Disynergy
Страница материала ↗

EXQT
Готовые ответы — на EXQT
Доступ к ответам приобретается отдельно.
Найти ответы ↗

Методические рекомендации. Теория

ОГЛАВЛЕНИЕ РАЗДЕЛА 1.1

- 1.1. Основы информационных систем и сопровождения
- 1.1.1. Понятие информационной системы
- 1.1.2. Жизненный цикл информационной системы
- 1.1.3. Основные задачи сопровождения ИС
- 1.1.4. Понятие архитектуры информационной системы

Ссылка

Ссылка на интерактивный документ методички: https://olgakraven.github.io/DKIP.PP.PM.06-method/
Данная ссылка представляет собой html-документ с гиперссылками, для более удобной навигацией.

1.1. Основы информационных систем и сопровождения

Раздел посвящён базовым понятиям об информационных системах, их жизненном цикле, задачах сопровождения и архитектуре. Материал служит основой для понимания последующих тем по разработке и обслуживанию Web API.

1.1.1. Понятие информационной системы

Информационная система (ИС) — это совокупность программных, технических, организационных и информационных компонентов, предназначенных для сбора, хранения, обработки и предоставления данных пользователям для решения их задач.

Информационная система обеспечивает автоматизацию процессов, повышение точности и скорости обработки данных, снижение затрат и повышение эффективности деятельности предприятия.

Компоненты информационной системы

1. **Аппаратное обеспечение (hardware)**
   Серверы, рабочие станции, сетевое оборудование, устройства хранения.
2. **Программное обеспечение (software)**

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

- системное ПО (ОС, СУБД);
- прикладные программы;
- сервисы и модули ИС.

3. **Данные**
   Структурированные и неструктурированные сведения, которые система хранит и обрабатывает.
4. **Пользователи**
   Конечные пользователи, администраторы, разработчики.
5. **Процедуры и регламенты**
   Описание процессов, правил взаимодействия, политики безопасности, инструкции эксплуатации.

1.1.2. Жизненный цикл информационной системы

Жизненный цикл ИС включает полный путь — от проектирования и создания до модернизации и вывода из эксплуатации.

Основные этапы жизненного цикла

1. **Разработка**
   - анализ требований;
   - проектирование архитектуры;
   - программирование модулей;
   - тестирование;
   - подготовка документации.
2. **Внедрение**
   - установка и настройка ПО;
   - миграция данных;
   - обучение пользователей;
   - пилотная эксплуатация.
3. **Сопровождение**
   - поддержание работоспособности системы после запуска;
   - обновления, исправления, мониторинг, оптимизация.
4. **Модернизация**
   - расширение функционала;
   - оптимизация производительности;
   - переход на новую платформу или архитектуру;
   - улучшение интерфейсов.

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

> Жизненный цикл может быть непрерывным (DevOps, CI/CD), что обеспечивает постоянное развитие системы.

1.1.3. Основные задачи сопровождения информационных систем

Сопровождение ИС — это комплекс мероприятий, направленных на обеспечение стабильной, безопасной и эффективной работы системы после её внедрения.

1. **Развертывание**
   - подготовка инфраструктуры;
   - установка сервисов, зависимостей, БД;
   - конфигурация окружений (prod/test/dev);
   - автоматизация развертывания (Docker, CI/CD).
2. **Обновление версии**
   - установка патчей и исправлений ошибок;
   - миграция схемы базы данных;
   - проверка совместимости модулей;
   - откат версий при необходимости (rollback).
3. **Мониторинг работоспособности**
   Контроль технических параметров:
   - нагрузка на CPU, RAM и сеть;
   - время отклика API;
   - состояние сервисов;
   - журнал ошибок.

   Используемые инструменты: Prometheus, Grafana, Zabbix, ELK.
4. **Логирование**
   - сбор логов приложений и системных событий;
   - анализ ошибок и предупреждений;
   - настройка централизованного логирования;
   - обеспечение трассировки запросов.
5. **Резервное копирование**
   - создание бекапов БД и файлов;
   - проверка восстановления;

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

- выбор стратегии (полное, инкрементальное, дифференциальное копирование);
- хранение копий в безопасных хранилищах.

6. **Устранение инцидентов**
   - обработка обращений пользователей;
   - анализ первопричин (root cause analysis);
   - устранение сбоев в работе модулей;
   - декомпозиция ошибок на задачи для разработчиков.

> **Важно**
>
> Сопровождение является критическим этапом жизненного цикла ИС, обеспечивающим непрерывность бизнеса и надёжность работы системы.

1.1.4. Понятие архитектуры информационной системы

Архитектура ИС — это логическая структура системы, определяющая, как её компоненты взаимодействуют, где хранятся данные и как пользователи получают доступ к функциям.

Основные архитектурные модели

1) Клиент–серверная архитектура

- Клиент отправляет запросы.
- Сервер обрабатывает запросы, выполняет бизнес-логику и работает с базой данных.
- Может быть двух- и трёхзвенной (client → server → database).

Плюсы: масштабируемость, централизованное управление.
Минусы: необходимость постоянного соединения.

2) Веб-сервисы

Используются для обмена данными между системами через интернет.

Типы:

- REST (обычно через JSON);
- SOAP (через XML);
- gRPC (оптимизированный бинарный формат).

Веб-сервисы обеспечивают интеграцию модулей и микросервисов.

3) API (Application Programming Interface)

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

API — это интерфейс, через который другие приложения могут обращаться к функционалу системы.

Основные виды:

- REST API — самый распространённый способ;
- GraphQL API — гибкий запрос данных;
- WebSocket API — обмен данными в реальном времени.

API является ключевым элементом современной архитектуры, позволяющим интегрировать системы и строить микросервисы.

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Методические рекомендации. Основы C# и ASP.NET Core

ОГЛАВЛЕНИЕ РАЗДЕЛА 1.2

- 1.2. Основы разработки на C# и ASP.NET Core
- 1.2.1. Основы языка программирования C#
- 1.2.2. Основы ASP.NET Core
- 1.2.3. CRUD-операции
- 1.2.4. Форматы обмена данными: JSON и сериализация

1.2. Основы разработки на C# и ASP.NET Core

Раздел описывает базовые возможности языка C# и фреймворка ASP.NET Core, необходимые для разработки Web API с CRUD-функционалом.

1.2.1. Основы языка программирования C#

C# — современный объектно-ориентированный язык, разработанный корпорацией Microsoft. Он используется для разработки web-приложений, сервисов, десктопных программ, игр (Unity), мобильных приложений и корпоративных систем.

Типы данных

C# поддерживает множество встроенных типов:

1. Числовые: `int`, `long`, `float`, `double`, `decimal`.
2. Логические: `bool`.
3. Строковые: `string`, `char`.
4. Дата и время: `DateTime`.
5. Ссылочные типы: классы, интерфейсы, массивы и др.

Типы делятся на значимые и ссылочные, что важно для управления памятью и передачи параметров в методы.

Помогу с учебным заданием без предоплаты. Оплата после выполнения

Нажмите на мессенджер, чтобы написать

Telegram ↗ WhatsApp ↗ MAX ↗

+7 (924) 305-23-08 · Пишите в мессенджер

disynergy.ru

Классы, методы, свойства

```
public class User
{
    public int Id { get; set; }        // Свойство
    public string Name { get; set; }

    public void SayHello()             // Метод
    {
        Console.WriteLine($"Привет, {Name}!");
    }
}
```

Класс — основной элемент объектно-ориентированной модели. Свойства используются для хранения данных, методы — для выполнения логики.

Ключевые концепции: инкапсуляция, наследование, полиморфизм.

Коллекции

Для хранения наборов данных используются:

- `List<T>` — список;
- `Dictionary<TKey, TValue>` — словарь;
- `Queue<T>` — очередь;
- `HashSet<T>` — множество.

```
var users = new List<string> { "Kate", "John" };
```

Коллекции активно используются в Web API для хранения, фильтрации и обработки данных.

Async / Await (базово)

Асинхронность позволяет выполнять операции без блокировки потоков, что критично для web-сервисов.

```
public async Task<string> LoadData()
{
    await Task.Delay(1000);               // имитация задержки
    return "готово";
}
```

`async/await` упрощает работу с длительными операциями: доступ к БД, HTTP-запросы, файловые операции.

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

1.2.2. Основы ASP.NET Core

ASP.NET Core — кроссплатформенный фреймворк для разработки web-приложений и API. Он построен вокруг минимализма, гибкости и встроенной поддержки Dependency Injection.

Структура Web API

Типичный проект включает директории и файлы:

```
/Controllers      — контроллеры
/Models           — модели домена
/DTOs             — объекты передачи данных
/Services         — бизнес-логика
Program.cs        — конфигурация приложения
appsettings.json  — настройки
```

Обработка запроса строится по цепочке: `middleware → контроллер → сервис → база данных → ответ`.

Контроллеры и маршруты

```
[ApiController]
[Route("api/[controller]")]
public class UsersController : ControllerBase
{
    [HttpGet]
    public IActionResult GetAll() => Ok("Список пользователей");
}
```

Маршруты задаются через атрибуты:

- `[HttpGet]`
- `[HttpPost]`
- `[HttpPut]`
- `[HttpDelete]`

Модели (Models)

```
public class User
{
    public int Id { get; set; }
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

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

Модели отражают структуру таблиц БД или сущностей предметной области.

DTO (Data Transfer Objects)

```
public class CreateUserDto
{
    public string Email { get; set; }
    public string Password { get; set; }
}
```

DTO используют для передачи данных между клиентом и сервером, чтобы скрывать лишние поля модели, отделять внутреннюю структуру от внешнего API и применять валидацию.

Валидация DTO

```
public class RegisterDto
{
    [Required]
    [EmailAddress]
    public string Email { get; set; }

    [MinLength(6)]
    public string Password { get; set; }
}
```

Ошибки автоматически возвращаются в формате JSON.

Dependency Injection (DI)

```
// Регистрация сервиса
services.AddScoped<IUserService, UserService>();

// Использование в контроллере
public class UsersController : ControllerBase
{
    private readonly IUserService _service;

    public UsersController(IUserService service)
    {
        _service = service;
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

```
    }
}
```

В ASP.NET Core DI встроен по умолчанию. Это улучшает тестируемость, расширяемость и читабельность приложения.

1.2.3. CRUD-операции

CRUD — это базовые операции работы с данными: создание, чтение, обновление и удаление.

Операция	HTTP-метод	Назначение
Create	POST	Создание записи
Read	GET	Получение данных
Update	PUT / PATCH	Изменение существующей записи
Delete	DELETE	Удаление записи

```
[HttpPost]
public IActionResult Create(User user) => Created(...);

[HttpGet("{id}")]
public IActionResult Get(int id) => Ok(...);

[HttpPut("{id}")]
public IActionResult Update(int id, User user) => NoContent();

[HttpDelete("{id}")]
public IActionResult Delete(int id) => NoContent();
```

Замечание

CRUD-операции — основа любого Web API и типовой учебный пример.

1.2.4. Форматы обмена данными: JSON и сериализация

JSON (JavaScript Object Notation)

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

Основной формат для обмена данными между клиентом и сервером:

```
{
  "id": 1,
  "email": "test@mail.com"
}
```

ASP.NET Core автоматически конвертирует JSON → объект C# и обратно.

Сериализация / десериализация

Используется библиотека `System.Text.Json`.

```
var json = JsonSerializer.Serialize(user);          // сериализация
var user = JsonSerializer.Deserialize<User>(json);  // десериализация
```

JSON удобен, легко читаем, кроссплатформенный и поддерживается всеми современными фреймворками.

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Методические рекомендации. Работа с базами данных

ОГЛАВЛЕНИЕ РАЗДЕЛА 1.3

- 1.3. Работа с базами данных
- 1.3.1. Понятие СУБД и таблиц
- 1.3.2. Основы SQL (PostgreSQL)
- 1.3.3. Первичный ключ и связи между таблицами
- 1.3.4. Миграции (EF Core)
- 1.3.5. ORM Entity Framework Core

1.3. Работа с базами данных

Работа с базами данных — один из ключевых элементов разработки информационных систем. В рамках создания Web API студент должен понимать основы реляционных СУБД, язык SQL и базовые возможности ORM Entity Framework Core.

1.3.1. Понятие СУБД и таблиц

СУБД (система управления базами данных) — программное обеспечение, которое обеспечивает создание, хранение, изменение и управление данными.

Основные функции СУБД

- хранение данных в структурированном виде;
- обеспечение целостности данных;
- выполнение запросов (SQL);
- управление пользователями и правами доступа;
- резервное копирование и восстановление.

Примеры СУБД: PostgreSQL, MySQL, SQL Server, SQLite.

Таблица

Помогу с учебным заданием без предоплаты. Оплата после выполнения

Нажмите на мессенджер, чтобы написать

Telegram ↗ WhatsApp ↗ MAX ↗

+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

Таблица — базовая структура реляционной базы данных, хранящая данные в виде строк и столбцов.

- Столбец — поле определённого типа данных.
- Строка — отдельная запись (объект сущности).
- Схема БД — набор таблиц и их связей.

Пример таблицы `Users`:

Id	Name	Email
1	Kate	kate@mail.com
2	John	john@mail.com

1.3.2. Основы SQL (PostgreSQL)

SQL — язык запросов для работы с реляционными данными. Студент должен владеть основными операциями.

SELECT — выборка данных

```
SELECT * FROM Users;

SELECT Id, Email
FROM Users
WHERE Id = 1;
```

INSERT — вставка данных

```
INSERT INTO Users (Name, Email)
VALUES ('Ivan', 'ivan@mail.com');
```

UPDATE — обновление данных

```
UPDATE Users
SET Email = 'new@mail.com'
WHERE Id = 1;
```

DELETE — удаление данных

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

```
DELETE FROM Users
WHERE Id = 3;
```

1.3.3. Первичный ключ и связи между таблицами

Первичный ключ (Primary Key, PK)

PK — колонка (или набор колонок), уникально идентифицирующая запись в таблице.

```
Id SERIAL PRIMARY KEY
```

Типы связей

1. **1–N (один ко многим)**
   Один пользователь → много заказов.
   Пример:
   - таблица `Users` (1);
   - таблица `Orders` (N);
   - `Orders.UserId` → внешний ключ (FK) на `Users.Id`.
2. **N–M (многие ко многим)**
   Много студентов → много курсов. Требуется промежуточная таблица:
   - `Students`
   - `Courses`
   - `StudentCourses (StudentId, CourseId)`
3. **1–1 (один к одному)**
   Используется реже, например для вынесения дополнительных данных в отдельную таблицу.

1.3.4. Миграции (если используется EF Core)

Миграции — механизм, позволяющий изменять структуру базы данных через код.

Миграции используются для:

- автоматического создания таблиц;
- добавления новых столбцов;
- изменения схемы;
- контроля версий базы.

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

```
SCRIPT

dotnet ef migrations add InitialCreate
dotnet ef database update
```

Команды вызываются из консоли и отражают изменения модели данных в структуре БД.

1.3.5. ORM Entity Framework Core

Entity Framework Core (EF Core) — ORM-фреймворк, позволяющий работать с базой данных через объекты C#.

ORM выполняет:

- отображение классов на таблицы;
- генерацию SQL-команд;
- отслеживание изменений;
- поддержку миграций.

DbContext

```
public class AppDbContext : DbContext
{
    public DbSet<User> Users { get; set; }

    public AppDbContext(DbContextOptions<AppDbContext> options)
        : base(options) { }
}
```

`DbContext` управляет подключением к базе, формирует SQL и выполняет операции с данными.

DbSet

```
public DbSet<User> Users { get; set; }

// Примеры операций:
context.Users.Add(user);
context.Users.Remove(user);
var list = context.Users.ToList();
```

Миграции EF Core

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Миграции EF Core используются для автоматического создания или обновления структуры БД:

```
SCRIPT

dotnet ef migrations add AddedOrders
dotnet ef database update
```

Конфигурация подключения

Подключение к PostgreSQL задаётся в `appsettings.json`:

```
{
  "ConnectionStrings": {
    "Default": "Host=localhost;Port=5432;Database=exampledb;Username=postgres;Password=1234"
  }
}
```

Регистрация контекста в `Program.cs`:

```
builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseNpgsql(builder.Configuration.GetConnectionString("Default")));
```

Для работы с PostgreSQL требуются пакеты:

- `Microsoft.EntityFrameworkCore.Design`
- `Npgsql.EntityFrameworkCore.PostgreSQL`

Практический совет

Перед выполнением миграций убедитесь, что строка подключения корректна и база данных доступна из среды разработки.

