehearsalPointRepository : IRehearsalPointRepository
{
    private readonly RehearsalStudioDbContext _context;
    public RehearsalPointRepository(RehearsalStudioDbContext context)
    {
        _context = context;
    }
    public async Task<IEnumerable<RehearsalPoint>> GetAllAsync()
    {
        return await _context.RehearsalPoints.ToListAsync();
    }
    public async Task<RehearsalPoint?> GetByIdAsync(int id)
    {
        return await _context.RehearsalPoints.FindAsync(id);
    }
    public async Task<IEnumerable<RehearsalPoint>> GetFilteredAsync(string? name, float? minRating)
    {
        var query = _context.RehearsalPoints.AsQueryable();
        if (!string.IsNullOrEmpty(name))
            query = query.Where(rp => rp.Name.Contains(name));
        if (minRating.HasValue)
            query = query.Where(rp => rp.Rating >= minRating.Value);
        return await query.ToListAsync();
    }
    public async Task<RehearsalPoint> AddAsync(RehearsalPoint rehearsalPoint)
    {
        _context.RehearsalPoints.Add(rehearsalPoint);
        await _context.SaveChangesAsync();
        return rehearsalPoint;
    }
    public async Task UpdateAsync(RehearsalPoint rehearsalPoint)
    {
        _context.RehearsalPoints.Update(rehearsalPoint);
        await _context.SaveChangesAsync();
    }
    public async Task DeleteAsync(int id)
    {
        var rehearsalPoint = await _context.RehearsalPoints.FindAsync(id);
        if (rehearsalPoint != null)
        {
            _context.RehearsalPoints.Remove(rehearsalPoint);
            await _context.SaveChangesAsync();
        }
    }
}
```

Файл RoomRepository.cs:
```
using Microsoft.EntityFrameworkCore;
using RehearsalStudio.Application.Interfaces;
using RehearsalStudio.Domain.Entities;
using RehearsalStudio.Infrastructure.Data;
namespace RehearsalStudio.Infrastructure.Repositories;
public class RoomRepository : IRoomRepository
{
    private readonly RehearsalStudioDbContext _context;
    public RoomRepository(RehearsalStudioDbContext context)
    {
        _context = context;
    }
    public async Task<IEnumerable<Room>> GetAllAsync()
    {
        return await _context.Rooms.ToListAsync();
    }
    public async Task<Room?> GetByIdAsync(int id)
    {
        return await _context.Rooms.FindAsync(id);
    }
    public async Task<IEnumerable<Room>> GetFilteredAsync(string? name, int? minPrice, int? idRehearsalPoint)
    {
        var query = _context.Rooms.AsQueryable();
        if (!string.IsNullOrEmpty(name))
            query = query.Where(r => r.Name.Contains(name));
        if (minPrice.HasValue)
            query = query.Where(r => r.Price >= minPrice.Value);
        if (idRehearsalPoint.HasValue)
            query = query.Where(r => r.IdRehearsalPoint == idRehearsalPoint.Value);
        return await query.ToListAsync();
    }
    public async Task<Room> AddAsync(Room room)
    {
        _context.Rooms.Add(room);
        await _context.SaveChangesAsync();
        return room;
    }
    public async Task UpdateAsync(Room room)
    {
        _context.Rooms.Update(room);
        await _context.SaveChangesAsync();
    }
    public async Task DeleteAsync(int id)
    {
        var room = await _context.Rooms.FindAsync(id);
        if (room != null)
        {
            _context.Rooms.Remove(room);
            await _context.SaveChangesAsync();
        }
    }
}
```

Файл ServiceBookingRepository.cs:
```
using Microsoft.EntityFrameworkCore;
using RehearsalStudio.Application.Interfaces;
using RehearsalStudio.Domain.Entities;
using RehearsalStudio.Infrastructure.Data;
```

```
namespace RehearsalStudio.Infrastructure.Repositories;
public class ServiceBookingRepository : IServiceBookingRepository
{
    private readonly RehearsalStudioDbContext _context;
    public ServiceBookingRepository(RehearsalStudioDbContext context)
    {
        _context = context;
    }
    public async Task<IEnumerable<ServiceBooking>> GetAllAsync()
    {
        return await _context.ServiceBookings.ToListAsync();
    }
    public async Task<ServiceBooking?> GetByIdAsync(int idService, int idBooking)
    {
        return await _context.ServiceBookings.FindAsync(idService, idBooking);
    }
    public async Task<IEnumerable<ServiceBooking>> GetFilteredAsync(int? idService, int? idBooking)
    {
        var query = _context.ServiceBookings.AsQueryable();
        if (idService.HasValue)
            query = query.Where(sb => sb.IdService == idService.Value);
        if (idBooking.HasValue)
            query = query.Where(sb => sb.IdBooking == idBooking.Value);
        return await query.ToListAsync();
    }
    public async Task<ServiceBooking> AddAsync(ServiceBooking serviceBooking)
    {
        _context.ServiceBookings.Add(serviceBooking);
        await _context.SaveChangesAsync();
        return serviceBooking;
    }
    public async Task DeleteAsync(int idService, int idBooking)
    {
        var serviceBooking = await _context.ServiceBookings.FindAsync(idService, idBooking);
        if (serviceBooking != null)
        {
            _context.ServiceBookings.Remove(serviceBooking);
            await _context.SaveChangesAsync();
        }
    }
}
```

Файл ServiceRepository.cs:

```
using Microsoft.EntityFrameworkCore;
using RehearsalStudio.Application.Interfaces;
using RehearsalStudio.Domain.Entities;
using RehearsalStudio.Infrastructure.Data;
namespace RehearsalStudio.Infrastructure.Repositories;
public class ServiceRepository : IServiceRepository
{
    private readonly RehearsalStudioDbContext _context;
    public ServiceRepository(RehearsalStudioDbContext context)
    {
        _context = context;
    }
    public async Task<IEnumerable<Service>> GetAllAsync()
    {
        return await _context.Services.ToListAsync();
    }
    public async Task<Service?> GetByIdAsync(int id)
    {
        return await _context.Services.FindAsync(id);
    }
    public async Task<IEnumerable<Service>> GetFilteredAsync(string? name, string? type, int? idRehearsalPoint)
    {
        var query = _context.Services.AsQueryable();
        if (!string.IsNullOrEmpty(name))
            query = query.Where(s => s.Name.Contains(name));
        if (!string.IsNullOrEmpty(type))
            query = query.Where(s => s.Type.Contains(type));
        if (idRehearsalPoint.HasValue)
            query = query.Where(s => s.IdRehearsalPoint == idRehearsalPoint.Value);
        return await query.ToListAsync();
    }
    public async Task<Service> AddAsync(Service service)
    {
        _context.Services.Add(service);
        await _context.SaveChangesAsync();
        return service;
    }
    public async Task UpdateAsync(Service service)
    {
        _context.Services.Update(service);
        await _context.SaveChangesAsync();
    }
    public async Task DeleteAsync(int id)
    {
        var service = await _context.Services.FindAsync(id);
        if (service != null)
        {
            _context.Services.Remove(service);
            await _context.SaveChangesAsync();
        }
    }
}
```

Файл StaffRepository.cs:

```
using Microsoft.EntityFrameworkCore;
using RehearsalStudio.Application.Interfaces;
using RehearsalStudio.Domain.Entities;
using RehearsalStudio.Infrastructure.Data;
namespace RehearsalStudio.Infrastructure.Repositories;
public class StaffRepository : IStaffRepository
{
    private readonly RehearsalStudioDbContext _context;
    public StaffRepository(RehearsalStudioDbContext context)
    {
        _context = context;
    }
    public async Task<IEnumerable<Staff>> GetAllAsync()
    {
        return await _context.Staff.ToListAsync();
    }
    public async Task<Staff?> GetByIdAsync(int id)
    {
        return await _context.Staff.FindAsync(id);
    }
    public async Task<IEnumerable<Staff>> GetFilteredAsync(string? fullName, int? minAge, int? idRehearsalPoint)
    {
        var query = _context.Staff.AsQueryable();
        if (!string.IsNullOrEmpty(fullName))
            query = query.Where(s => s.FullName.Contains(fullName));
        if (minAge.HasValue)
            query = query.Where(s => s.Age >= minAge.Value);
        if (idRehearsalPoint.HasValue)
            query = query.Where(s => s.IdRehearsalPoint == idRehearsalPoint.Value);
        return await query.ToListAsync();
    }
    public async Task<Staff> AddAsync(Staff staff)
    {
        _context.Staff.Add(staff);
        await _context.SaveChangesAsync();
        return staff;
    }
    public async Task UpdateAsync(Staff staff)
    {
        _context.Staff.Update(staff);
        await _context.SaveChangesAsync();
    }
    public async Task DeleteAsync(int id)
    {
        var staff = await _context.Staff.FindAsync(id);
        if (staff != null)
        {
            _context.Staff.Remove(staff);
            await _context.SaveChangesAsync();
        }
    }
}
```

Файл UserRepository.cs:

```
using Microsoft.EntityFrameworkCore;
using RehearsalStudio.Application.Interfaces;
using RehearsalStudio.Domain.Entities;
using RehearsalStudio.Infrastructure.Data;
namespace RehearsalStudio.Infrastructure.Repositories;
public class UserRepository : IUserRepository
{
    private readonly RehearsalStudioDbContext _context;
    public UserRepository(RehearsalStudioDbContext context)
    {
        _context = context;
    }
    public async Task<IEnumerable<User>> GetAllAsync()
    {
        return await _context.Users.ToListAsync();
    }
    public async Task<User?> GetByIdAsync(int id)
    {
        return await _context.Users.FindAsync(id);
    }
    public async Task<IEnumerable<User>> GetFilteredAsync(string? fullName, string? email)
    {
        var query = _context.Users.AsQueryable();
        if (!string.IsNullOrEmpty(fullName))
            query = query.Where(u => u.FullName.Contains(fullName));
        if (!string.IsNullOrEmpty(email))
            query = query.Where(u => u.Email.Contains(email));
        return await query.ToListAsync();
    }
    public async Task<User> AddAsync(User user)
    {
        _context.Users.Add(user);
        await _context.SaveChangesAsync();
        return user;
    }
    public async Task UpdateAsync(User user)
    {
        _context.Users.Update(user);
        await _context.SaveChangesAsync();
    }
    public async Task DeleteAsync(int id)
    {
        var user = await _context.Users.FindAsync(id);
        if (user != null)
        {
            _context.Users.Remove(user);
            await _context.SaveChangesAsync();
        }
    }
}
```

Файл BackupService.cs:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;
using System.Threading.Tasks;
using Dapper;
using Microsoft.EntityFrameworkCore;
using Npgsql;
using RehearsalStudio.Application.Interfaces;
using RehearsalStudio.Infrastructure.Data;
using System.Text.Json;
using System.Linq;
namespace RehearsalStudio.Application.Services;
public class BackupService : IBackupService
{
    private readonly RehearsalStudioDbContext _context;
    public BackupService(RehearsalStudioDbContext context)
    {
        _context = context;
    }
    public async Task<string> CreateDatabaseBackupAsync()
    {
        var connectionString = _context.Database.GetConnectionString();
        var backupFilePath = $"backup_{DateTime.Now:yyyyMMddHHmmss}.sql";
        using var connection = new NpgsqlConnection(connectionString);
        await connection.OpenAsync();
        var backupScript = new StringBuilder();
        // List of tables to back up
        var tables = new[] { "rehearsal_points", "rooms", "service", "equipment", "staff", "users", "booking", "service_booking", "equipment_booking" };
        foreach (var table in tables)
        {
            // Generate table structure using information_schema
            var columns = await connection.QueryAsync<ColumnInfo>(
                @"SELECT column_name, data_type, is_nullable, character_maximum_length
                  FROM information_schema.columns
                  WHERE table_schema = 'main' AND table_name = @TableName",
                new { TableName = table });
            // Start CREATE TABLE statement
            backupScript.AppendLine($"DROP TABLE IF EXISTS main.{table} CASCADE;");
            backupScript.AppendLine($"CREATE TABLE main.{table} (");
            var columnDefinitions = columns.Select(c =>
            {
                var dataType = c.data_type switch
                {
                    "integer" => "INTEGER",
                    "real" => "REAL",
                    "boolean" => "BOOLEAN",
                    "text" => "TEXT",
                    "timestamp with time zone" => "TIMESTAMP WITH TIME ZONE",
                    _ => c.data_type.ToUpper()
                };
                var nullable = c.is_nullable == "YES" ? "" : " NOT NULL";
                return $"    {c.column_name} {dataType}{nullable}";
            });
            backupScript.AppendLine(string.Join(",\n", columnDefinitions));
            // Add primary key constraints
            var primaryKeys = await connection.QueryAsync<string>(
                @"SELECT a.attname
                  FROM pg_index i
                  JOIN pg_attribute a ON a.attrelid = i.indrelid AND a.attnum = ANY(i.indkey)
                  JOIN pg_class c ON c.oid = i.indrelid
                  JOIN pg_namespace n ON n.oid = c.relnamespace
                  WHERE n.nspname = 'main' AND c.relname = @TableName AND i.indisprimary",
                new { TableName = table });
            if (primaryKeys.Any())
            {
                backupScript.AppendLine($",    PRIMARY KEY ({string.Join(", ", primaryKeys)})");
            }
            backupScript.AppendLine(");");
            backupScript.AppendLine();
            // Export table data
            using var reader = await connection.ExecuteReaderAsync($"SELECT * FROM main.{table}");
            var columnNames = Enumerable.Range(0, reader.FieldCount).Select(reader.GetName).ToList();
            while (await reader.ReadAsync())
            {
                var values = new List<string>();
                for (int i = 0; i < reader.FieldCount; i++)
                {
                    var value = reader.GetValue(i);
                    if (value == DBNull.Value)
                        values.Add("NULL");
                    else if (reader.GetFieldType(i) == typeof(DateTime))
                        values.Add($"'{(DateTime)value:yyyy-MM-dd HH:mm:ss.fffz}'");
                    else if (reader.GetFieldType(i) == typeof(string))
                        values.Add($"'{value.ToString().Replace("'", "''")}'");
                    else
                        values.Add(value.ToString());
                }
                backupScript.AppendLine($"INSERT INTO main.{table} ({string.Join(", ", columnNames)}) VALUES ({string.Join(", ", values)});");
            }
            backupScript.AppendLine();
        }
        await File.WriteAllTextAsync(backupFilePath, backupScript.ToString());
        return backupFilePath;
    }
    public async Task<string> SaveQueryResultsToFileAsync(string sqlQuery, string fileFormat = "json")
    {
        var connectionString = _context.Database.GetConnectionString();
        var resultFilePath = $"query_results_{DateTime.Now:yyyyMMddHHmmss}.{fileFormat}";
        using var connection = new NpgsqlConnection(connectionString);
        await connection.OpenAsync();
        var results = await connection.QueryAsync<dynamic>(sqlQuery);
        if (fileFormat.ToLower() == "json")
        {
            var json = JsonSerializer.Serialize(results);
            await File.WriteAllTextAsync(resultFilePath, json);
        }
        else if (fileFormat.ToLower() == "csv")
        {
            var csv = new StringBuilder();
            if (results.Any())
            {
                var columns = ((IDictionary<string, object>)results.First()).Keys;
                csv.AppendLine(string.Join(",", columns));
                foreach (var row in results)
                {
                    var dict = (IDictionary<string, object>)row;
                    var values = columns.Select(c => dict[c]?.ToString() ?? string.Empty).Select(v => v.Contains(",") ? $"\"{v}\"" : v);
                    csv.AppendLine(string.Join(",", values));
                }
            }
            await File.WriteAllTextAsync(resultFilePath, csv.ToString());
        }
        else
        {
            throw new ArgumentException("Unsupported file format. Use 'json' or 'csv'.");
        }
        return resultFilePath;
    }
    private class ColumnInfo
    {
        public string column_name { get; set; }
        public string data_type { get; set; }
        public string is_nullable { get; set; }
        public int? character_maximum_length { get; set; }
    }
}
```