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Методические рекомендации. Основы REST API

ОГЛАВЛЕНИЕ РАЗДЕЛА 1.4

- 1.4. Основы REST API
- 1.4.1. Ключевые принципы REST
- 1.4.2. Структура HTTP-запросов
- 1.4.3. Пример REST-запроса
- 1.4.4. Swagger / OpenAPI
- 1.4.5. OpenAPI

1.4. Основы REST API

REST (Representational State Transfer) — архитектурный стиль построения web-сервисов, основанный на использовании стандартных методов HTTP и представления данных в виде ресурсов. REST API используется в большинстве современных веб-приложений, мобильных сервисов и микросервисов.

1.4.1. Ключевые принципы REST

REST основывается на наборе концепций, определяющих, как должен быть устроен API.

1. Ресурсы

Ресурс — это объект или сущность, на которую можно выполнять операции.

Примеры ресурсов:

- `/users` — пользователи;
- `/orders` — заказы;
- `/products` — товары.

Каждый ресурс имеет уникальный URL (endpoint) и может включать подресурсы:

- `/users/5/orders` — заказы пользователя с `id = 5`.

Помогу с учебным заданием без предоплаты. Оплата после выполнения

Нажмите на мессенджер, чтобы написать

Telegram ↗ WhatsApp ↗ MAX ↗

+7 (924) 305-23-08 · Пишите в мессенджер

disynergy.ru

> Ресурс — это *что*, а HTTP-метод — *что с ним делать*.

2. Методы HTTP

Методы HTTP определяют операцию над ресурсом.

Метод	Назначение	REST-операция
GET	Получение данных	Read
POST	Создание нового ресурса	Create
PUT	Полное обновление ресурса	Update
PATCH	Частичное обновление	Partial Update
DELETE	Удаление ресурса	Delete

Примеры:

- `GET /api/users` — получить список пользователей;
- `POST /api/users` — создать пользователя;
- `PUT /api/users/3` — обновить пользователя №3;
- `DELETE /api/users/5` — удалить пользователя №5.

3. Статусы ответов

REST активно использует стандартные HTTP-коды:

Код	Назначение
200 OK	успешный запрос
201 Created	успешно создан новый объект
204 No Content	успешный запрос без тела ответа
400 Bad Request	ошибка клиента, неверный формат данных
401 Unauthorized	нет авторизации
403 Forbidden	доступ запрещён
404 Not Found	ресурс не найден
409 Conflict	конфликт данных (например, уникальность)

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Код	Назначение
500 Internal Server Error	ошибка сервера

> **Важно**
>
> Корректные статусы помогают клиенту понимать, что произошло, и упрощают обработку ошибок на стороне фронтенда и других сервисов.

1.4.2. Структура HTTP-запросов

Запросы в REST API состоят из трёх ключевых частей: заголовки, тело и параметры строки запроса.

1. Headers (заголовки)

Заголовки используются для передачи дополнительных параметров:

- `Content-Type: application/json` — тип данных;
- `Authorization: Bearer <token>` — токен доступа;
- `Accept: application/json` — ожидаемый формат ответа.

```
SCRIPT

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5c...
```

2. Body (тело запроса)

Передаётся в `POST` / `PUT` / `PATCH` запросах. Обычно в формате JSON.

```
{
  "email": "user@mail.com",
  "password": "123456"
}
```

Для `GET` и `DELETE` тело, как правило, не используется.

3. Query parameters

Используются для фильтрации, поиска, сортировки, пагинации, например:

```
GET /api/products?category=books&sort=price&page=2
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Примеры параметров:

- `?search=phone`
- `?limit=20`
- `?isActive=true`

1.4.3. Пример REST-запроса

Задача: создать нового пользователя.

```
SCRIPT

POST /api/users
Content-Type: application/json

{
  "name": "Alex",
  "email": "alex@mail.com"
}
```

Ответ от сервера:

```
HTTP/1.1 201 Created
Location: /api/users/15
```

> **Замечание**
>
> Полезно возвращать заголовок `Location` с URL созданного ресурса, чтобы клиент мог сразу обратиться к нему повторно.

1.4.4. Swagger / OpenAPI

Swagger — это инструмент, который генерирует интерактивную документацию REST API.

Он позволяет:

- автоматически отображать все эндпоинты;
- показывать модель данных (DTO);
- отправлять тестовые запросы прямо из браузера;
- сохранять структуру API в формате OpenAPI (JSON/YAML);

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

- использовать автогенерацию клиентов.

Swagger облегчает тестирование и демонстрацию API студентами.

1.4.4.1. Пример отображения метода в Swagger

Эндпоинт `POST /api/users` в Swagger UI будет иметь:

- кнопку `TRY IT OUT` для отправки запроса;
- пример JSON-запроса;
- описание параметров и схемы DTO;
- список возможных статусов ответов (200, 201, 400, 500 и др.).

1.4.4.2. Подключение Swagger в ASP.NET Core

```
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

app.UseSwagger();
app.UseSwaggerUI();
```

После запуска API документация будет доступна по адресу, например:

```
http://localhost:5000/swagger
```

1.4.5. OpenAPI (желательно)

OpenAPI — это формат описания API, позволяющий автоматически генерировать документацию, тесты и клиентов.

Обычно файл называется:

- `openapi.json`
- `openapi.yaml`

Он содержит:

- описание всех методов;
- модели данных;
- типы параметров;
- примеры запросов и ответов.

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Связь со Swagger UI

Swagger UI использует именно спецификацию OpenAPI для отображения интерактивной документации, поэтому корректное описание API в `openapi.json` или `openapi.yaml` критично для качества документации.

Помогу с учебным заданием без предоплаты. Оплата после выполнения

Нажмите на мессенджер, чтобы написать

Telegram ↗ WhatsApp ↗ MAX ↗

+7 (924) 305-23-08 · Пишите в мессенджер

disynergy.ru

Методические рекомендации. Основы Docker

ОГЛАВЛЕНИЕ РАЗДЕЛА 1.5

- 1.5. Основы Docker
- 1.5.1. Базовые понятия
- 1.5.2. Отличие контейнера от виртуальной машины
- 1.5.3. Основные команды Docker
- 1.5.4. Работа с томами (Volumes)

1.5. Основы Docker

Docker — это технология контейнеризации, которая позволяет изолированно запускать приложения вместе со всеми зависимостями, обеспечивая переносимость, простоту развертывания и воспроизводимость среды.

1.5.1. Базовые понятия

Docker

Docker — это платформа для упаковки, доставки и запуска приложений в изолированных контейнерах. Включает инструменты и сервисы:

- Docker Engine
- Docker CLI
- Docker Desktop
- Docker Hub

Контейнер

Контейнер — это легковесная изолированная среда, содержащая:

- само приложение,
- все зависимости,
- настройки окружения,
- собственную файловую систему (на основе образа).

Помогу с учебным заданием без предоплаты. Оплата после выполнения

Нажмите на мессенджер, чтобы написать

Telegram ↗ WhatsApp ↗ MAX ↗

+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

Изоляция реализуется через namespace и cgroups.

Образ (Image)

Образ — это шаблон контейнера, состоящий из слоёв. Содержит:

- исходные файлы приложения,
- библиотеки и зависимости,
- конфигурации,
- инструкции сборки.

> Образы создаются на основе Dockerfile.

Docker Engine

Управляет контейнерами:

- скачивание и хранение образов;
- создание контейнеров;
- управление сетью и томами;
- выполнение команд CLI.

Docker Hub

Публичный/приватный репозиторий образов:

- официальные образы (Nginx, Postgres, Redis),
- хранение собственных образов,
- совместная работа над проектами.

1.5.2. Отличие контейнера от виртуальной машины

Характеристика	Контейнер	Виртуальная машина
Ядро ОС	Использует ядро хостовой ОС	Каждая ВМ имеет собственное ядро
Скорость запуска	секунды	десятки секунд/минут
Вес	десятки МБ	гигабайты
Изоляция	процессная изоляция	полная виртуализация
Производительность	близка к нативной	ниже из-за гипервизора

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Характеристика	Контейнер	Виртуальная машина
Назначение	запуск приложений	запуск полноценной ОС

> Итог: контейнеры — лёгкие и быстрые. ВМ — тяжёлые, но полностью изолированные.

1.5.3. Основные команды Docker

1) Сборка образа

```
SCRIPT

docker build -t myapp .
```

- `-t` — имя образа
- `.` — путь к Dockerfile

2) Запуск контейнера

```
SCRIPT

docker run myapp
docker run -d -p 8080:80 myapp
```

- `-d` — фоновый режим
- `-p` — проброс портов

3) Просмотр работающих контейнеров

```
SCRIPT

docker ps
docker ps -a
```

4) Остановка контейнера

```
SCRIPT

docker stop my_container
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

5) Удаление контейнера

```
SCRIPT
docker rm my_container
```

6) Логи контейнера

```
SCRIPT
docker logs my_container
```

7) Просмотр образов

```
SCRIPT
docker images
```

8) Скачивание образа

```
SCRIPT
docker pull postgres
```

9) Публикация образа

```
SCRIPT
docker push myrepo/myapp
```

Предварительно требуется `docker login`.

1.5.4. Работа с томами (Volumes)

Назначение томов

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

Томы используются для хранения данных вне контейнера, чтобы:

- данные не удалялись после остановки контейнера,
- сохранять БД между перезапусками,
- мигрировать данные между версиями приложения.

Создание тома

```
SCRIPT

docker volume create myvolume
```

Использование тома при запуске

```
SCRIPT

docker run -v myvolume:/var/lib/postgresql/data postgres
```

Хранение данных

Томы хранятся на хостовой машине, например в Linux:

```
/var/lib/docker/volumes/
```

Резервное копирование тома

```
SCRIPT

docker run --rm -v myvolume:/data -v $(pwd):/backup \
  ubuntu tar cvf /backup/backup.tar /data
```

Восстановление тома

```
SCRIPT

docker run --rm -v myvolume:/data -v $(pwd):/backup \
  ubuntu tar xvf /backup/backup.tar -C /
```

Важно

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Тома являются ключевым механизмом Docker для долговременного хранения данных, особенно при использовании контейнеров с СУБД.

Помогу с учебным заданием без предоплаты. Оплата после выполнения

Нажмите на мессенджер, чтобы написать

Telegram ↗ WhatsApp ↗ MAX ↗

+7 (924) 305-23-08 · Пишите в мессенджер

disynergy.ru

1.6. Dockerfile и контейнеризация приложений

ОГЛАВЛЕНИЕ РАЗДЕЛА 1.6

- 1.6. Dockerfile и контейнеризация приложений
- 1.6.1. Структура Dockerfile
- 1.6.2. Multi-stage build в .NET
- 1.6.3. Пример полного Dockerfile для ASP.NET Core Web API
- 1.6.4. Создание собственного образа

1.6. Dockerfile и контейнеризация приложений

Dockerfile — это сценарий сборки образа Docker. Он содержит набор инструкций, которые определяют, какие слои будут включены в конечный образ и как будет запускаться приложение.

Для ASP.NET Core и .NET проектов Dockerfile является ключевым элементом при создании переносимых Web API-сервисов.

1.6.1. Структура Dockerfile

Dockerfile — это текстовый файл, который описывает последовательность инструкций. Основные инструкции:

1) FROM — базовый образ

Определяет, на основе какого образа создаётся новый:

```
FROM mcr.microsoft.com/dotnet/aspnet:8.0
```

Типичные базовые образы для .NET:

- `mcr.microsoft.com/dotnet/sdk:8.0` — для сборки;
- `mcr.microsoft.com/dotnet/aspnet:8.0` — для запуска;
- `ubuntu`, `alpine` — минимальные образы Linux.

2) WORKDIR — рабочая директория

Помогу с учебным заданием без предоплаты. Оплата после выполнения

Нажмите на мессенджер, чтобы написать

Telegram ↗ WhatsApp ↗ MAX ↗

+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

Определяет папку внутри контейнера, в которой будут выполняться команды:

```
WORKDIR /app
```

Все последующие команды выполняются относительно этой директории.

3) COPY — копирование файлов в контейнер

```
COPY . .
```

Инструкция `COPY` используется для переноса исходного кода, бинарных файлов или конфигураций в образ.

Типичные варианты:

```
COPY MyApi.csproj .
COPY . /app
```

4) RUN — выполнение команд внутри контейнера

Команда выполняется на этапе сборки образа, а не при запуске контейнера.

```
RUN dotnet restore
RUN apt-get update && apt-get install -y curl
```

5) CMD и ENTRYPOINT — команда запуска

`CMD` задаёт команду по умолчанию, а `ENTRYPOINT` фиксирует исполняемый процесс.

```
CMD ["dotnet", "MyApi.dll"]
ENTRYPOINT ["dotnet", "MyApi.dll"]
```

Разница между инструкциями:

Инструкция	Можно переопределить	Использование
CMD	Да	Параметры и команда по умолчанию
ENTRYPOINT	Нет (без дополнительных параметров)	Обязательная команда запуска контейнера

6) EXPOSE — объявление порта

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

Указывает, какой порт используется приложением внутри контейнера:

```
EXPOSE 8080
```

Важно: `EXPOSE` не открывает порт наружу — это лишь декларация для документации и вспомогательных инструментов.

Проброс порта осуществляется при запуске:

```
docker run -p 8080:8080 image
```

1.6.2. Multi-stage build в .NET

Многоэтапная сборка (multi-stage build) используется для:

- уменьшения размера образа;
- отделения этапа сборки от этапа выполнения;
- повышения безопасности (исходники остаются только в первом этапе).

Этап 1 — SDK (сборка)

Используется образ SDK, который включает компилятор и утилиты.

```
FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
WORKDIR /src

COPY MyApi.csproj .
RUN dotnet restore

COPY . .
RUN dotnet publish -c Release -o /app/out
```

Этап 2 — Runtime (запуск)

Используется минимальный образ, содержащий только runtime.

```
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS runtime
WORKDIR /app

COPY --from=build /app/out .

ENTRYPOINT ["dotnet", "MyApi.dll"]
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Преимущества multi-stage build

Преимущество	Описание
Маленький размер образа	В итоговом образе только runtime (~80 MB вместо 700+ MB SDK)
Повышенная безопасность	Исходный код и инструменты сборки не попадают в финальный образ
Производительность CI/CD	Быстрее передача и запуск образа в разных средах
Чистота структуры	В финальном слое только собранные артефакты (DLL, конфигурации)

1.6.3. Пример полного Dockerfile для ASP.NET Core Web API

```
# Этап 1 — сборка
FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
WORKDIR /src

COPY *.csproj .
RUN dotnet restore

COPY . .
RUN dotnet publish -c Release -o /app/publish

# Этап 2 — запуск
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS final
WORKDIR /app

COPY --from=build /app/publish .

EXPOSE 8080
ENTRYPOINT ["dotnet", "MyApi.dll"]
```

Комментарий

В учебных проектах достаточно одного Dockerfile на корне Web API. При смене имени сборки (MyApi.dll) необходимо обновить его в инструкции ENTRYPOINT.

1.6.4. Создание собственного образа

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

1. Сборка образа

```
SCRIPT

docker build -t myapi .
```

Здесь `myapi` — имя образа, а `.` — текущая директория с Dockerfile.

2. Проверка списка образов

```
SCRIPT

docker images
```

3. Запуск контейнера с API

```
SCRIPT

docker run -d -p 8080:8080 --name api myapi
```

- `-d` — запуск в фоновом режиме;
- `-p 8080:8080` — проброс порта контейнера на хост;
- `--name api` — имя контейнера.

4. Остановка контейнера

```
SCRIPT

docker stop api
```

5. Удаление контейнера

```
SCRIPT

docker rm api
```

6. Публикация образа в Docker Hub

```
SCRIPT

docker login
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

```
docker tag myapi username/myapi
docker push username/myapi
```

Вместо `username` необходимо указать свой логин в Docker Hub. После публикации образ можно использовать в других средах и на серверах.

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

1.7. Docker Compose (если используется БД или несколько сервисов)

ОГЛАВЛЕНИЕ РАЗДЕЛА 1.7

- 1.7. Docker Compose (общие сведения)
- 1.7.1. Что такое Docker Compose и для чего оно применяется
- 1.7.2. Структура файла docker-compose.yml
- 1.7.3. Связь нескольких контейнеров
- 1.7.4. Пример полноценного docker-compose.yml
- 1.7.5. Основные команды Docker Compose

1.7. Docker Compose (если используется БД или несколько сервисов)

Docker Compose — инструмент для описания и совместного запуска нескольких контейнеров. Чаще всего используется для запуска приложений, состоящих из нескольких сервисов: API + база данных, API + Redis, несколько микросервисов, набор инструментов разработки.

Compose позволяет автоматизировать развёртывание и управление всей инфраструктурой проекта с помощью одного файла `docker-compose.yml`.

1.7.1. Что такое Docker Compose и для чего оно применяется

Docker Compose позволяет:

- описывать инфраструктуру проекта в одном YAML-файле,
- запускать сразу несколько контейнеров одной командой,
- автоматически создавать сети и точки связи между контейнерами,
- конфигурировать переменные среды, тома и порты,
- обеспечивать воспроизводимость среды разработки.