Файл BookingService.cs:

```
using RehearsalStudio.Application.DTOs;
using RehearsalStudio.Application.Interfaces;
using RehearsalStudio.Domain.Entities;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
namespace RehearsalStudio.Application.Services;
public class BookingService : IBookingService
{
    private readonly IBookingRepository _repository;
    public BookingService(IBookingRepository repository)
    {
        _repository = repository;
    }
    public async Task<IEnumerable<BookingDto>> GetAllAsync()
    {
        var entities = await _repository.GetAllAsync();
        return entities.Select(e => new BookingDto
        {
            Id = e.Id,
            Time = e.Time,
            Duration = e.Duration,
            Cost = e.Cost,
            CreationDate = e.CreationDate,
            Status = e.Status,
            NumberOfPeople = e.NumberOfPeople,
            IdRoom = e.IdRoom,
            IdUser = e.IdUser
        });
    }
    public async Task<BookingDto?> GetByIdAsync(int id)
    {
        var entity = await _repository.GetByIdAsync(id);
        if (entity == null) return null;
        return new BookingDto
        {
            Id = entity.Id,
            Time = entity.Time,
            Duration = entity.Duration,
            Cost = entity.Cost,
            CreationDate = entity.CreationDate,
            Status = entity.Status,
            NumberOfPeople = entity.NumberOfPeople,
            IdRoom = entity.IdRoom,
            IdUser = entity.IdUser
        };
    }
    public async Task<IEnumerable<BookingDto>> GetFilteredAsync(string? status, int? idRoom, int? idUser)
    {
        var entities = await _repository.GetFilteredAsync(status, idRoom, idUser);
        return entities.Select(e => new BookingDto
        {
            Id = e.Id,
            Time = e.Time,
            Duration = e.Duration,
            Cost = e.Cost,
            CreationDate = e.CreationDate,
            Status = e.Status,
            NumberOfPeople = e.NumberOfPeople,
            IdRoom = e.IdRoom,
            IdUser = e.IdUser
        });
    }
    public async Task<BookingDto> CreateAsync(BookingDto dto)
    {
        if (string.IsNullOrEmpty(dto.Status) || dto.Cost <= 0 || dto.NumberOfPeople <= 0)
            throw new ArgumentException("Status, Cost, and NumberOfPeople are required and must be valid.");
        var entity = new Booking
        {
            Time = dto.Time,
            Duration = dto.Duration,
            Cost = dto.Cost,
            CreationDate = dto.CreationDate,
            Status = dto.Status,
            NumberOfPeople = dto.NumberOfPeople,
            IdRoom = dto.IdRoom,
            IdUser = dto.IdUser
        };
        var created = await _repository.AddAsync(entity);
        return new BookingDto
        {
            Id = created.Id,
            Time = created.Time,
            Duration = created.Duration,
            Cost = created.Cost,
            CreationDate = created.CreationDate,
            Status = created.Status,
            NumberOfPeople = created.NumberOfPeople,
            IdRoom = created.IdRoom,
            IdUser = created.IdUser
        };
    }
    public async Task UpdateAsync(int id, BookingDto dto)
    {
        if (string.IsNullOrEmpty(dto.Status) || dto.Cost <= 0 || dto.NumberOfPeople <= 0)
            throw new ArgumentException("Status, Cost, and NumberOfPeople are required and must be valid.");
        var entity = await _repository.GetByIdAsync(id);
        if (entity == null)
            throw new KeyNotFoundException($"Booking with ID {id} not found.");
        entity.Time = dto.Time;
        entity.Duration = dto.Duration;
        entity.Cost = dto.Cost;
        entity.CreationDate = dto.CreationDate;
```

```csharp
        entity.Status = dto.Status;
        entity.NumberOfPeople = dto.NumberOfPeople;
        entity.IdRoom = dto.IdRoom;
        entity.IdUser = dto.IdUser;
        await _repository.UpdateAsync(entity);
    }
    public async Task DeleteAsync(int id)
    {
        var entity = await _repository.GetByIdAsync(id);
        if (entity == null)
            throw new KeyNotFoundException($"Booking with ID {id} not found.");
        await _repository.DeleteAsync(id);
    }
}
```