Compose незаменим при разработке и сопровождении систем, где есть:

- API + PostgreSQL,

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

- API + брокер сообщений (MessageBroker),
- набор микросервисов,
- системы мониторинга (Grafana, Prometheus),
- окружения для разработки.

Одной командой можно развернуть всё приложение и его инфраструктуру.

1.7.2. Структура файла docker-compose.yml

Файл `docker-compose.yml` описывает, какие сервисы входят в систему, как они связаны и какие настройки используются.

Ключевые разделы

1. services — перечень контейнеров

В блоке `services` описываются все контейнеры, которые нужно запустить.

```
services:
  app:
    image: myapi
  db:
    image: postgres
```

2. image — образ для контейнера

Указывает, какой Docker-образ использовать:

```
image: postgres:15
image: myapi:latest
```

Вместо `image` можно использовать `build:` для сборки образа из Dockerfile:

```
services:
  app:
    build: .
```

3. volumes — тома для хранения данных

Используются для:

- хранения данных БД,

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

- сохранения логов,
- передачи конфигураций.

```
volumes:
  pgdata:
```

Привязка тома в сервисе:

```
services:
  db:
    image: postgres:15
    volumes:
      - pgdata:/var/lib/postgresql/data
```

Том сохраняет данные, даже если контейнер удалён, и обеспечивает долговременное хранение информации.

4. ports — проброс портов

Позволяет открыть порт контейнера наружу (на хостовую машину):

```
ports:
  - "8080:8080"
```

Формат: `внешний:внутренний` порт.

5. environment — переменные среды

Используются для передачи конфигураций в контейнер:

```
environment:
  POSTGRES_USER: admin
  POSTGRES_PASSWORD: 1234
  POSTGRES_DB: appdb

  ASPNETCORE_ENVIRONMENT: Production
  ConnectionStrings__Default: "Host=db;Port=5432;Database=appdb;Username=admin;Password=1234"
```

В docker-compose сервисы доступны по имени как по хосту, например: `Host=db`.

1.7.3. Связь нескольких контейнеров

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Docker Compose автоматически создаёт виртуальную сеть (bridge), в которой каждый контейнер доступен по имени сервиса.

Например, API-сервис `app` подключается к БД `db`, используя:

- `Host=db`
- `Port=5432`

Образец конфигурации

```
services:
  app:
    build: .
    depends_on:
      - db
    environment:
      ConnectionStrings__Default: "Host=db;Port=5432;Database=appdb;Username=admin;Password=1234"

  db:
    image: postgres:15
    environment:
      POSTGRES_USER: admin
      POSTGRES_PASSWORD: 1234
      POSTGRES_DB: appdb
```

Директива `depends_on` говорит Compose, что сначала надо запустить `db`, а затем `app`.

> **Замечание**
>
> `depends_on` отвечает только за порядок запуска контейнеров, но не гарантирует готовность сервиса (например, базы данных). Для продакшена дополнительно используют health-check и ожидание готовности.

1.7.4. Пример полноценного docker-compose.yml

```
version: "3.9"

services:
  app:
    build: .
    container_name: myapi
    ports:
      - "8080:8080"
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

```
    depends_on:
      - db
    environment:
      ASPNETCORE_ENVIRONMENT: Development
      ConnectionStrings__Default: "Host=db;Port=5432;Database=appdb;Username=admin;Password=1234"

  db:
    image: postgres:15
    container_name: postgres_db
    environment:
      POSTGRES_USER: admin
      POSTGRES_PASSWORD: 1234
      POSTGRES_DB: appdb
    volumes:
      - pgdata:/var/lib/postgresql/data

volumes:
  pgdata:
```

Этот compose-файл создаёт API-сервис и PostgreSQL с автоматическим подключением и сохранением данных БД в томе `pgdata`.

1.7.5. Основные команды Docker Compose

1. Запуск всех сервисов

```
SCRIPT

docker compose up -d
```

- `up` — поднять контейнеры,
- `-d` — фоновый режим.

После запуска все сервисы работают одновременно.

2. Остановка и удаление контейнеров

```
SCRIPT

docker compose down
```

С удалением томов:

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

SCRIPT

```
docker compose down -v
```

3. Просмотр логов всех сервисов

SCRIPT

```
docker compose logs
```

Просмотр логов конкретного сервиса:

SCRIPT

```
docker compose logs app
```

Потоковое наблюдение:

SCRIPT

```
docker compose logs -f
```

Практический совет

Для учебных проектов достаточно хранить файл `docker-compose.yml` в корне репозитория и запускать всю инфраструктуру одной командой `docker compose up -d`.

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

1.8. Мониторинг, логи и health-check

ОГЛАВЛЕНИЕ РАЗДЕЛА 1.8

- 1.8. Мониторинг, логи и health-check
- 1.8.1. Просмотр логов контейнера
- 1.8.2. Health-check
- 1.8.3. Диагностика ошибок API

1.8. Мониторинг, логи и health-check

Мониторинг, анализ логов и проверка состояния приложения — ключевые элементы сопровождения информационных систем. Они позволяют обнаруживать ошибки, отслеживать состояние API и предотвращать падения контейнеров.

1.8.1. Просмотр логов контейнера

Логи отображают:

- ошибки запуска приложения,
- вывод консоли,
- сообщения API,
- исключения .NET,
- системные уведомления контейнера.

Команда для просмотра логов

```script
docker logs <container_name>
```

Пример:

```script
docker logs myapi
```


Помогу с учебным заданием без предоплаты. Оплата после выполнения

Нажмите на мессенджер, чтобы написать

Telegram ↗ WhatsApp ↗ MAX ↗

+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

Получение логов в реальном времени

```
SCRIPT

docker logs -f myapi
```

`-f` означает «follow» — потоковый вывод.

Зачем нужны логи контейнера

- быстрое обнаружение ошибок во время запуска,
- диагностика падений или зависаний API,
- просмотр ошибок базы данных,
- анализ внутренних исключений C#/.NET,
- определение, что контейнер вообще запустился.

> **Практический факт**
>
> Логи — первое место, куда смотрит администратор при инциденте.

1.8.2. Health-check: что это и зачем нужно

Health-check — это механизм автоматической проверки состояния приложения. Он показывает не только факт запуска контейнера, но и способность API корректно отвечать на запросы.

Проверяется:

- доступность API,
- корректность ответа,
- время отклика.

1.8.2.1. Endpoint `/health`

Стандартный endpoint:

```
GET /health
```

Пример ответа:

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

```
{
  "status": "Healthy"
}
```

В ASP.NET Core можно подключить встроенный middleware:

```
app.MapHealthChecks("/health");
```
SCRIPT

1.8.2.2. Health-check в Docker

Docker может автоматически проверять состояние контейнера с помощью инструкции HEALTHCHECK.

```
HEALTHCHECK --interval=30s --timeout=3s \
  CMD curl -f http://localhost:8080/health || exit 1
```

Статусы:

- **healthy** — контейнер работает корректно;
- **unhealthy** — приложение не отвечает, контейнер повреждён.

Посмотреть состояние:

```
docker ps
```
SCRIPT

1.8.2.3. Для чего нужен health-check

- автоматически обнаруживать зависшие приложения,
- перезапускать контейнеры оркестраторами (Docker Swarm, Kubernetes),
- уведомлять администраторов о сбоях,
- не допускать трафик в сломанный контейнер.

1.8.3. Диагностика ошибок API

При работе с Web API важно уметь распознавать типичные ошибки, которые возвращаются сервисом.

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

1.8.3.1. Ошибка 500 — Internal Server Error

Причины:

- необработанное исключение в .NET;
- ошибка в бизнес-логике;
- ошибка подключения к базе данных;
- неправильная сериализация JSON;
- неверная конфигурация среды.

Диагностика:

1. Посмотреть логи контейнера:

```
docker logs myapi
```
SCRIPT

2. Проверить стек исключений.
3. Проверить подключение к БД.
4. Проверить переменные среды.

1.8.3.2. Ошибка 400 — Bad Request

Причины:

- некорректный JSON;
- пропущенные обязательные поля;
- некорректный DTO;
- ошибка валидации атрибутов;
- ошибка параметров запроса.

Диагностика:

- проверить тело запроса;
- проверить DTO и атрибуты валидации;
- сверить ожидаемую схему данных.

1.8.3.3. Падения контейнера

Статусы:

- **Exited (1)** — ошибка выполнения;
- **Restarting** — пытается перезапуститься;
- **Dead** — контейнер полностью упал.

Причины:

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

- ошибка приложения в точке входа;
- неправильный ENTRYPOINT или CMD;
- неверный порт;
- ошибка в миграциях EF Core;
- отсутствующие зависимости.

Как анализировать

1. Проверить статус контейнера:

```
docker ps -a
```
SCRIPT

2. Посмотреть причину:

```
docker logs <container>
```
SCRIPT

3. Проверить ENTRYPOINT и EXPOSE.
4. Проверить работу базы данных и строку подключения.

> **Совет**
>
> 80% ошибок выявляется через просмотр логов и проверку конфигурации.

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

1.9. Практики сопровождения информационных систем

ОГЛАВЛЕНИЕ РАЗДЕЛА 1.9

- 1.9. Практики сопровождения информационных систем
- 1.9.1. Обновление версий приложения
- 1.9.2. Резервное копирование данных
- 1.9.3. CI/CD (базовое понимание)
- 1.9.4. Почему контейнеры упрощают сопровождение

1.9. Практики сопровождения информационных систем

Сопровождение — это комплекс мероприятий, направленных на обеспечение стабильной, безопасной и актуальной работы информационной системы.

Студент должен понимать, как выполняются обновления системы, как защищаются данные и как современные инструменты (CI/CD, Docker) упрощают эксплуатацию.

1.9.1. Обновление версий приложения

Обновление версии — это процесс развертывания новой сборки приложения в рабочую среду. Ниже приведена типовая схема обновления контейнеризованного приложения.

1. Новый код

Разработчик вносит изменения в проект:

- новое API,
- исправление ошибок,
- обновлённые зависимости.

Код сохраняется в Git-репозитории.

2. Сборка нового образа

Помогу с учебным заданием без предоплаты. Оплата после выполнения

Нажмите на мессенджер, чтобы написать

Telegram ↗ WhatsApp ↗ MAX ↗

+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

На основе Dockerfile создаётся новый образ приложения:

```
SCRIPT

docker build -t myapp:v2 .
```

Версионирование (например, `v1`, `v2`, `v3`) позволяет при необходимости откатываться на предыдущие версии.

3. Публикация образа (опционально)

Отправка образа в Docker Hub или GitHub Container Registry:

```
SCRIPT

docker push username/myapp:v2
```

4. Перезапуск контейнера

Сначала останавливается и удаляется старый контейнер:

```
SCRIPT

docker stop app
docker rm app
```

Затем запускается новый контейнер на основе обновлённого образа:

```
SCRIPT

docker run -d -p 8080:8080 --name app myapp:v2
```

Если используется Docker Compose:

```
SCRIPT

docker compose down
docker compose up -d --build
```

5. Проверка работоспособности

- проверка endpoint `/health`,
- просмотр логов:

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

```
docker logs app
```
SCRIPT

- проверка ответов API и статусов HTTP.

1.9.2. Резервное копирование данных

Резервное копирование (backup) — обязательный элемент сопровождения. Оно позволяет восстановить систему при:

- сбоях оборудования,
- ошибках обновления,
- повреждении данных,
- ошибках пользователя.

1. SQL dump (экспорт базы данных)

Для PostgreSQL:

```
pg_dump -U postgres appdb > backup.sql
```
SCRIPT

Восстановление:

```
psql -U postgres appdb < backup.sql
```
SCRIPT

SQL dump позволяет хранить копию данных вне контейнера и вне томов Docker.

2. Резервное копирование Docker volume

Если база данных работает в контейнере, данные обычно хранятся в томе (volume). Для резервного копирования тома:

```
docker run --rm \
  -v pgdata:/data \
  -v $(pwd):/backup \
  ubuntu tar cvf /backup/pgdata.tar /data
```
SCRIPT

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Восстановление volume:

```
SCRIPT

docker run --rm \
  -v pgdata:/data \
  -v $(pwd):/backup \
  ubuntu tar xvf /backup/pgdata.tar -C /
```

Том содержит файлы базы данных (каталоги `base/`, журналы `*.wal` и другие данные PostgreSQL).

3. Рекомендации по резервному копированию

- хранить резервные копии на другом диске или сервере,
- делать ежедневные инкрементальные и еженедельные полные копии,
- регулярно тестировать процесс восстановления,
- хранить резервные копии не менее 30 дней.

1.9.3. CI/CD (базовое понимание)

CI/CD — это подход к автоматизации разработки и сопровождения приложений.

CI — Continuous Integration (постоянная интеграция)

Включает:

- автоматическую сборку кода при коммитах,
- запуск модульных и интеграционных тестов,
- анализ ошибок и отчёты о качестве сборки.

CD — Continuous Delivery / Deployment

Обычно включает:

- автоматическое создание Docker-образа,
- развёртывание на тестовой или продуктивной среде,
- раскатку обновлений по серверам.

Зачем нужно CI/CD

- уменьшает количество ошибок при запуске новых версий,
- ускоряет выпуск обновлений,

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

- обеспечивает повторяемость процессов,
- уменьшает объём ручной работы и человеческий фактор,
- повышает качество сопровождения и стабильность системы.

1.9.4. Почему контейнеры упрощают сопровождение

Контейнеризация — один из ключевых инструментов DevOps, значительно упрощающий сопровождение приложений.

1. Единая среда

Контейнер включает в себя:

- приложение,
- зависимости и библиотеки,
- конфигурации,
- конкретные версии компонентов.

Не требуется отдельно настраивать сервер под каждое приложение.

2. Предсказуемость

Если контейнер работает на локальной машине разработчика, он будет работать и на сервере при тех же настройках. Проблема «работает только у меня» существенно снижается.

3. Быстрое развертывание

Новый контейнер запускается за секунды, что ускоряет обновление и масштабирование.

4. Лёгкий откат

Если новая версия работает некорректно, можно запустить предыдущую:

```
SCRIPT

docker run myapp:v1
```

5. Масштабирование

Легко запускать несколько экземпляров одного и того же образа и распределять нагрузку между ними.

6. Упрощённый мониторинг и логирование

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Базовые команды Docker помогают быстро диагностировать проблемы:

```
SCRIPT

docker logs
docker ps
docker inspect
```

В сочетании с внешними системами мониторинга (Prometheus, Grafana, ELK) контейнеры дают полную картину состояния приложения.

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

1.10. Документирование

ОГЛАВЛЕНИЕ РАЗДЕЛА 1.10

- 1.10. Документирование
- 1.10.1. Архитектурная схема
- 1.10.2. Описание API
- 1.10.3. Инструкция по запуску
- 1.10.4. Инструкция по обновлению
- 1.10.5. Инструкции по резервному копированию
- 1.10.6. Swagger и Postman: основы работы

1.10. Документирование

ОГЛАВЛЕНИЕ РАЗДЕЛА 1.10

- 1.10. Документирование
- 1.10.1. Архитектурная схема
- 1.10.2. Описание API
- 1.10.3. Инструкция по запуску
- 1.10.4. Инструкция по обновлению
- 1.10.5. Инструкции по резервному копированию
- 1.10.6. Swagger и Postman: основы работы

1.10. Документирование

Документирование — ключевой этап разработки и сопровождения информационных систем. Грамотная документация позволяет новым разработчикам, администраторам и пользователям быстро понять структуру проекта, особенности его запуска и эксплуатации.

Помогу с учебным заданием без предоплаты. Оплата после выполнения

Нажмите на мессенджер, чтобы написать

Telegram ↗ WhatsApp ↗ MAX ↗

+7 (924) 305-23-08 · Пишите в мессенджер

disynergy.ru

Документы должны быть структурированными, однозначными, актуальными и соответствовать реальному состоянию системы.

1.10.1. Архитектурная схема

Архитектурная схема — это графическое представление структуры информационной системы.

Обычно включает:

- компоненты приложения (API, базы данных, сервисы),
- взаимодействие модулей,
- сетевые соединения (порты, протоколы),
- внешние интеграции (сторонние API, платежные сервисы),
- контейнерную инфраструктуру (Docker),
- схему деплоя.

Архитектурная схема помогает:

- быстро понять структуру проекта,
- определить зависимости,
- выявить узкие места,
- упростить сопровождение и модернизацию системы.

Используемые инструменты:

- Draw.io
- Miro
- Figma
- PlantUML
- Mermaid

1.10.2. Описание API

Описание API — это документ, определяющий правила взаимодействия внешних клиентов с системой.

1. Эндпоинты

Каждый эндпоинт должен быть описан:

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

- адрес (например, `/api/users`),
- HTTP-метод (GET, POST, PUT, DELETE),
- назначение,
- параметры запроса,
- структура тела запроса,
- структура тела ответа,
- коды ошибок (400, 404, 500).

2. Примеры запросов

```
POST /api/users
Content-Type: application/json

{
  "email": "test@mail.com",
  "password": "123456"
}
```

3. Форматы данных

- JSON (основной формат),
- правила сериализации и десериализации.

Качественная документация API ускоряет разработку и облегчает интеграцию.

1.10.3. Инструкция по запуску

Инструкция по запуску позволяет поднять систему «с нуля».

Обязательные элементы:

1. **Системные требования**
   Версии .NET, PostgreSQL, Docker и других инструментов.
2. **Переменные среды**
   Примеры:
   `ConnectionStrings__Default`,
   `JwtKey`
3. **Команды запуска**

```
SCRIPT

docker build -t myapi .
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

```
docker run -p 8080:8080 myapi
```

4. **Порядок запуска сервисов**
   сначала база данных → затем API → затем миграции
5. **Проверка успешного запуска**
   открыть `/swagger` или `/health`

> **Важно**
>
> Инструкция должна быть протестирована и повторяемой.

1.10.4. Инструкция по обновлению

Инструкция по обновлению используется при выпуске новой версии приложения.

1. Описание того, что обновляется (код, конфигурации, миграции).
2. Подготовка резервной копии.
3. Остановка старой версии.
4. Загрузка образа или сборка проекта.
5. Выполнение миграций:

```
SCRIPT
dotnet ef database update
```

6. Запуск новой версии.
7. Проверка работоспособности через `/health` или мониторинг.
8. Откат (rollback) на случай ошибки.

> **Комментарий**
>
> Документирование процесса обновления снижает риск ошибок и ускоряет сопровождение.

1.10.5. Инструкции по резервному копированию

Резервное копирование — обязательный процесс для защиты данных.

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

1. Что нужно копировать

- базы данных PostgreSQL (dump),
- Docker volumes,
- конфигурационные файлы,
- логи,
- файлы окружения и ключи.

2. Инструменты

- `pg_dump`, `pg_restore`
- резервирование Docker volume
- отдельные backup-скрипты

3. Частота и политика хранения

- ежедневные инкрементальные копии,
- еженедельные полные,
- хранение не менее 30 дней.

4. Примеры команд

Резервная копия PostgreSQL:

```
SCRIPT
pg_dump -U postgres mydb > backup.sql
```

Резервная копия Docker volume:

```
SCRIPT
docker run --rm -v pgdata:/data -v $(pwd):/backup \
  ubuntu tar cvf /backup/pgdata.tar /data
```

5. Порядок восстановления

- развернуть БД,
- импортировать dump или volume,
- проверить целостность данных,
- запустить приложение.

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Рекомендация

Все инструкции backup должны храниться вместе с проектом в отдельной папке docs/.

1.10.6. Swagger и Postman: основы работы

Swagger

Swagger — встроенный инструмент ASP.NET Core для автоматической генерации документации API.

Позволяет:

- просматривать список всех эндпоинтов,
- изучать структуру запросов и ответов,
- отправлять тестовые запросы,
- использовать JSON/OpenAPI-спецификацию.

Обычно доступен по адресу `/swagger`.

Подключение:

```
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();
app.UseSwagger();
app.UseSwaggerUI();
```

Postman

Postman используется для ручного тестирования API.

- создание коллекций запросов,
- проверка ошибок и кодов ответа,
- работа с параметрами и JSON,
- отправка токенов (Bearer JWT),
- экспорт коллекций.

Postman помогает проверить API до написания автотестов или интеграционных сценариев.

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Методические рекомендации. 2. Выполнение задания

ОГЛАВЛЕНИЕ РАЗДЕЛА 2

- 2. Выполнение задания
- 2.1. Описание задания
- 2.1.1. Описание задачи
- 2.1.2. Тестовая предметная область

2. Выполнение задания

В данном разделе описывается состав задания для производствееной практики, ожидаемый результат и базовая тестовая предметная область, на основе которой студенты реализуют информационную систему.

2.1. Описание задания

2.1.1. Описание задачи

Студенту предлагается выполнить следующие шаги:

- Разработать информационную систему на языке C# с использованием ASP.NET Core Web API.
- Реализовать базовый CRUD-функционал для выбранной предметной области (например, задачи, клиенты, товары, заявки).
- Контейнеризировать приложение с помощью Docker.
- Создать **Dockerfile** с использованием многоэтапной (multi-stage) сборки.
- Собрать собственный Docker-образ и развернуть приложение в контейнере.
- При необходимости подключить базу данных через `docker-compose`.
- Выполнить обновление версии приложения в процессе сопровождения (например, выпустить версию v2, внести изменения и повторно развернуть).
- Продемонстрировать работу с логами контейнеров (`docker logs`).
- Реализовать и проверить работоспособность системы через **health-check** (endpoint `/health`, HEALTHCHECK в Dockerfile).

Помогу с учебным заданием без предоплаты. Оплата после выполнения

Нажмите на мессенджер, чтобы написать

Telegram ↗ WhatsApp ↗ MAX ↗

+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

- Выполнить базовые процедуры резервного копирования и восстановления данных (если используется база данных).
- Подготовить краткую документацию:
  - описание системы и предметной области,
  - инструкцию запуска в Docker и через docker-compose,
  - порядок обновления (release v1 → v2),
  - примеры обращений к API (эндпоинты, запросы, ответы).

Результат

Итогом выполнения задания должен быть рабочий Web API-сервис, развёрнутый в контейнере, с реализованными CRUD-операциями, базовыми процедурами сопровождения и минимальным комплектом документации.

2.1.2. Тестовая предметная область

«Система управления заявками на ремонт и обслуживание техники в колледже»

Информационная система предназначена для учёта заявок на ремонт и обслуживание компьютерной и офисной техники в учебном заведении. Система должна позволять регистрировать, отслеживать и закрывать заявки, а также хранить информацию об оборудовании и исполнителях.

Основные сущности (под CRUD)

1. Equipment (Оборудование)

Хранит информацию о единице техники.

- `id` — идентификатор;
- `inventoryNumber` — инвентарный номер;
- `type` — тип (ПК, принтер, проектор и др.);
- `room` — кабинет/аудитория;
- `state` — состояние (исправно, требует ремонта, списано и т.п.).

2. Request (Заявка)

Отражает процесс обращения по поводу поломки или обслуживания.

- `id` — идентификатор заявки;
- `requester` — заявитель (ФИО или группа);

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

- `equipmentId` — ссылка на оборудование (внешний ключ к Equipment);
- `description` — описание проблемы;
- `priority` — приоритет (низкий, средний, высокий);
- `status` — статус (новая / в работе / выполнена / отменена);
- `assigneeId` — исполнитель (ссылка на User);
- `createdAt` — дата и время создания заявки;
- `closedAt` — дата и время закрытия заявки (опционально).

3. User (Пользователь / исполнитель)

Представляет участников процесса обработки заявок.

- `id` — идентификатор пользователя;
- `fullName` — ФИО;
- `role` — роль (оператор, техник, администратор);
- `contacts` — контактные данные (телефон, e-mail).

4. Comment (Комментарий к заявке, опционально)

Позволяет фиксировать ход выполнения работы по заявке.

- `id` — идентификатор комментария;
- `requestId` — ссылка на заявку;
- `authorId` — автор комментария (ссылка на User);
- `text` — текст комментария;
- `createdAt` — время добавления комментария.

Рекомендуемый минимальный набор операций

Сущность	Операции
Equipment	Создание, просмотр списка, просмотр одного, обновление, удаление
Request	Создание заявки, изменение статуса, назначение исполнителя, просмотр истории заявок
User	Создание пользователей, назначение ролей, просмотр списка исполнителей
Comment (опционально)	Добавление комментария к заявке, просмотр списка комментариев

Замечание

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

При желании студент может расширить выбранную из списка предметную область: добавить историю изменений статуса, файлы-вложения, категории оборудования или дополнительные роли пользователей. Важно документировать все расширения в описании системы и API.

Помогу с учебным заданием без предоплаты. Оплата после выполнения

Нажмите на мессенджер, чтобы написать

Telegram ↗ WhatsApp ↗ MAX ↗

+7 (924) 305-23-08 · Пишите в мессенджер

disynergy.ru

2.2. Разработать информационную систему на языке C# с использованием ASP.NET Core Web API

ОГЛАВЛЕНИЕ РАЗДЕЛА 2.2

- 2.2.0. Цель и результат блока
- 2.2.1. Подготовка окружения
- 2.2.2. Создание проекта Web API
- 2.2.3. Установка Swagger-пакета
- 2.2.4. Очистка и восстановление пакетов
- 2.2.5. Приведение Program.cs в порядок
- 2.2.6. Проверка запуска
- 2.2.7. Создание моделей предметной области
- 2.2.8. Настройка БД (InMemory)
- 2.2.9. Реализация CRUD-контроллера
- 2.2.10. Установка пакетов EF Core и проверка .csproj
- 2.2.11. Тестирование CRUD через Swagger
- Пример тестового проекта

2.2.0. Цель и результат блока

Цель:

К концу блока у студента должен быть локально работающий Web API-проект на ASP.NET Core, реализующий CRUD для выбранной предметной области (например: «Заявки на ремонт техники в колледже»).

Результат:

- решение .NET с проектом `ServiceDesk.Api` (название примерное);
- описанные C#-модели домена (например, `Equipment`, `ServiceRequest`, `User`);
- настроенный `DbContext` (на данном этапе можно использовать InMemory-БД);

Помогу с учебным заданием без предоплаты. Оплата после выполнения

Нажмите на мессенджер, чтобы написать

Telegram ↗ WhatsApp ↗ MAX ↗

+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

- контроллеры с CRUD-методами и работающий Swagger (UI для тестирования API).

> **Внимание**
>
> Вам требуется выполнять действия под свою предметную область.

2.2.1. Подготовка окружения

Задача: убедиться, что всё нужное установлено.

1. Установить:
   - .NET SDK (например, .NET 8);
   - любую IDE: Visual Studio / Rider / VS Code.
2. Проверить версию .NET в терминале:

```
SCRIPT

dotnet --version
```



Рисунок 1 – Пример проверки в терминале

Ожидаемый результат: команда отрабатывает без ошибок и выводит номер версии.

> **Внимание**
>
> Чтобы выйти из режима запуска сервисов в консоли, используйте сочетание клавиш `Ctrl + C`.

2.2.2. Создание проекта Web API

Задача: создать скелет проекта ASP.NET Core Web API.

2.2.2.1. Создание папки проекта

1. Создать папку под проект (например, `ppSIS` или любую другую).

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

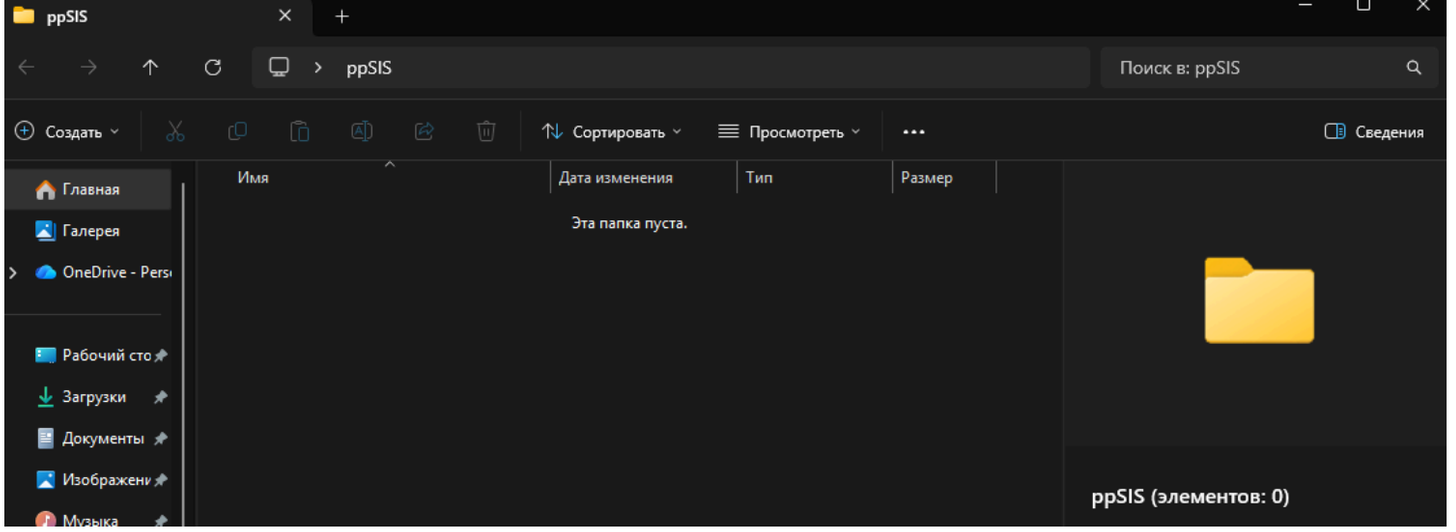

Рисунок 2 – Создание папки

2.2.2.2. Открытие папки в терминале

1. Открыть созданную папку в терминале (PowerShell, CMD, встроенный терминал IDE).

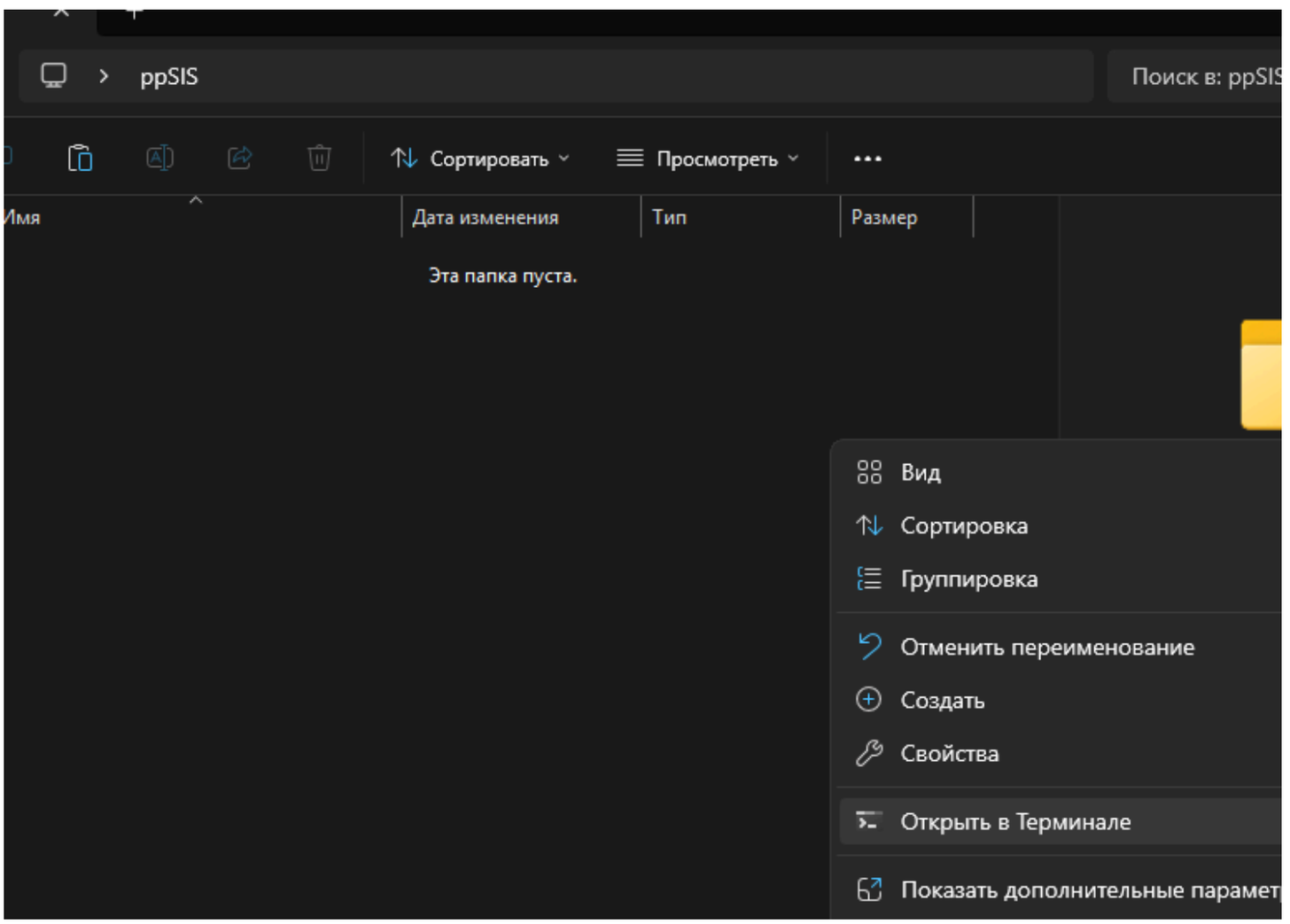

Рисунок 3 – Открытие терминала в папке

2.2.2.3. Создание проекта Web API

1. В папке для проектов выполнить команды:

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

```
dotnet new webapi -n ServiceDesk.Api
cd ServiceDesk.Api
```

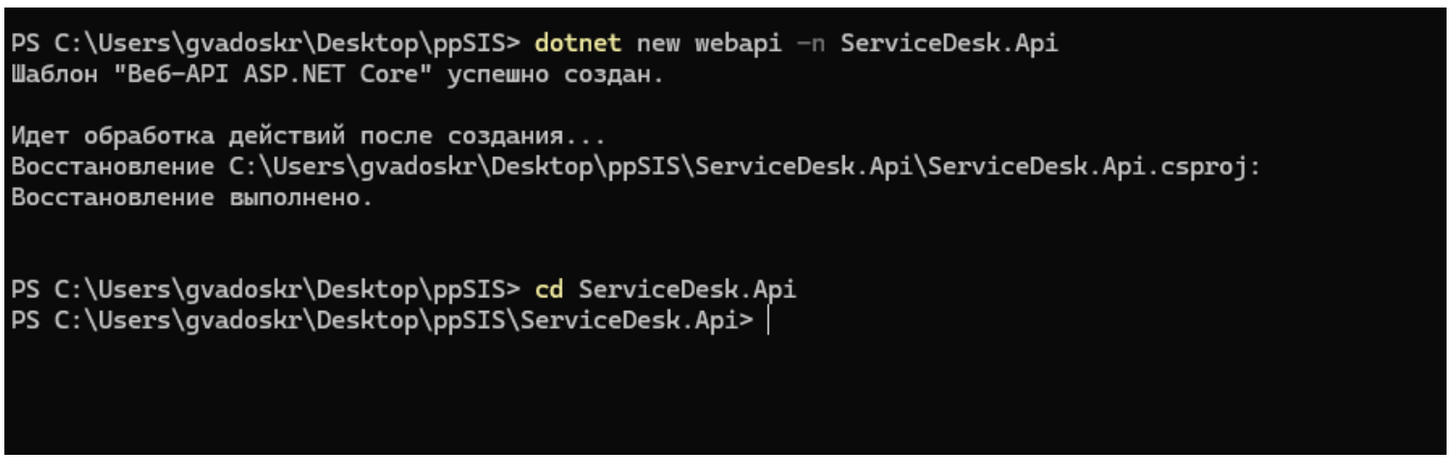

Рисунок 4 – Запуск команд в терминале

2.2.2.4. Открытие проекта в IDE

1. Открыть проект `ServiceDesk.Api` в IDE (Visual Studio, Rider или VS Code).

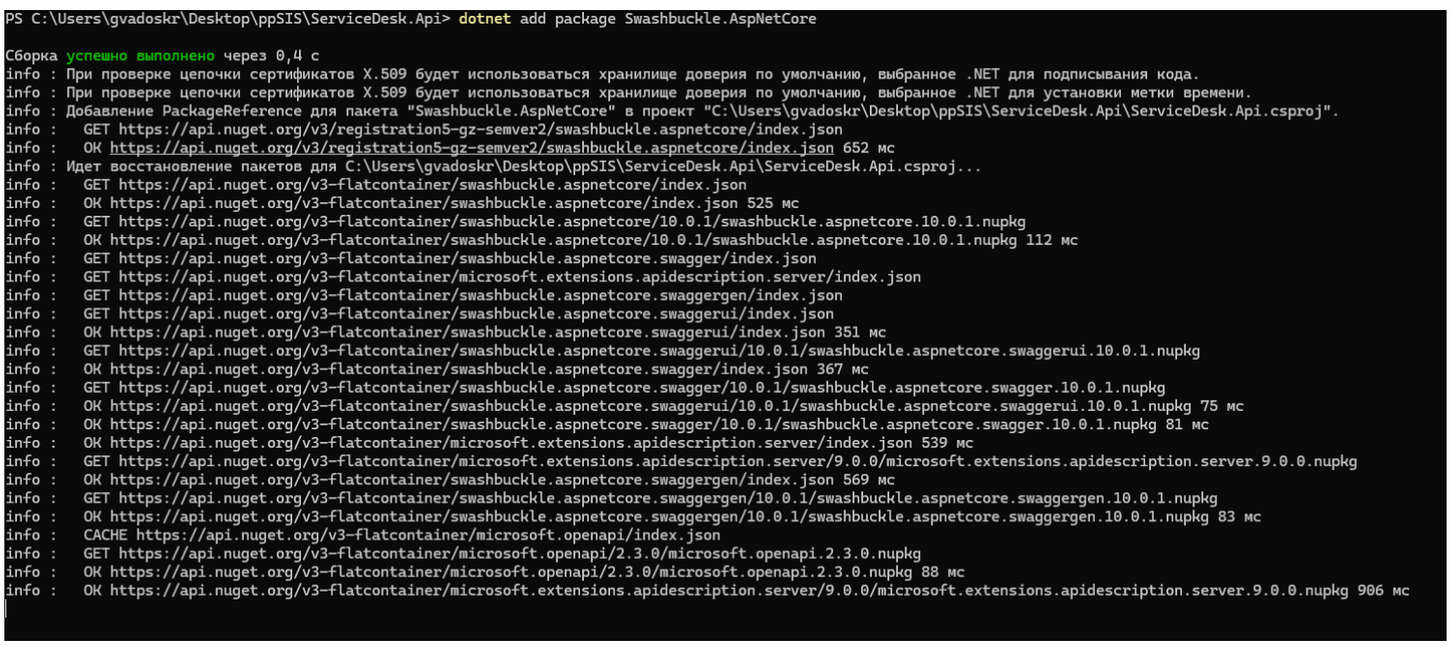

Рисунок 5 – Возможность открытия через нужную IDE

2.2.3. Установка правильного Swagger-пакета

1. Выполнить команду для добавления библиотеки Swagger:

```
dotnet add package Swashbuckle.AspNetCore
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

2. Открыть файл `ServiceDesk.Api.csproj` и найти блок:

```
<ItemGroup>
  <PackageReference Include="Microsoft.AspNetCore.OpenApi" Version="..." />
  <PackageReference Include="Microsoft.OpenApi" Version="..." />
  <PackageReference Include="Swashbuckle.AspNetCore" Version="..." />
  ...
</ItemGroup>
```

3. Оставить только строку с `Swashbuckle.AspNetCore`, чтобы получилось:

```
<ItemGroup>
  <PackageReference Include="Swashbuckle.AspNetCore" Version="6.5.0" />
</ItemGroup>
```

Если версия другая – не страшно, главное, чтобы **не было** строк с `Microsoft.AspNetCore.OpenApi` и `Microsoft.OpenApi`.

2.2.4. Почистить bin/obj и восстановить пакеты

В терминале из папки проекта `ServiceDesk.Api` выполнить:

```
SCRIPT

dotnet clean
dotnet restore
```

2.2.5. Приводим Program.cs в порядок

Файл `Program.cs` должен быть точно таким (замените содержимое полностью):

```
using Microsoft.AspNetCore.Builder;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.EntityFrameworkCore;
using ServiceDesk.Api.Data;

var builder = WebApplication.CreateBuilder(args);

// Регистрируем контроллеры
builder.Services.AddControllers();
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

```
// Swagger (Swashbuckle)
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

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

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();
```

2.2.6. Проверка запуска

В терминале из папки проекта выполнить:

```
SCRIPT

dotnet run
```

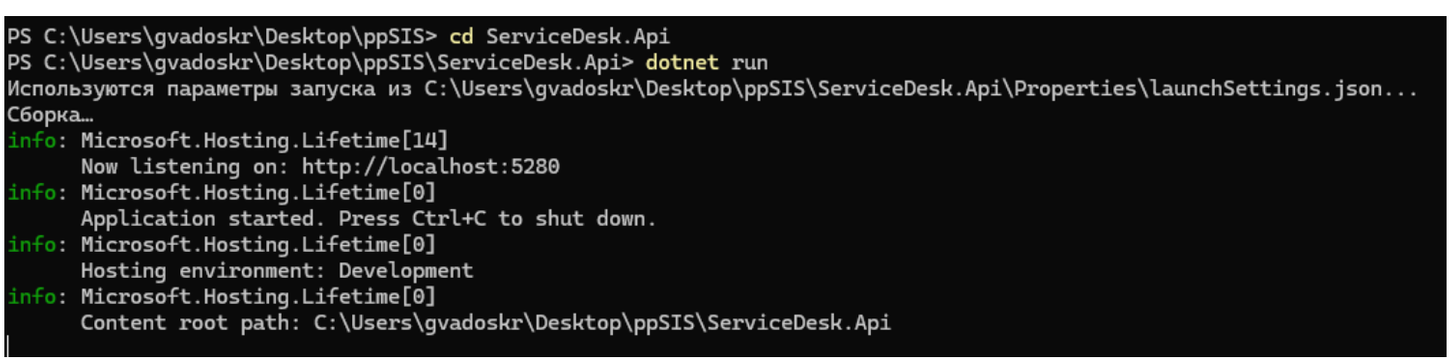

Рисунок 6 – Запуск приложения

В консоли появится сообщение вида:

```
SCRIPT

Now listening on: http://localhost:5280
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

```
Используются параметры запуска из C:\Users\gvadoskr\Desk
Сборка…
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5280
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
```

Рисунок 7 – Появление ссылки в консоли

1. Открыть в браузере адрес из консоли.
2. Перейти на `/swagger` (Swagger UI).

Пример:

```
SCRIPT

http://localhost:5280/swagger
```

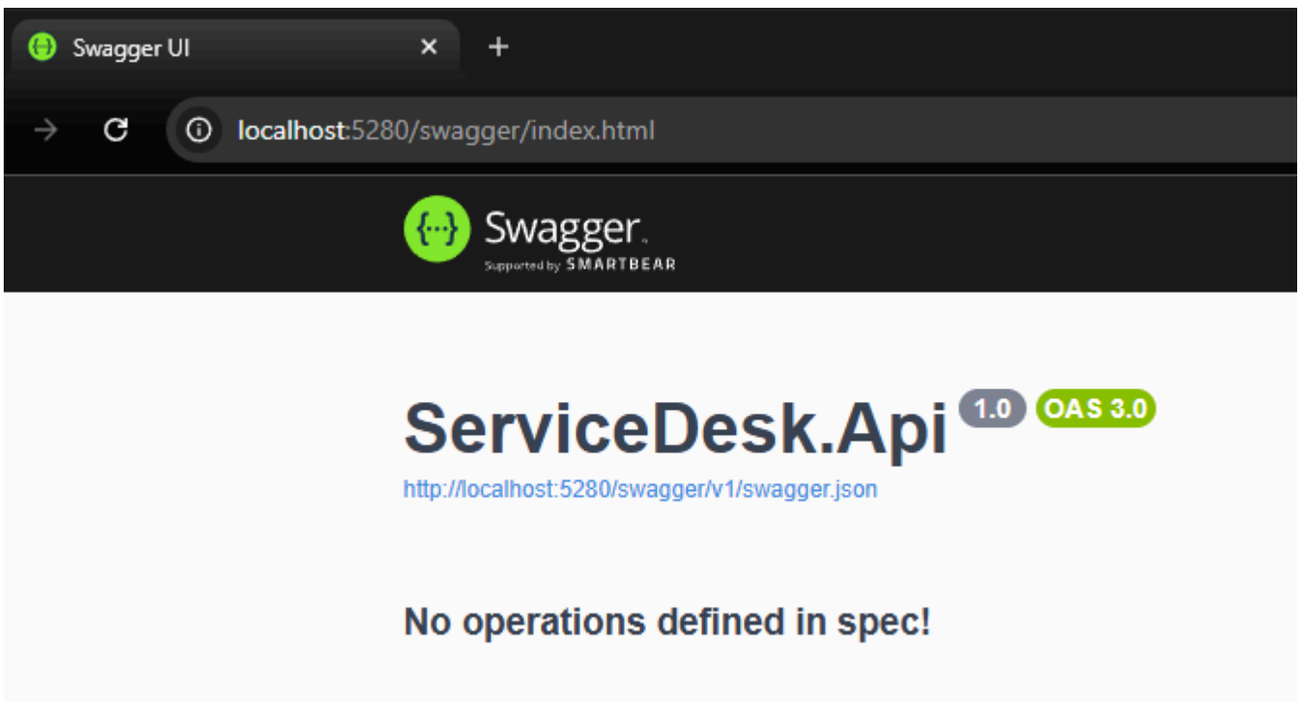

Рисунок 8 – Запуск Swagger

Если открылся Swagger UI — блок успешно выполнен.

2.2.7. Создание моделей предметной области

Создать папку `Models` и два файла:

Models/Equipment.cs

```
namespace ServiceDesk.Api.Models
{
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

```
    public class Equipment
    {
        public int Id { get; set; }
        public string InventoryNumber { get; set; } = "";
        public string Type { get; set; } = "";
        public string Room { get; set; } = "";
        public string Status { get; set; } = "Работает";
    }
}
```

Models/ServiceRequest.cs

```
namespace ServiceDesk.Api.Models
{
    public class ServiceRequest
    {
        public int Id { get; set; }
        public string Title { get; set; } = "";
        public string Description { get; set; } = "";
        public string Priority { get; set; } = "Средний";
        public string Status { get; set; } = "Новая";
        public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
        public DateTime? ClosedAt { get; set; }

        public int? EquipmentId { get; set; }
        public Equipment? Equipment { get; set; }
    }
}
```

2.2.8. Настройка БД (InMemory)

1. Добавить пакет:

```
SCRIPT

dotnet add package Microsoft.EntityFrameworkCore.InMemory
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

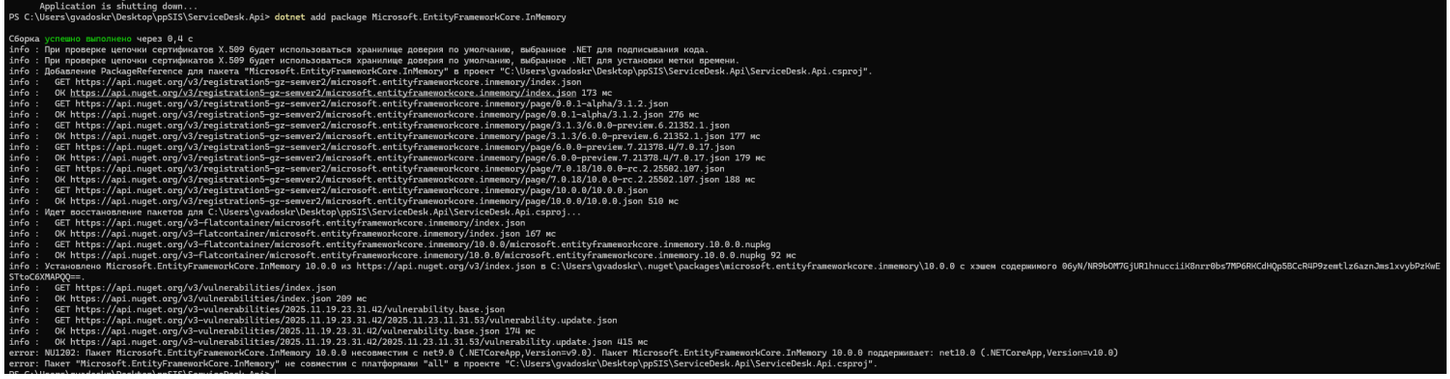

Рисунок 9 – Добавление пакета

Создать папку `Data` и файл `Data/AppDbContext.cs`:

```
using Microsoft.EntityFrameworkCore;     // ← ОБЯЗАТЕЛЬНО
using ServiceDesk.Api.Models;

namespace ServiceDesk.Api.Data
{
    public class AppDbContext : DbContext
    {
        public AppDbContext(DbContextOptions<AppDbContext> options)
            : base(options)
        {
        }

        public DbSet<Equipment> Equipment { get; set; }
        public DbSet<ServiceRequest> ServiceRequests { get; set; }
    }
}
```

Добавить контекст в `Program.cs`:

```
builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseInMemoryDatabase("ServiceDeskDb"));
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

```
using Microsoft.AspNetCore.Builder;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.EntityFrameworkCore;      // ← ОБЯЗАТЕЛЬНО
using ServiceDesk.Api.Data;               // ← чтобы видеть AppDbContext

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

// регистрируем DbContext с InMemory
builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseInMemoryDatabase("ServiceDeskDb"));

builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

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

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();
```