Файл EquipmentBookingService.cs:

```csharp
using RehearsalStudio.Application.DTOs;
using RehearsalStudio.Application.Interfaces;
using RehearsalStudio.Domain.Entities;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
namespace RehearsalStudio.Application.Services;
public class EquipmentBookingService : IEquipmentBookingService
{
    private readonly IEquipmentBookingRepository _repository;
    public EquipmentBookingService(IEquipmentBookingRepository repository)
    {
        _repository = repository;
    }
    public async Task<IEnumerable<EquipmentBookingDto>> GetAllAsync()
    {
        var entities = await _repository.GetAllAsync();
        return entities.Select(e => new EquipmentBookingDto
        {
            IdEquipment = e.IdEquipment,
            IdBooking = e.IdBooking
        });
    }
    public async Task<EquipmentBookingDto?> GetByIdAsync(int idEquipment, int idBooking)
    {
        var entity = await _repository.GetByIdAsync(idEquipment, idBooking);
        if (entity == null) return null;
        return new EquipmentBookingDto
        {
            IdEquipment = entity.IdEquipment,
            IdBooking = entity.IdBooking
        };
    }
    public async Task<IEnumerable<EquipmentBookingDto>> GetFilteredAsync(int? idEquipment, int? idBooking)
    {
        var entities = await _repository.GetFilteredAsync(idEquipment, idBooking);
        return entities.Select(e => new EquipmentBookingDto
        {
            IdEquipment = e.IdEquipment,
            IdBooking = e.IdBooking
        });
    }
    public async Task<EquipmentBookingDto> CreateAsync(EquipmentBookingDto dto)
    {
        if (dto.IdEquipment <= 0 || dto.IdBooking <= 0)
            throw new ArgumentException("IdEquipment and IdBooking must be valid.");
        var entity = new EquipmentBooking
        {
            IdEquipment = dto.IdEquipment,
            IdBooking = dto.IdBooking
        };
        var created = await _repository.AddAsync(entity);
        return new EquipmentBookingDto
        {
            IdEquipment = created.IdEquipment,
            IdBooking = created.IdBooking
        };
    }
    public async Task DeleteAsync(int idEquipment, int idBooking)
    {
        var entity = await _repository.GetByIdAsync(idEquipment, idBooking);
        if (entity == null)
            throw new KeyNotFoundException($"EquipmentBooking with IdEquipment {idEquipment} and IdBooking {idBooking} not found.");
        await _repository.DeleteAsync(idEquipment, idBooking);
    }
}
```

Файл EquipmentService.cs:

```
using RehearsalStudio.Application.DTOs;
using RehearsalStudio.Application.Interfaces;
using RehearsalStudio.Domain.Entities;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
namespace RehearsalStudio.Application.Services;
public class EquipmentService : IEquipmentService
{
    private readonly IEquipmentRepository _repository;
    public EquipmentService(IEquipmentRepository repository)
    {
        _repository = repository;
    }
    public async Task<IEnumerable<EquipmentDto>> GetAllAsync()
    {
        var entities = await _repository.GetAllAsync();
        return entities.Select(e => new EquipmentDto
        {
            Id = e.Id,
            Name = e.Name,
            Type = e.Type,
            Brand = e.Brand,
            Model = e.Model,
            Condition = e.Condition,
            IdRehearsalPoint = e.IdRehearsalPoint
        });
    }
    public async Task<EquipmentDto?> GetByIdAsync(int id)
    {
        var entity = await _repository.GetByIdAsync(id);
        if (entity == null) return null;
        return new EquipmentDto
        {
            Id = entity.Id,
            Name = entity.Name,
            Type = entity.Type,
            Brand = entity.Brand,
            Model = entity.Model,
            Condition = entity.Condition,
            IdRehearsalPoint = entity.IdRehearsalPoint
        };
    }
    public async Task<IEnumerable<EquipmentDto>> GetFilteredAsync(string? name, string? type, int? idRehearsalPoint)
    {
        var entities = await _repository.GetFilteredAsync(name, type, idRehearsalPoint);
        return entities.Select(e => new EquipmentDto
        {
            Id = e.Id,
            Name = e.Name,
            Type = e.Type,
            Brand = e.Brand,
            Model = e.Model,
            Condition = e.Condition,
            IdRehearsalPoint = e.IdRehearsalPoint
        });
    }
    public async Task<EquipmentDto> CreateAsync(EquipmentDto dto)
    {
        if (string.IsNullOrEmpty(dto.Name) || string.IsNullOrEmpty(dto.Type) || string.IsNullOrEmpty(dto.Brand) ||
            string.IsNullOrEmpty(dto.Model) || string.IsNullOrEmpty(dto.Condition))
            throw new ArgumentException("Name, Type, Brand, Model, and Condition are required.");
        var entity = new Equipment
        {
            Name = dto.Name,
            Type = dto.Type,
            Brand = dto.Brand,
            Model = dto.Model,
            Condition = dto.Condition,
            IdRehearsalPoint = dto.IdRehearsalPoint
        };
        var created = await _repository.AddAsync(entity);
        return new EquipmentDto
        {
            Id = created.Id,
            Name = created.Name,
            Type = created.Type,
            Brand = created.Brand,
            Model = created.Model,
            Condition = created.Condition,
            IdRehearsalPoint = created.IdRehearsalPoint
        };
    }
    public async Task UpdateAsync(int id, EquipmentDto dto)
    {
        if (string.IsNullOrEmpty(dto.Name) || string.IsNullOrEmpty(dto.Type) || string.IsNullOrEmpty(dto.Brand) ||
            string.IsNullOrEmpty(dto.Model) || string.IsNullOrEmpty(dto.Condition))
            throw new ArgumentException("Name, Type, Brand, Model, and Condition are required.");
        var entity = await _repository.GetByIdAsync(id);
        if (entity == null)
            throw new KeyNotFoundException($"Equipment with ID {id} not found.");
        entity.Name = dto.Name;
        entity.Type = dto.Type;
        entity.Brand = dto.Brand;
        entity.Model = dto.Model;
        entity.Condition = dto.Condition;
        entity.IdRehearsalPoint = dto.IdRehearsalPoint;
        await _repository.UpdateAsync(entity);
    }
    public async Task DeleteAsync(int id)
    {
        var entity = await _repository.GetByIdAsync(id);
        if (entity == null)
            throw new KeyNotFoundException($"Equipment with ID {id} not found.");
        await _repository.DeleteAsync(id);
    }
}
```

Файл RehearsalPointService.cs:

```
using RehearsalStudio.Application.DTOs;
using RehearsalStudio.Application.Interfaces;
using RehearsalStudio.Domain.Entities;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
namespace RehearsalStudio.Application.Services;
public class RehearsalPointService : IRehearsalPointService
{
    private readonly IRehearsalPointRepository _repository;
    public RehearsalPointService(IRehearsalPointRepository repository)
    {
        _repository = repository;
    }
    public async Task<IEnumerable<RehearsalPointDto>> GetAllAsync()
    {
        var entities = await _repository.GetAllAsync();
        return entities.Select(e => new RehearsalPointDto
        {
            Id = e.Id,
            Rating = e.Rating,
            ContactNumber = e.ContactNumber,
            Schedule = e.Schedule,
            Name = e.Name,
            Address = e.Address
        });
    }
    public async Task<RehearsalPointDto?> GetByIdAsync(int id)
    {
        var entity = await _repository.GetByIdAsync(id);
        if (entity == null) return null;
        return new RehearsalPointDto
        {
            Id = entity.Id,
            Rating = entity.Rating,
            ContactNumber = entity.ContactNumber,
            Schedule = entity.Schedule,
            Name = entity.Name,
            Address = entity.Address
        };
    }
    public async Task<IEnumerable<RehearsalPointDto>> GetFilteredAsync(string? name, float? minRating)
    {
        var entities = await _repository.GetFilteredAsync(name, minRating);
        return entities.Select(e => new RehearsalPointDto
        {
            Id = e.Id,
            Rating = e.Rating,
            ContactNumber = e.ContactNumber,
            Schedule = e.Schedule,
            Name = e.Name,
            Address = e.Address
        });
    }
    public async Task<RehearsalPointDto> CreateAsync(RehearsalPointDto dto)
    {
        if (string.IsNullOrEmpty(dto.Name) || string.IsNullOrEmpty(dto.Address) || string.IsNullOrEmpty(dto.ContactNumber))
            throw new ArgumentException("Name, Address, and ContactNumber are required.");
        var entity = new RehearsalPoint
        {
            Rating = dto.Rating,
            ContactNumber = dto.ContactNumber,
            Schedule = dto.Schedule,
            Name = dto.Name,
            Address = dto.Address
        };
        var created = await _repository.AddAsync(entity);
        return new RehearsalPointDto
        {
            Id = created.Id,
            Rating = created.Rating,
            ContactNumber = created.ContactNumber,
            Schedule = created.Schedule,
            Name = created.Name,
            Address = created.Address
        };
    }
    public async Task UpdateAsync(int id, RehearsalPointDto dto)
    {
        if (string.IsNullOrEmpty(dto.Name) || string.IsNullOrEmpty(dto.Address) || string.IsNullOrEmpty(dto.ContactNumber))
            throw new ArgumentException("Name, Address, and ContactNumber are required.");
        var entity = await _repository.GetByIdAsync(id);
        if (entity == null)
            throw new KeyNotFoundException($"RehearsalPoint with ID {id} not found.");
        entity.Rating = dto.Rating;
        entity.ContactNumber = dto.ContactNumber;
        entity.Schedule = dto.Schedule;
        entity.Name = dto.Name;
        entity.Address = dto.Address;
        await _repository.UpdateAsync(entity);
    }
    public async Task DeleteAsync(int id)
    {
        var entity = await _repository.GetByIdAsync(id);
        if (entity == null)
            throw new KeyNotFoundException($"RehearsalPoint with ID {id} not found.");
        await _repository.DeleteAsync(id);
    }
}
```

Файл RoomService.cs:

```
using RehearsalStudio.Application.DTOs;
using RehearsalStudio.Application.Interfaces;
using RehearsalStudio.Domain.Entities;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
namespace RehearsalStudio.Application.Services;
public class RoomService : IRoomService
{
    private readonly IRoomRepository _repository;
    public RoomService(IRoomRepository repository)
    {
        _repository = repository;
    }
    public async Task<IEnumerable<RoomDto>> GetAllAsync()
    {
        var entities = await _repository.GetAllAsync();
        return entities.Select(e => new RoomDto
        {
            Id = e.Id,
            Name = e.Name,
            AirConditioner = e.AirConditioner,
            Price = e.Price,
            RecordingSupport = e.RecordingSupport,
            Area = e.Area,
            IdRehearsalPoint = e.IdRehearsalPoint
        });
    }
    public async Task<RoomDto?> GetByIdAsync(int id)
    {
        var entity = await _repository.GetByIdAsync(id);
        if (entity == null) return null;
        return new RoomDto
        {
            Id = entity.Id,
            Name = entity.Name,
            AirConditioner = entity.AirConditioner,
            Price = entity.Price,
            RecordingSupport = entity.RecordingSupport,
            Area = entity.Area,
            IdRehearsalPoint = entity.IdRehearsalPoint
        };
    }
    public async Task<IEnumerable<RoomDto>> GetFilteredAsync(string? name, int? minPrice, int? idRehearsalPoint)
    {
        var entities = await _repository.GetFilteredAsync(name, minPrice, idRehearsalPoint);
        return entities.Select(e => new RoomDto
        {
            Id = e.Id,
            Name = e.Name,
            AirConditioner = e.AirConditioner,
            Price = e.Price,
            RecordingSupport = e.RecordingSupport,
            Area = e.Area,
            IdRehearsalPoint = e.IdRehearsalPoint
        });
    }
    public async Task<RoomDto> CreateAsync(RoomDto dto)
    {
        if (string.IsNullOrEmpty(dto.Name) || dto.Price <= 0 || dto.Area <= 0)
            throw new ArgumentException("Name, Price, and Area are required and must be valid.");
        var entity = new Room
        {
            Name = dto.Name,
            AirConditioner = dto.AirConditioner,
            Price = dto.Price,
            RecordingSupport = dto.RecordingSupport,
            Area = dto.Area,
            IdRehearsalPoint = dto.IdRehearsalPoint
        };
        var created = await _repository.AddAsync(entity);
        return new RoomDto
        {
            Id = created.Id,
            Name = created.Name,
            AirConditioner = created.AirConditioner,
            Price = created.Price,
            RecordingSupport = created.RecordingSupport,
            Area = created.Area,
            IdRehearsalPoint = created.IdRehearsalPoint
        };
    }
    public async Task UpdateAsync(int id, RoomDto dto)
    {
        if (string.IsNullOrEmpty(dto.Name) || dto.Price <= 0 || dto.Area <= 0)
            throw new ArgumentException("Name, Price, and Area are required and must be valid.");
        var entity = await _repository.GetByIdAsync(id);
        if (entity == null)
            throw new KeyNotFoundException($"Room with ID {id} not found.");
        entity.Name = dto.Name;
        entity.AirConditioner = dto.AirConditioner;
        entity.Price = dto.Price;
        entity.RecordingSupport = dto.RecordingSupport;
        entity.Area = dto.Area;
        entity.IdRehearsalPoint = dto.IdRehearsalPoint;
        await _repository.UpdateAsync(entity);
    }
    public async Task DeleteAsync(int id)
    {
        var entity = await _repository.GetByIdAsync(id);
        if (entity == null)
            throw new KeyNotFoundException($"Room with ID {id} not found.");
        await _repository.DeleteAsync(id);
    }
```

```
}
```