Рисунок 10 – Пример файла

2.2.9. Реализация CRUD-контроллера

Создать папку `Controllers` и файл `Controllers/EquipmentController.cs` (полный CRUD):

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;     // ← ОБЯЗАТЕЛЬНО
using ServiceDesk.Api.Data;
using ServiceDesk.Api.Models;
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

```
namespace ServiceDesk.Api.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class EquipmentController : ControllerBase
    {
        private readonly AppDbContext _context;

        public EquipmentController(AppDbContext context)
        {
            _context = context;
        }

        [HttpGet]
        public async Task<IEnumerable<Equipment>> GetAll()
            => await _context.Equipment.ToListAsync();


        [HttpGet("{id:int}")]
        public async Task<ActionResult<Equipment>> Get(int id)
        {
            var item = await _context.Equipment.FindAsync(id);
            return item == null ? NotFound() : item;
        }


        [HttpPost]
        public async Task<ActionResult<Equipment>> Create(Equipment model)
        {
            _context.Equipment.Add(model);
            await _context.SaveChangesAsync();

            return CreatedAtAction(nameof(Get), new { id = model.Id }, model);
        }


        [HttpPut("{id:int}")]
        public async Task<IActionResult> Update(int id, Equipment model)
        {
            if (id != model.Id)
                return BadRequest();

            _context.Entry(model).State = EntityState.Modified;
            await _context.SaveChangesAsync();
            return NoContent();
        }
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения

Нажмите на мессенджер, чтобы написать

Telegram ↗ WhatsApp ↗ MAX ↗

+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

```
        [HttpDelete("{id:int}")]
        public async Task<IActionResult> Delete(int id)
        {
            var item = await _context.Equipment.FindAsync(id);
            if (item == null) return NotFound();

            _context.Equipment.Remove(item);
            await _context.SaveChangesAsync();
            return NoContent();
        }
    }
}
```

2.2.10. Установи пакеты EF Core

Открыть терминал в папке проекта `ServiceDesk.Api` (там, где файл `.csproj`) и выполнить:

```
SCRIPT

dotnet add package Microsoft.EntityFrameworkCore
dotnet add package Microsoft.EntityFrameworkCore.InMemory
```

После этого (не обязательно, но полезно):

```
SCRIPT

dotnet restore
```

ServiceDesk.Api.csproj к правильному виду

Открыть файл, например:

```
SCRIPT

C:\Users\gvadoskr\Desktop\ppSIS\ServiceDesk.Api\ServiceDesk.Api.csproj
```

И заменить содержимое целиком на:

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

```
    <TargetFramework>net8.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
  </PropertyGroup>

  <ItemGroup>
    <!-- Swagger -->
    <PackageReference Include="Swashbuckle.AspNetCore" Version="6.5.0" />

    <!-- EF Core -->
    <PackageReference Include="Microsoft.EntityFrameworkCore" Version="8.0.0" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.InMemory" Version="8.0.0" />
  </ItemGroup>

</Project>
```

Очистка и восстановление зависимостей

Открыть терминал в папке проекта:

SCRIPT

```
cd C:\Users\gvadoskr\Desktop\ppSIS\ServiceDesk.Api
dotnet clean
dotnet restore
dotnet build
```

Если `dotnet build` прошёл без ошибок про EntityFrameworkCore — зависимость подтянулась.

Для контроля можно посмотреть список пакетов:

SCRIPT

```
dotnet list package
```

В разделе **Top-level Package** должны быть, например:

- Microsoft.EntityFrameworkCore 0.0.0
- Microsoft.EntityFrameworkCore.InMemory 0.0.0
- Swashbuckle.AspNetCore 0.0.0

Внимание

У вас будут свои варианты версий, которые нужно соотнести правильно с кодом.

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Проверь using в трёх файлах

Program.cs

```
using Microsoft.AspNetCore.Builder;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.EntityFrameworkCore;      // ← ОБЯЗАТЕЛЬНО
using ServiceDesk.Api.Data;              // ← чтобы видеть AppDbContext

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

// регистрируем DbContext с InMemory
builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseInMemoryDatabase("ServiceDeskDb"));

builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

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

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();
```

Data/AppDbContext.cs

```
using Microsoft.EntityFrameworkCore;     // ← ОБЯЗАТЕЛЬНО
using ServiceDesk.Api.Models;

namespace ServiceDesk.Api.Data
{
    public class AppDbContext : DbContext
    {
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

```
        public AppDbContext(DbContextOptions<AppDbContext> options)
            : base(options)
        {
        }

        public DbSet<Equipment> Equipment { get; set; }
        public DbSet<ServiceRequest> ServiceRequests { get; set; }
    }
}
```

Controllers/EquipmentController.cs

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;     // ← ОБЯЗАТЕЛЬНО
using ServiceDesk.Api.Data;
using ServiceDesk.Api.Models;

namespace ServiceDesk.Api.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class EquipmentController : ControllerBase
    {
        private readonly AppDbContext _context;

        public EquipmentController(AppDbContext context)
        {
            _context = context;
        }

        // CRUD-методы...
    }
}
```

Ещё раз сборка и запуск

Из папки проекта:

```
SCRIPT

dotnet build
dotnet run
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

2.2.11. Тестирование CRUD через Swagger

В Swagger выполнить последовательно операции с ресурсом `/api/equipment`:

1. GET /api/equipment

Получение списка оборудования.

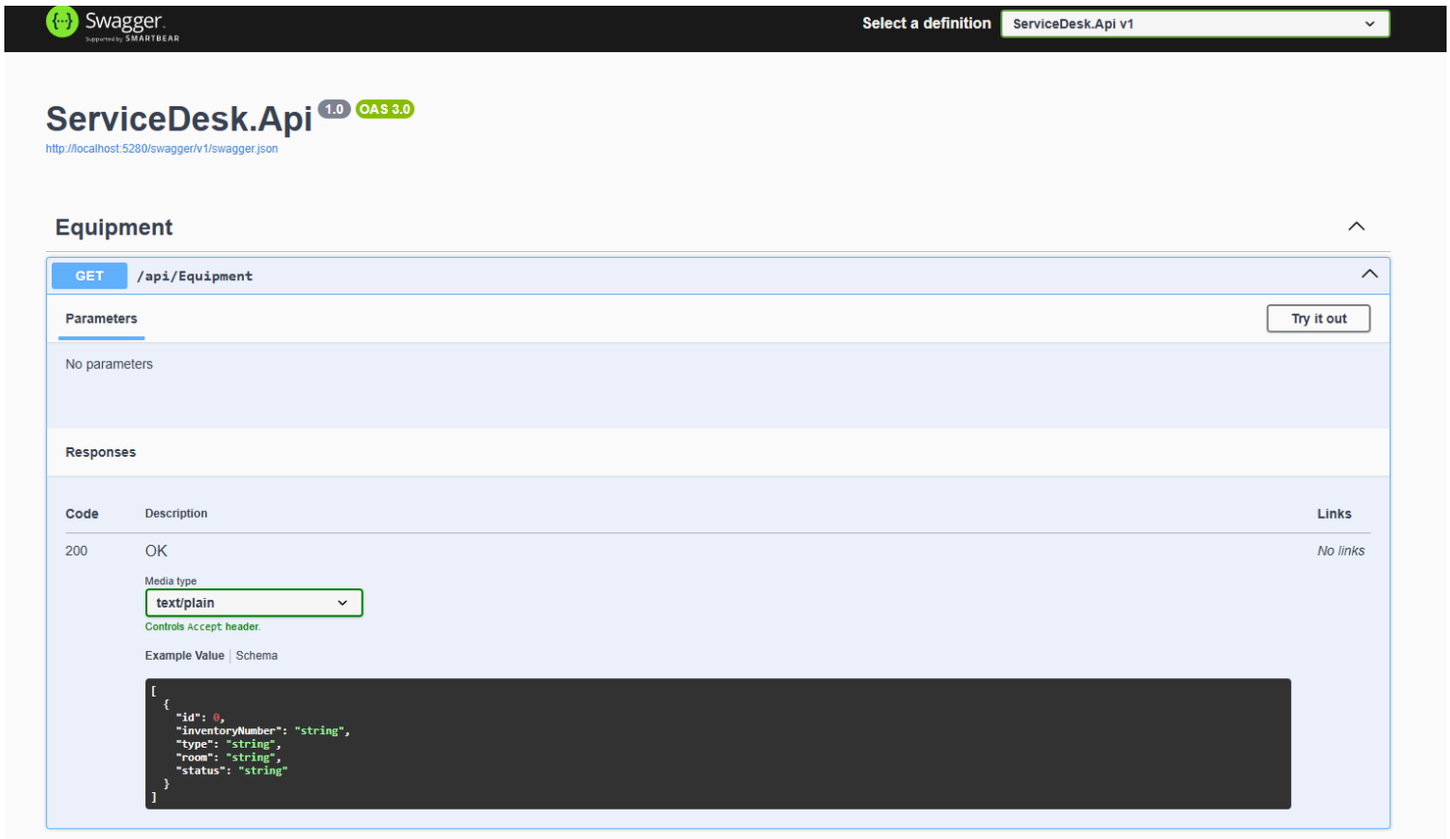

Рисунок 11 – GET /api/equipment

2. POST /api/equipment

Создание новой записи об оборудовании.

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

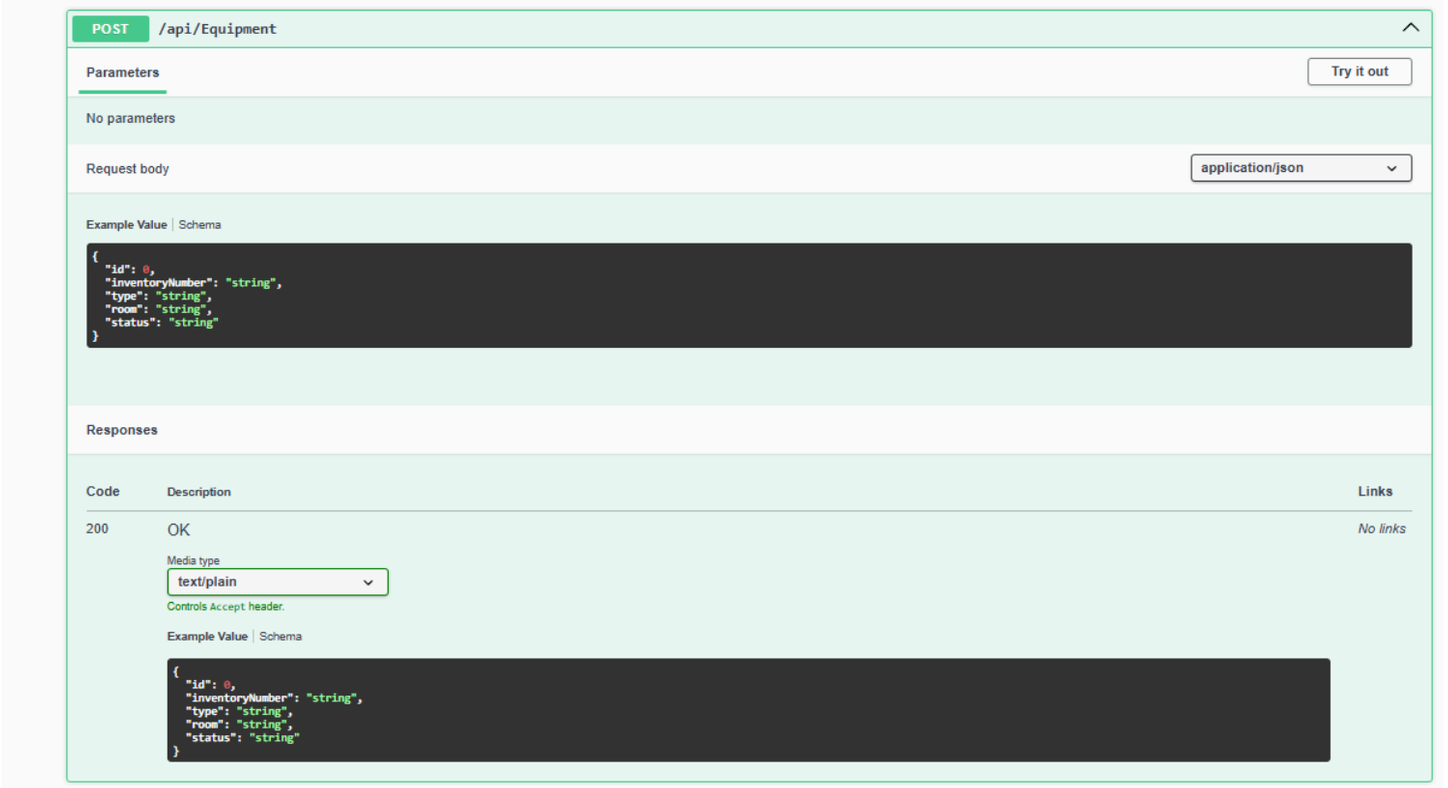

Рисунок 12 – POST /api/equipment

3. PUT /api/equipment/{id}

Обновление существующей записи по идентификатору.

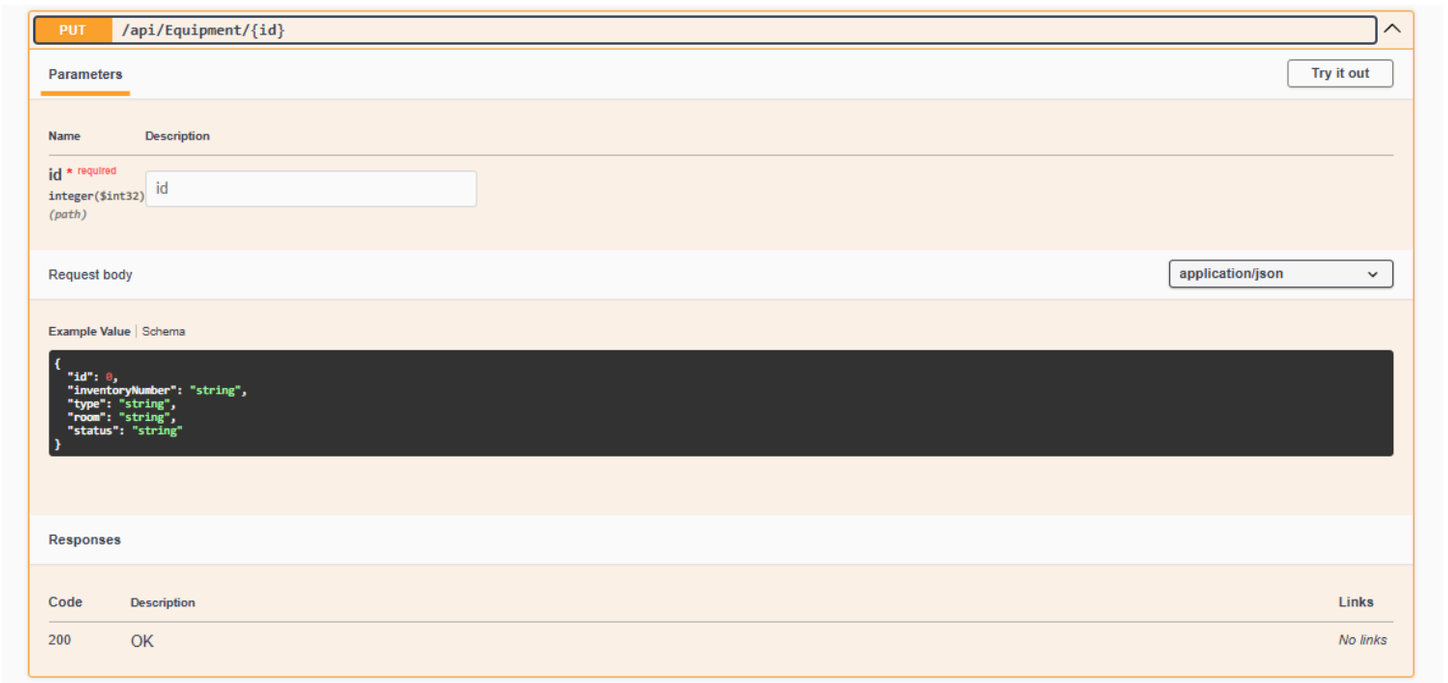

Рисунок 13 – PUT /api/equipment/{id}

4. DELETE /api/equipment/{id}

Удаление записи по идентификатору.