Файл ServiceBookingService.cs:

```
using RehearsalStudio.Application.DTOs;
using RehearsalStudio.Application.Interfaces;
using RehearsalStudio.Domain.Entities;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
namespace RehearsalStudio.Application.Services;
public class ServiceBookingService : IServiceBookingService
{
    private readonly IServiceBookingRepository _repository;
    public ServiceBookingService(IServiceBookingRepository repository)
    {
        _repository = repository;
    }
    public async Task<IEnumerable<ServiceBookingDto>> GetAllAsync()
    {
        var entities = await _repository.GetAllAsync();
        return entities.Select(e => new ServiceBookingDto
        {
            IdService = e.IdService,
            IdBooking = e.IdBooking
        });
    }
    public async Task<ServiceBookingDto?> GetByIdAsync(int idService, int idBooking)
    {
        var entity = await _repository.GetByIdAsync(idService, idBooking);
        if (entity == null) return null;
        return new ServiceBookingDto
        {
            IdService = entity.IdService,
            IdBooking = entity.IdBooking
        };
    }
    public async Task<IEnumerable<ServiceBookingDto>> GetFilteredAsync(int? idService, int? idBooking)
    {
        var entities = await _repository.GetFilteredAsync(idService, idBooking);
        return entities.Select(e => new ServiceBookingDto
        {
            IdService = e.IdService,
            IdBooking = e.IdBooking
        });
    }
    public async Task<ServiceBookingDto> CreateAsync(ServiceBookingDto dto)
    {
        if (dto.IdService <= 0 || dto.IdBooking <= 0)
            throw new ArgumentException("IdService and IdBooking must be valid.");
        var entity = new ServiceBooking
        {
            IdService = dto.IdService,
            IdBooking = dto.IdBooking
        };
        var created = await _repository.AddAsync(entity);
        return new ServiceBookingDto
        {
            IdService = created.IdService,
            IdBooking = created.IdBooking
        };
    }
    public async Task DeleteAsync(int idService, int idBooking)
    {
        var entity = await _repository.GetByIdAsync(idService, idBooking);
        if (entity == null)
            throw new KeyNotFoundException($"ServiceBooking with IdService {idService} and IdBooking {idBooking} not found.");
        await _repository.DeleteAsync(idService, idBooking);
    }
}
```

Файл ServiceService.cs:

```
using RehearsalStudio.Application.DTOs;
using RehearsalStudio.Application.Interfaces;
using RehearsalStudio.Domain.Entities;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
namespace RehearsalStudio.Application.Services;
public class ServiceService : IServiceService
{
    private readonly IServiceRepository _repository;
    public ServiceService(IServiceRepository repository)
    {
        _repository = repository;
    }
    public async Task<IEnumerable<ServiceDto>> GetAllAsync()
    {
        var entities = await _repository.GetAllAsync();
        return entities.Select(e => new ServiceDto
        {
            Id = e.Id,
            Name = e.Name,
            Price = e.Price,
            Type = e.Type,
            Requirements = e.Requirements,
            IdRehearsalPoint = e.IdRehearsalPoint
        });
    }
    public async Task<ServiceDto?> GetByIdAsync(int id)
    {
        var entity = await _repository.GetByIdAsync(id);
        if (entity == null) return null;
        return new ServiceDto
        {
            Id = entity.Id,
            Name = entity.Name,
            Price = entity.Price,
            Type = entity.Type,
            Requirements = entity.Requirements,
            IdRehearsalPoint = entity.IdRehearsalPoint
        };
    }
    public async Task<IEnumerable<ServiceDto>> GetFilteredAsync(string? name, string? type, int? idRehearsalPoint)
    {
        var entities = await _repository.GetFilteredAsync(name, type, idRehearsalPoint);
        return entities.Select(e => new ServiceDto
        {
            Id = e.Id,
            Name = e.Name,
            Price = e.Price,
            Type = e.Type,
            Requirements = e.Requirements,
            IdRehearsalPoint = e.IdRehearsalPoint
        });
    }
    public async Task<ServiceDto> CreateAsync(ServiceDto dto)
    {
        if (string.IsNullOrEmpty(dto.Name) || string.IsNullOrEmpty(dto.Type) || dto.Price <= 0)
            throw new ArgumentException("Name, Type, and Price are required and must be valid.");
        var entity = new Service
        {
            Name = dto.Name,
            Price = dto.Price,
            Type = dto.Type,
            Requirements = dto.Requirements,
            IdRehearsalPoint = dto.IdRehearsalPoint
        };
        var created = await _repository.AddAsync(entity);
        return new ServiceDto
        {
            Id = created.Id,
            Name = created.Name,
            Price = created.Price,
            Type = created.Type,
            Requirements = created.Requirements,
            IdRehearsalPoint = created.IdRehearsalPoint
        };
    }
    public async Task UpdateAsync(int id, ServiceDto dto)
    {
        if (string.IsNullOrEmpty(dto.Name) || string.IsNullOrEmpty(dto.Type) || dto.Price <= 0)
            throw new ArgumentException("Name, Type, and Price are required and must be valid.");
        var entity = await _repository.GetByIdAsync(id);
        if (entity == null)
            throw new KeyNotFoundException($"Service with ID {id} not found.");
        entity.Name = dto.Name;
        entity.Price = dto.Price;
        entity.Type = dto.Type;
        entity.Requirements = dto.Requirements;
        entity.IdRehearsalPoint = dto.IdRehearsalPoint;
        await _repository.UpdateAsync(entity);
    }
    public async Task DeleteAsync(int id)
    {
        var entity = await _repository.GetByIdAsync(id);
        if (entity == null)
            throw new KeyNotFoundException($"Service with ID {id} not found.");
        await _repository.DeleteAsync(id);
    }
}
```

Файл StaffService.cs:

```
using RehearsalStudio.Application.DTOs;
using RehearsalStudio.Application.Interfaces;
using RehearsalStudio.Domain.Entities;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
namespace RehearsalStudio.Application.Services;
public class StaffService : IStaffService
{
    private readonly IStaffRepository _repository;
    public StaffService(IStaffRepository repository)
    {
        _repository = repository;
    }
    public async Task<IEnumerable<StaffDto>> GetAllAsync()
    {
        var entities = await _repository.GetAllAsync();
        return entities.Select(e => new StaffDto
        {
            Id = e.Id,
            FullName = e.FullName,
            Address = e.Address,
            Experience = e.Experience,
            Phone = e.Phone,
            Age = e.Age,
            IdRehearsalPoint = e.IdRehearsalPoint
        });
    }
    public async Task<StaffDto?> GetByIdAsync(int id)
    {
        var entity = await _repository.GetByIdAsync(id);
        if (entity == null) return null;
        return new StaffDto
        {
            Id = entity.Id,
            FullName = entity.FullName,
            Address = entity.Address,
            Experience = entity.Experience,
            Phone = entity.Phone,
            Age = entity.Age,
            IdRehearsalPoint = entity.IdRehearsalPoint
        };
    }
    public async Task<IEnumerable<StaffDto>> GetFilteredAsync(string? fullName, int? minAge, int? idRehearsalPoint)
    {
        var entities = await _repository.GetFilteredAsync(fullName, minAge, idRehearsalPoint);
        return entities.Select(e => new StaffDto
        {
            Id = e.Id,
            FullName = e.FullName,
            Address = e.Address,
            Experience = e.Experience,
            Phone = e.Phone,
            Age = e.Age,
            IdRehearsalPoint = e.IdRehearsalPoint
        });
    }
    public async Task<StaffDto> CreateAsync(StaffDto dto)
    {
        if (string.IsNullOrEmpty(dto.FullName) || string.IsNullOrEmpty(dto.Phone) || dto.Age <= 0)
            throw new ArgumentException("FullName, Phone, and Age are required and must be valid.");
        var entity = new Staff
        {
            FullName = dto.FullName,
            Address = dto.Address,
            Experience = dto.Experience,
            Phone = dto.Phone,
            Age = dto.Age,
            IdRehearsalPoint = dto.IdRehearsalPoint
        };
        var created = await _repository.AddAsync(entity);
        return new StaffDto
        {
            Id = created.Id,
            FullName = created.FullName,
            Address = created.Address,
            Experience = created.Experience,
            Phone = created.Phone,
            Age = created.Age,
            IdRehearsalPoint = created.IdRehearsalPoint
        };
    }
    public async Task UpdateAsync(int id, StaffDto dto)
    {
        if (string.IsNullOrEmpty(dto.FullName) || string.IsNullOrEmpty(dto.Phone) || dto.Age <= 0)
            throw new ArgumentException("FullName, Phone, and Age are required and must be valid.");
        var entity = await _repository.GetByIdAsync(id);
        if (entity == null)
            throw new KeyNotFoundException($"Staff with ID {id} not found.");
        entity.FullName = dto.FullName;
        entity.Address = dto.Address;
        entity.Experience = dto.Experience;
        entity.Phone = dto.Phone;
        entity.Age = dto.Age;
        entity.IdRehearsalPoint = dto.IdRehearsalPoint;
        await _repository.UpdateAsync(entity);
    }
    public async Task DeleteAsync(int id)
    {
        var entity = await _repository.GetByIdAsync(id);
        if (entity == null)
            throw new KeyNotFoundException($"Staff with ID {id} not found.");
        await _repository.DeleteAsync(id);
    }
}
```

Файл UserService.cs:

```
using RehearsalStudio.Application.DTOs;
using RehearsalStudio.Application.Interfaces;
using RehearsalStudio.Domain.Entities;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
namespace RehearsalStudio.Application.Services;
```

```
public class UserService : IUserService
{
    private readonly IUserRepository _repository;
    public UserService(IUserRepository repository)
    {
        _repository = repository;
    }
    public async Task<IEnumerable<UserDto>> GetAllAsync()
    {
        var entities = await _repository.GetAllAsync();
        return entities.Select(e => new UserDto
        {
            Id = e.Id,
            FullName = e.FullName,
            Phone = e.Phone,
            Email = e.Email,
            RegistrationDate = e.RegistrationDate
        });
    }
    public async Task<UserDto?> GetByIdAsync(int id)
    {
        var entity = await _repository.GetByIdAsync(id);
        if (entity == null) return null;
        return new UserDto
        {
            Id = entity.Id,
            FullName = entity.FullName,
            Phone = entity.Phone,
            Email = entity.Email,
            RegistrationDate = entity.RegistrationDate
        };
    }
    public async Task<IEnumerable<UserDto>> GetFilteredAsync(string? fullName, string? email)
    {
        var entities = await _repository.GetFilteredAsync(fullName, email);
        return entities.Select(e => new UserDto
        {
            Id = e.Id,
            FullName = e.FullName,
            Phone = e.Phone,
            Email = e.Email,
            RegistrationDate = e.RegistrationDate
        });
    }
    public async Task<UserDto> CreateAsync(UserDto dto)
    {
        if (string.IsNullOrEmpty(dto.FullName) || string.IsNullOrEmpty(dto.Phone) || string.IsNullOrEmpty(dto.Email))
            throw new ArgumentException("FullName, Phone, and Email are required.");
        var entity = new User
        {
            FullName = dto.FullName,
            Phone = dto.Phone,
            Email = dto.Email,
            RegistrationDate = dto.RegistrationDate
        };
        var created = await _repository.AddAsync(entity);
        return new UserDto
        {
            Id = created.Id,
            FullName = created.FullName,
            Phone = created.Phone,
            Email = created.Email,
            RegistrationDate = created.RegistrationDate
        };
    }
    public async Task UpdateAsync(int id, UserDto dto)
    {
        if (string.IsNullOrEmpty(dto.FullName) || string.IsNullOrEmpty(dto.Phone) || string.IsNullOrEmpty(dto.Email))
            throw new ArgumentException("FullName, Phone, and Email are required.");
        var entity = await _repository.GetByIdAsync(id);
        if (entity == null)
            throw new KeyNotFoundException($"User with ID {id} not found.");
        entity.FullName = dto.FullName;
        entity.Phone = dto.Phone;
        entity.Email = dto.Email;
        entity.RegistrationDate = dto.RegistrationDate;
        await _repository.UpdateAsync(entity);
    }
    public async Task DeleteAsync(int id)
    {
        var entity = await _repository.GetByIdAsync(id);
        if (entity == null)
            throw new KeyNotFoundException($"User with ID {id} not found.");
        await _repository.DeleteAsync(id);
    }
}
```

Файл Class1.cs:

```
namespace RehearsalStudio.Domain;
public class Class1
{
}
```

Файл Booking.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
namespace RehearsalStudio.Domain.Entities;
public class Booking
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int Id { get; set; }
    [Required]
    public DateTime Time { get; set; }
    public int? Duration { get; set; }
    [Required]
    public int Cost { get; set; }
    [Required]
    public DateTime CreationDate { get; set; }
    [Required]
    public string Status { get; set; } = string.Empty;
    [Required]
    public int NumberOfPeople { get; set; }
    public int? IdRoom { get; set; }
    [ForeignKey("IdRoom")]
    public Room? Room { get; set; }
    public int? IdUser { get; set; }
    [ForeignKey("IdUser")]
    public User? User { get; set; }
    public List<ServiceBooking> ServiceBookings { get; set; } = new();
    public List<EquipmentBooking> EquipmentBookings { get; set; } = new();
}
```

Файл Equipment.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
namespace RehearsalStudio.Domain.Entities;
public class Equipment
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int Id { get; set; }
    [Required]
    public string Name { get; set; } = string.Empty;
    [Required]
    public string Type { get; set; } = string.Empty;
    [Required]
    public string Brand { get; set; } = string.Empty;
    [Required]
    public string Model { get; set; } = string.Empty;
    [Required]
    public string Condition { get; set; } = string.Empty;
    public int? IdRehearsalPoint { get; set; }
    [ForeignKey("IdRehearsalPoint")]
    public RehearsalPoint? RehearsalPoint { get; set; }
    public List<EquipmentBooking> EquipmentBookings { get; set; } = new();
}
```