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

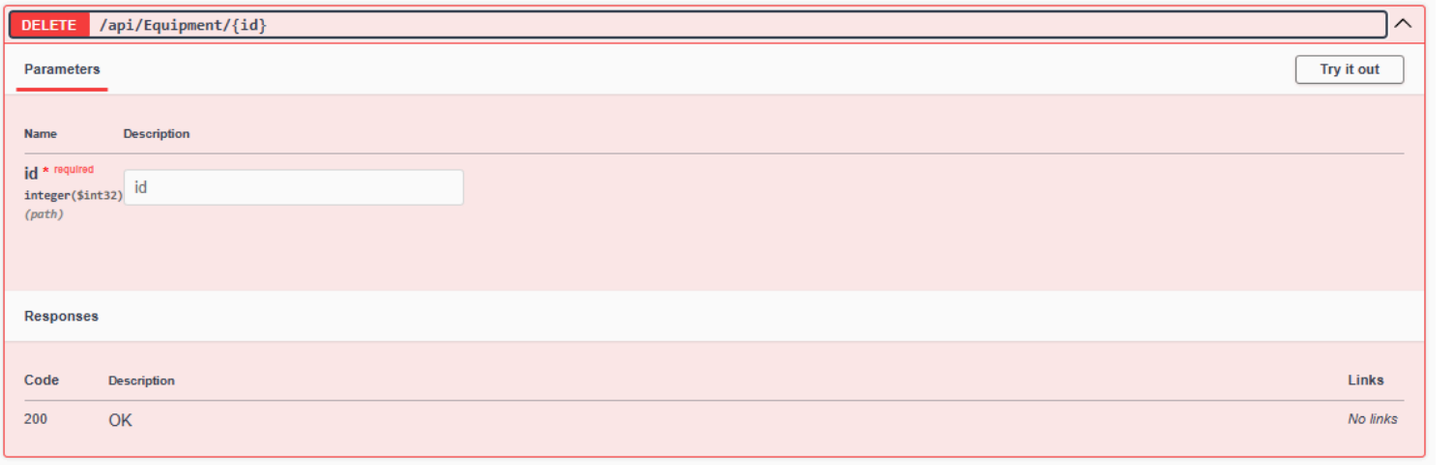

Рисунок 14 – DELETE /api/equipment/{id}

Если каждое действие работает — CRUD реализован корректно.

Пример тестового проекта

Пример тестового проекта с реализованным Web API и базовой инфраструктурой доступен по ссылке:

https://github.com/OlgaKraven/DKIP.PP.PM.06-testProject.git

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

2.3. Блок. Реализация базового CRUD-функционала

ОГЛАВЛЕНИЕ РАЗДЕЛА 2.3

- 2.3.0. Что такое CRUD в контексте Web API
- 2.3.1. Проверяем модели
- 2.3.2. Контекст данных (для CRUD через EF Core)
- 2.3.3. CRUD-контроллер для Equipment
- 2.3.4. CRUD-контроллер для ServiceRequest
- 2.3.5. Методика проверки CRUD через Swagger
- 2.3.6. Что студент должен зафиксировать в отчёте

2.3.0. Что такое CRUD в контексте Web API

Для каждой сущности (например, `Equipment`):

- Create – создать запись → HTTP **POST**
- Read – прочитать записи → HTTP **GET** (список и по id)
- Update – изменить запись → HTTP **PUT** (или **PATCH**)
- Delete – удалить запись → HTTP **DELETE**

В API это обычно 4–5 эндпоинтов вида:

- `GET /api/equipment` — получить список
- `GET /api/equipment/{id}` — получить по id
- `POST /api/equipment` — создать
- `PUT /api/equipment/{id}` — обновить
- `DELETE /api/equipment/{id}` — удалить

2.3.2. Проверяем модели

У тебя должны быть модели (в папке `Models`):

Помогу с учебным заданием без предоплаты. Оплата после выполнения

Нажмите на мессенджер, чтобы написать

Telegram ↗ WhatsApp ↗ MAX ↗

+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

Models/Equipment.cs

```
namespace ServiceDesk.Api.Models
{
    public class Equipment
    {
        public int Id { get; set; }
        public string InventoryNumber { get; set; } = "";
        public string Type { get; set; } = "";
        public string Room { get; set; } = "";
        public string Status { get; set; } = "Работает";
    }
}
```

Models/ServiceRequest.cs

```
namespace ServiceDesk.Api.Models
{
    public class ServiceRequest
    {
        public int Id { get; set; }
        public string Title { get; set; } = "";
        public string Description { get; set; } = "";
        public string Priority { get; set; } = "Средний";
        public string Status { get; set; } = "Новая";
        public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
        public DateTime? ClosedAt { get; set; }

        public int? EquipmentId { get; set; }
        public Equipment? Equipment { get; set; }
    }
}
```

2.3.2. Контекст данных (для CRUD через EF Core)

Файл `Data/AppDbContext.cs`:

```
using Microsoft.EntityFrameworkCore;
using ServiceDesk.Api.Models;

namespace ServiceDesk.Api.Data
{
    public class AppDbContext : DbContext
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

```
    {
        public AppDbContext(DbContextOptions<AppDbContext> options)
            : base(options)
        {
        }

        public DbSet<Equipment> Equipment { get; set; }
        public DbSet<ServiceRequest> ServiceRequests { get; set; }
    }
}
```

В `Program.cs` уже должна быть регистрация (мы добавляли):

```
builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseInMemoryDatabase("ServiceDeskDb"));
```

2.3.3. CRUD-контроллер для Equipment

Цель: показать полный CRUD на одной сущности, чтобы студент по аналогии сделал вторую.

Файл `Controllers/EquipmentController.cs`:

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using ServiceDesk.Api.Data;
using ServiceDesk.Api.Models;

namespace ServiceDesk.Api.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class EquipmentController : ControllerBase
    {
        private readonly AppDbContext _context;

        public EquipmentController(AppDbContext context)
        {
            _context = context;
        }

        // GET: api/equipment
        [HttpGet]
        public async Task<ActionResult<IEnumerable<Equipment>>> GetAll()
        {
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

```
    var items = await _context.Equipment.ToListAsync();
    return Ok(items);
}

// GET: api/equipment/5
[HttpGet("{id:int}")]
public async Task<ActionResult<Equipment>> GetById(int id)
{
    var item = await _context.Equipment.FindAsync(id);
    if (item == null)
        return NotFound();

    return Ok(item);
}

// POST: api/equipment
[HttpPost]
public async Task<ActionResult<Equipment>> Create(Equipment model)
{
    _context.Equipment.Add(model);
    await _context.SaveChangesAsync();

    // Возвращаем 201 Created с Location
    return CreatedAtAction(nameof(GetById), new { id = model.Id }, model);
}

// PUT: api/equipment/5
[HttpPut("{id:int}")]
public async Task<IActionResult> Update(int id, Equipment model)
{
    if (id != model.Id)
        return BadRequest("Id в пути и в теле запроса не совпадают.");

    // Обновляем всё целиком
    _context.Entry(model).State = EntityState.Modified;

    await _context.SaveChangesAsync();
    return NoContent(); // 204
}

// DELETE: api/equipment/5
[HttpDelete("{id:int}")]
public async Task<IActionResult> Delete(int id)
{
    var item = await _context.Equipment.FindAsync(id);
    if (item == null)
        return NotFound();
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения

Нажмите на мессенджер, чтобы написать

Telegram ↗ WhatsApp ↗ MAX ↗

+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

```
            _context.Equipment.Remove(item);
            await _context.SaveChangesAsync();

            return NoContent();
        }
    }
}
```

Что важно методически

- `[ApiController]` + `[Route("api/[controller]")]` → маршрут `api/equipment`
- Возвращаемые коды состояния:
  - `200 OK` — успешное чтение
  - `201 Created` — при создании
  - `204 NoContent` — при успешном обновлении/удалении
  - `404 NotFound` — если сущность не найдена
  - `400 BadRequest` — если некорректные данные (например, разные id)

2.3.4. CRUD-контроллер для ServiceRequest

Теперь студент делает по аналогии для сущности заявок.

Файл `Controllers/ServiceRequestsController.cs`:

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using ServiceDesk.Api.Data;
using ServiceDesk.Api.Models;

namespace ServiceDesk.Api.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class ServiceRequestsController : ControllerBase
    {
        private readonly AppDbContext _context;

        public ServiceRequestsController(AppDbContext context)
        {
            _context = context;
        }

        // GET: api/servicerequests
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

```
[HttpGet]
public async Task<ActionResult<IEnumerable<ServiceRequest>>> GetAll()
{
    var items = await _context.ServiceRequests
        .Include(r => r.Equipment) // при желании подгружаем оборудование
        .ToListAsync();

    return Ok(items);
}

// GET: api/servicerequests/5
[HttpGet("{id:int}")]
public async Task<ActionResult<ServiceRequest>> GetById(int id)
{
    var item = await _context.ServiceRequests
        .Include(r => r.Equipment)
        .FirstOrDefaultAsync(r => r.Id == id);

    if (item == null)
        return NotFound();

    return Ok(item);
}

// POST: api/servicerequests
[HttpPost]
public async Task<ActionResult<ServiceRequest>> Create(ServiceRequest model)
{
    model.Id = 0;                   // на всякий случай
    model.CreatedAt = DateTime.UtcNow;
    if (model.Status == null || model.Status == "")
        model.Status = "Новая";

    _context.ServiceRequests.Add(model);
    await _context.SaveChangesAsync();

    return CreatedAtAction(nameof(GetById), new { id = model.Id }, model);
}

// PUT: api/servicerequests/5
[HttpPut("{id:int}")]
public async Task<IActionResult> Update(int id, ServiceRequest model)
{
    if (id != model.Id)
        return BadRequest("Id в пути и в теле запроса не совпадают.");

    _context.Entry(model).State = EntityState.Modified;
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения

Нажмите на мессенджер, чтобы написать

Telegram ↗ WhatsApp ↗ MAX ↗

+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

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

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

            _context.ServiceRequests.Remove(item);
            await _context.SaveChangesAsync();

            return NoContent();
        }

        // Дополнительно: получить только открытые заявки
        // GET: api/servicerequests/open
        [HttpGet("open")]
        public async Task<ActionResult<IEnumerable<ServiceRequest>>> GetOpen()
        {
            var items = await _context.ServiceRequests
                .Where(r => r.Status != "Выполнена" && r.Status != "Отменена")
                .ToListAsync();

            return Ok(items);
        }
    }
}
```

2.3.5. Методика проверки CRUD через Swagger

1. Запусти приложение:

```
dotnet run
```
SCRIPT

2. Открой Swagger:

```
http://localhost:XXXX/swagger
```
SCRIPT

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

(порт смотри в консоли)

Последовательность проверки для Equipment

```
1. POST /api/Equipment

{
  "inventoryNumber": "PC-001",
  "type": "ПК",
  "room": "101",
  "status": "Работает"
}

2. GET /api/Equipment
→ должен вернуть массив с созданной записью

3. GET /api/Equipment/1
→ получить по id

4. PUT /api/Equipment/1

{
  "id": 1,
  "inventoryNumber": "PC-001",
  "type": "ПК",
  "room": "102",
  "status": "Требует ремонта"
}

5. DELETE /api/Equipment/1

6. GET /api/Equipment
→ запись исчезла
```

Проверка ServiceRequests

```
1. Сначала создай ещё одно Equipment, чтобы была техника.

2. POST /api/ServiceRequests

{
  "title": "Не включается компьютер",
  "description": "ПК в кабинете 101 не включается",
  "priority": "Высокий",
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

```
  "status": "Новая",
  "equipmentId": 1
}

3. GET /api/ServiceRequests

4. GET /api/ServiceRequests/open

5. PUT /api/ServiceRequests/1
— сменить status на "В работе" или "Выполнена"

6. DELETE /api/ServiceRequests/1
```

2.3.6. Что студент должен зафиксировать в отчёте по этому блоку

1. Кратко описать сущности и их поля.
2. Привести таблицу эндпоинтов API:

Сущность	Метод	Маршрут	Назначение
Equipment	GET	/api/equipment	Получить список оборудования
Equipment	GET	/api/equipment/{id}	Получить по id
Equipment	POST	/api/equipment	Создать запись
Equipment	PUT	/api/equipment/{id}	Обновить запись
Equipment	DELETE	/api/equipment/{id}	Удалить запись
ServiceRequest	GET	/api/servicerequests	Получить список заявок
ServiceRequest	GET	/api/servicerequests/{id}	Получить заявку по id
ServiceRequest	POST	/api/servicerequests	Создать заявку
ServiceRequest	PUT	/api/servicerequests/{id}	Обновить заявку
ServiceRequest	DELETE	/api/servicerequests/{id}	Удалить заявку
ServiceRequest	GET	/api/servicerequests/open	Получить только открытые заявки

3. Сделать скриншоты Swagger с:
    - списком эндпоинтов;

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

- примерами запросов и ответов (`201` , `200` , `204` , `404`).

Помогу с учебным заданием без предоплаты. Оплата после выполнения

Нажмите на мессенджер, чтобы написать

Telegram ↗ WhatsApp ↗ MAX ↗

+7 (924) 305-23-08 · Пишите в мессенджер

disynergy.ru

2.4. Контейнеризация ASP.NET Core Web API с помощью Docker (multi-stage)

ОГЛАВЛЕНИЕ РАЗДЕЛА 2.4

- 2.4.0. Цель блока
- 2.4.1. Подготовка: проверка Docker
- 2.4.2. Структура проекта (важно для Dockerfile)
- 2.4.3. Создание файла .dockerignore
- 2.4.4. Multi-stage Dockerfile для .NET Web API
- 2.4.5. Сборка Docker-образа
- 2.4.6. Запуск контейнера и проверка Swagger
- 2.4.7. Базовые команды управления контейнером
- 2.4.8. Что студент должен зафиксировать по этому блоку

2.4.0. Цель блока

Студент должен:

1. Создать Dockerfile с multi-stage сборкой для проекта `ServiceDesk.Api`.
2. На его основе собрать Docker-образ.
3. Запустить контейнер и открыть Swagger из контейнера, а не из локального `dotnet run`.

2.4.1. Подготовка: проверка Docker

1. Установить Docker Desktop (Windows) или Docker Engine (Linux).
2. Проверить в терминале:

```
SCRIPT
docker --version
```

Ожидаемый результат: вывод версии Docker, например `24.x.x`.

Помогу с учебным заданием без предоплаты. Оплата после выполнения

Нажмите на мессенджер, чтобы написать

Telegram ↗ WhatsApp ↗ MAX ↗

+7 (924) 305-23-08 · Пишите в мессенджер

disynergy.ru

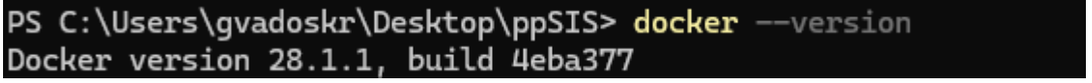

Рисунок 1 – Пример результата

Если команда не узнаётся — Docker не установлен или не добавлен в `PATH`.

2.4.2. Структура проекта (важно для Dockerfile)

Допустим, сейчас структура проекта выглядит так:

```
ppSIS/
  ServiceDesk.Api/
    ServiceDesk.Api.csproj
    Program.cs
    Controllers/
    Models/
    Data/
    ...
```

Важно: файл `Dockerfile` размещается в папке `ServiceDesk.Api` рядом с файлом `.csproj`.

2.4.3. Создание файла .dockerignore

В папке `ServiceDesk.Api` создайте файл `.dockerignore` со следующим содержимым:

```
bin/
obj/
*.user
*.swp
*.suo
*.csproj.user
.vs/
.git/
.gitignore
Dockerfile
docker-compose*
*.md
```

Назначение: чтобы Docker не включал в контекст сборки временные файлы и служебные директории (`bin/`, `obj/`, `.git` и т.п.).

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

.htm | task1-webapi.html | task2-crud.html | .dockerignore

```
bin/
obj/
*.user
*.swp
*.suo
*.csproj.user
.vs/
.git/
.gitignore
Dockerfile
docker-compose*
*.md
```

Рисунок 2 – Пример файла .dockerignore

2.4.5. Multi-stage Dockerfile для .NET Web API

В папке `ServiceDesk.Api` создайте файл `Dockerfile` (без расширения) и вставьте:

```
# Этап 1: базовый рантайм ASP.NET Core 9
FROM mcr.microsoft.com/dotnet/aspnet:9.0 AS base
WORKDIR /app

# Приложение будет слушать на 8080 внутри контейнера
EXPOSE 8080
ENV ASPNETCORE_URLS=http://+:8080

# Чтобы Swagger работал в контейнере, оставляем окружение Development
ENV ASPNETCORE_ENVIRONMENT=Development

# Этап 2: SDK для сборки .NET 9
FROM mcr.microsoft.com/dotnet/sdk:9.0 AS build
WORKDIR /src

# Копируем только csproj
COPY ["ServiceDesk.Api.csproj", "./"]

# Восстанавливаем зависимости
RUN dotnet restore "ServiceDesk.Api.csproj"
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

```
# Копируем остальной код
COPY . .

# Публикуем (Release)
RUN dotnet publish "ServiceDesk.Api.csproj" -c Release -o /app/publish /p:UseAppHost=false

# Этап 3: финальный образ (минимальный)
FROM base AS final
WORKDIR /app

# Копируем опубликованные файлы
COPY --from=build /app/publish .

# Точка входа
ENTRYPOINT ["dotnet", "ServiceDesk.Api.dll"]
```

Краткое пояснение

- `base` — лёгкий образ с рантаймом (для запуска приложения).
- `build` — образ с SDK (для сборки и публикации).
- `final` — финальный образ на основе `base`, в который попадают только опубликованные файлы.