Файл EquipmentBooking.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
namespace RehearsalStudio.Domain.Entities;
public class EquipmentBooking
{
    [Key]
    [Column(Order = 0)]
    public int IdEquipment { get; set; }
    [Key]
    [Column(Order = 1)]
    public int IdBooking { get; set; }
    [ForeignKey("IdEquipment")]
    public Equipment? Equipment { get; set; }
    [ForeignKey("IdBooking")]
    public Booking? Booking { get; set; }
}
```

Файл RehearsalPont.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
namespace RehearsalStudio.Domain.Entities;
public class RehearsalPoint
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int Id { get; set; }
    public float? Rating { get; set; }
    [Required]
    public string ContactNumber { get; set; } = string.Empty;
    public string Schedule { get; set; } = string.Empty;
    [Required]
    public string Name { get; set; } = string.Empty;
    [Required]
    public string Address { get; set; } = string.Empty;
    public List<Room> Rooms { get; set; } = new();
    public List<Service> Services { get; set; } = new();
    public List<Equipment> Equipment { get; set; } = new();
    public List<Staff> Staff { get; set; } = new();
}
```

Файл Room.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
namespace RehearsalStudio.Domain.Entities;
public class Room
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int Id { get; set; }
    [Required]
    public string Name { get; set; } = string.Empty;
    public bool AirConditioner { get; set; } = false;
    [Required]
    public int Price { get; set; }
    public bool RecordingSupport { get; set; } = false;
    [Required]
    public int Area { get; set; }
    public int? IdRehearsalPoint { get; set; }
    [ForeignKey("IdRehearsalPoint")]
    public RehearsalPoint? RehearsalPoint { get; set; }
    public List<Booking> Bookings { get; set; } = new();
}
```

Файл Service.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
namespace RehearsalStudio.Domain.Entities;
public class Service
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int Id { get; set; }
    [Required]
```

```csharp
    public string Name { get; set; } = string.Empty;
    [Required]
    public int Price { get; set; }
    [Required]
    public string Type { get; set; } = string.Empty;
    public string? Requirements { get; set; }
    public int? IdRehearsalPoint { get; set; }
    [ForeignKey("IdRehearsalPoint")]
    public RehearsalPoint? RehearsalPoint { get; set; }
    public List<ServiceBooking> ServiceBookings { get; set; } = new();
}
```

Файл ServiceBooking.cs:

```csharp
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
namespace RehearsalStudio.Domain.Entities;
public class ServiceBooking
{
    [Key]
    [Column(Order = 0)]
    public int IdService { get; set; }
    [Key]
    [Column(Order = 1)]
    public int IdBooking { get; set; }
    [ForeignKey("IdService")]
    public Service? Service { get; set; }
    [ForeignKey("IdBooking")]
    public Booking? Booking { get; set; }
}
```

Файл Staff.cs:

```csharp
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
namespace RehearsalStudio.Domain.Entities;
public class Staff
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int Id { get; set; }
    [Required]
    public string FullName { get; set; } = string.Empty;
    public string? Address { get; set; }
    public int? Experience { get; set; }
    [Required]
    public string Phone { get; set; } = string.Empty;
    [Required]
    public int Age { get; set; }
    public int? IdRehearsalPoint { get; set; }
    [ForeignKey("IdRehearsalPoint")]
    public RehearsalPoint? RehearsalPoint { get; set; }
}
```

Файл User.cs:

```csharp
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
namespace RehearsalStudio.Domain.Entities;
public class User
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int Id { get; set; }
    [Required]
    public string FullName { get; set; } = string.Empty;
    [Required]
    public string Phone { get; set; } = string.Empty;
    [Required]
    public string Email { get; set; } = string.Empty;
    [Required]
    public DateTime RegistrationDate { get; set; }
    public List<Booking> Bookings { get; set; } = new();
}
```

Файл .NETCoreApp,Version=v9.0.AssemblyAttributes.cs:

```
// <autogenerated />
using System;
using System.Reflection;
[assembly: global::System.Runtime.Versioning.TargetFrameworkAttribute(".NETCoreApp,Version=v9.0", FrameworkDisplayName = ".NET 9.0")]
```

Файл RehearsalStudio.Domain.AssemblyInfo.cs:

```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------
using System;
using System.Reflection;
[assembly: System.Reflection.AssemblyCompanyAttribute("RehearsalStudio.Domain")]
[assembly: System.Reflection.AssemblyConfigurationAttribute("Debug")]
[assembly: System.Reflection.AssemblyFileVersionAttribute("1.0.0.0")]
[assembly: System.Reflection.AssemblyInformationalVersionAttribute("1.0.0")]
[assembly: System.Reflection.AssemblyProductAttribute("RehearsalStudio.Domain")]
[assembly: System.Reflection.AssemblyTitleAttribute("RehearsalStudio.Domain")]
[assembly: System.Reflection.AssemblyVersionAttribute("1.0.0.0")]
// Generated by the MSBuild WriteCodeFragment class.
```

Файл RehearsalStudio.Domain.GlobalUsings.g.cs:

```
// <auto-generated/>
global using global::System;
global using global::System.Collections.Generic;
global using global::System.IO;
global using global::System.Linq;
global using global::System.Net.Http;
global using global::System.Threading;
global using global::System.Threading.Tasks;
```

Файл Class1.cs:

```
namespace RehearsalStudio.Infrastructure;
public class Class1
{
}
```

Файл .NETCoreApp,Version=v9.0.AssemblyAttributes.cs:

```
// <autogenerated />
using System;
using System.Reflection;
[assembly: global::System.Runtime.Versioning.TargetFrameworkAttribute(".NETCoreApp,Version=v9.0", FrameworkDisplayName = ".NET 9.0")]
```

Файл RehearsalStudio.Infrastructure.AssemblyInfo.cs:

```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------
using System;
using System.Reflection;
[assembly: System.Reflection.AssemblyCompanyAttribute("RehearsalStudio.Infrastructure")]
[assembly: System.Reflection.AssemblyConfigurationAttribute("Debug")]
[assembly: System.Reflection.AssemblyFileVersionAttribute("1.0.0.0")]
[assembly: System.Reflection.AssemblyInformationalVersionAttribute("1.0.0")]
[assembly: System.Reflection.AssemblyProductAttribute("RehearsalStudio.Infrastructure")]
[assembly: System.Reflection.AssemblyTitleAttribute("RehearsalStudio.Infrastructure")]
[assembly: System.Reflection.AssemblyVersionAttribute("1.0.0.0")]
// Generated by the MSBuild WriteCodeFragment class.
```

Файл RehearsalStudio.Infrastructure.GlobalUsings.g.cs:

```
// <auto-generated/>
global using global::System;
global using global::System.Collections.Generic;
global using global::System.IO;
global using global::System.Linq;
global using global::System.Net.Http;
global using global::System.Threading;
global using global::System.Threading.Tasks;
```