2.4.5. Сборка Docker-образа

> **Важно**
>
> Перед началом работы запустите Docker Desktop. Если есть проблемы с его работой, закройте программы, которые могут конфликтовать с сетевыми портами (например, XAMPP, Discord).

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

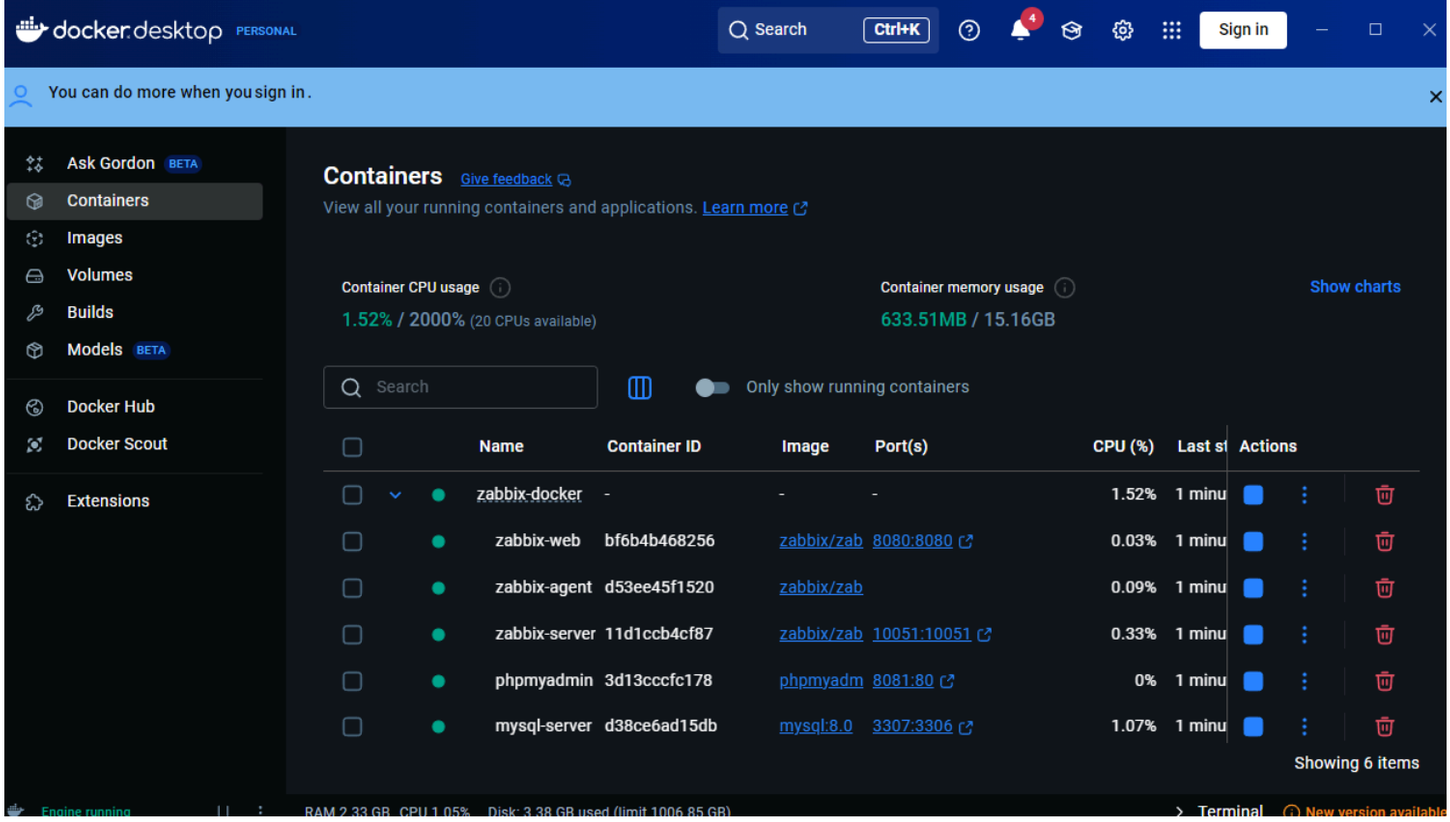

Рисунок 3 – Запущенный Docker

1. Открыть терминал в папке проекта:

```
SCRIPT

cd C:\Users\gvadoskr\Desktop\ppSIS\ServiceDesk.Api
```

2. Выполнить команду сборки образа:

```
SCRIPT

docker build -t servicedesk-api:dev .
```

Где:

- `-t servicedesk-api:dev` — имя и тег образа (можно изменить по необходимости);
- `.` — контекст сборки (текущая папка с `Dockerfile`).

Ожидаемый результат в конце лога:

```
Successfully built <hash>
Successfully tagged servicedesk-api:dev
```

Типичная ошибка при сборке

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

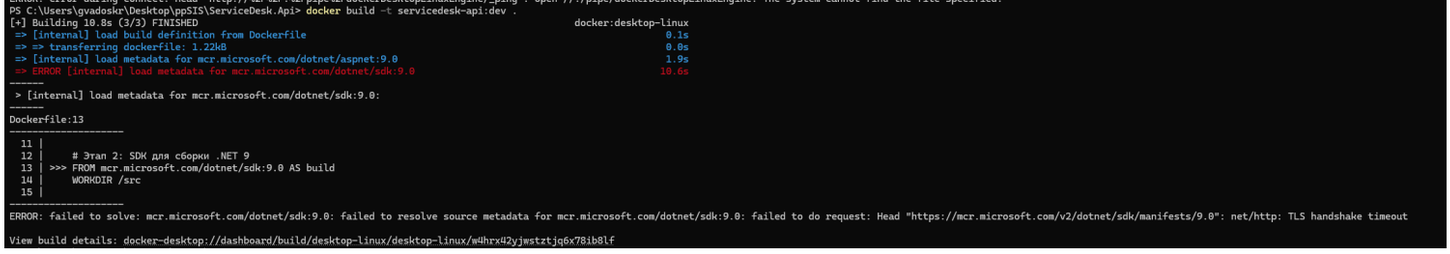

Рисунок 4 – Пример ошибки

Сообщение вида:

```
failed to resolve source metadata for mcr.microsoft.com/dotnet/sdk:9.0
net/http: TLS handshake timeout
```

означает не проблему с Dockerfile, а:

- Docker не смог скачать базовый образ `mcr.microsoft.com/dotnet/sdk:9.0`
- причина: блокировка доступа в интернет, firewall, прокси или ошибка TLS.

Способы решения

1. Повторить команду:

```
SCRIPT
docker build -t servicedesk-api:dev .
```

2. Включить VPN на компьютере.

   Docker Desktop использует сетевой стек Windows, поэтому VPN часто решает проблему загрузки образов.

3. Использовать зеркала Microsoft Container Registry (MCR Mirror). Заменить строки в `Dockerfile`:

```
# Было
FROM mcr.microsoft.com/dotnet/aspnet:9.0 AS base
FROM mcr.microsoft.com/dotnet/sdk:9.0 AS build

# Стало
FROM registry-1.docker.io/microsoft/dotnet-aspnet:9.0 AS base
FROM registry-1.docker.io/microsoft/dotnet-sdk:9.0 AS build
```

4. Проверить доступность образа вручную:

```
SCRIPT
docker pull mcr.microsoft.com/dotnet/sdk:9.0
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

Если снова ошибка, проверить доступность сайта:

```
SCRIPT

https://mcr.microsoft.com
```

5. Проверить сетевые настройки Docker Desktop:
   - Settings → Resources → Network;
   - попробовать включить или отключить опции DNS и Proxy в зависимости от текущей конфигурации.
6. При необходимости перейти на стабильную версию .NET 8 LTS:
   - в `.csproj`: `<TargetFramework>net8.0</TargetFramework>`;
   - в `Dockerfile`:

```
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base
FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
```



Рисунок 5 – Пример перехода по ссылке из консоли для проверки работы приложения

2.4.6. Запуск контейнера и проверка Swagger

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Важно

Если у вас уже запущены другие контейнеры со старых практик, может возникнуть конфликт по портам. Перед выполнением далее убедитесь, что старые контейнеры остановлены.



Рисунок 6 – Выключенный контейнер

Запустите контейнер на порту 8080:

```
SCRIPT

docker run --name servicedesk-api-container -p 8080:8080 servicedesk-api:dev
```

Расшифровка:

- `--name servicedesk-api-container` — имя контейнера;
- `-p 8080:8080` — проброс порта хост:контейнер;
- `servicedesk-api:dev` — наш образ.

Если всё в порядке, в логах будет что-то вроде:

```
Now listening on: http://0.0.0.0:8080
Application started. Press Ctrl+C to shut down.
```

Теперь можно открыть в браузере:

```
SCRIPT

http://localhost:8080/swagger
```

Либо запустить контейнер через Docker Desktop, нажав кнопку запуска и перейдя по ссылке в столбце Port.

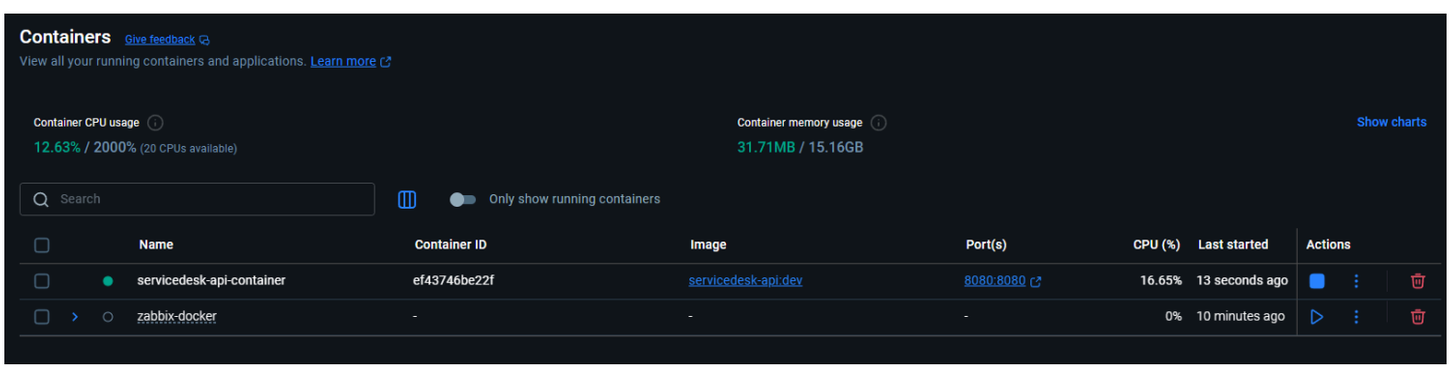

Рисунок 7 – Пример запуска из Docker Desktop

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Swagger должен открыться уже из контейнера, а не из локального `dotnet run`.

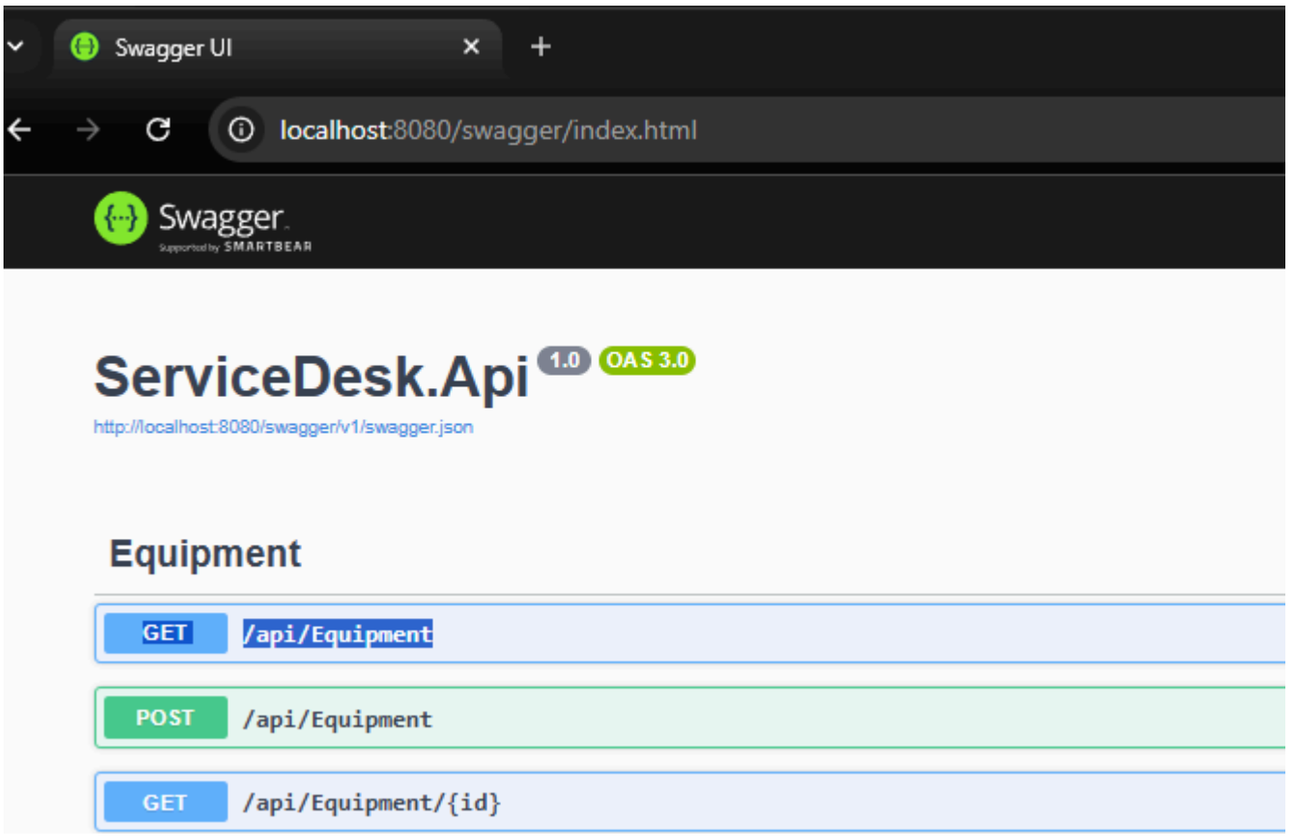

Рисунок 8 – Пример работающего приложения

2.4.7. Базовые команды управления контейнером

Для отчёта и практики полезно показать студентам базовые команды:

- Посмотреть запущенные контейнеры:

```
SCRIPT
docker ps
```

- Остановить контейнер:

```
SCRIPT
docker stop servicedesk-api-container
```

- Запустить контейнер снова:

```
SCRIPT
docker start servicedesk-api-container
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

- Удалить контейнер:

```
SCRIPT

docker rm servicedesk-api-container
```

- Удалить образ:

```
SCRIPT

docker rmi servicedesk-api:dev
```

2.4.8. Что студент должен зафиксировать по этому блоку

В отчёт или методические материалы студент фиксирует:

1. Текст `Dockerfile` с кратким пояснением:
   - какие этапы используются (`build`, `base`, `final`);
   - зачем нужен multi-stage (меньший финальный образ, разделение сборки и запуска).
2. Основные команды:

```
SCRIPT

docker build -t servicedesk-api:dev .
docker run -p 8080:8080 servicedesk-api:dev
```

3. Скриншот Swagger, работающего по адресу:

```
SCRIPT

http://localhost:8080/swagger
```

4. Команды управления контейнером:

```
SCRIPT

docker ps
docker stop servicedesk-api-container
docker start servicedesk-api-container
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

```
docker rm servicedesk-api-container
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения

Нажмите на мессенджер, чтобы написать

Telegram ↗ WhatsApp ↗ MAX ↗

+7 (924) 305-23-08 · Пишите в мессенджер

disynergy.ru

2.5. Подключение PostgreSQL через docker-compose

ОГЛАВЛЕНИЕ РАЗДЕЛА 2.5

- 2.5.0. Цель блока
- 2.5.1. Добавляем провайдер PostgreSQL в проект
- 2.5.2. Настраиваем строку подключения
- 2.5.3. Обновляем Program.cs под PostgreSQL
- 2.5.4. Dockerfile (сверка)
- 2.5.5. Создаём docker-compose.yml с API и PostgreSQL
- 2.5.6. Запуск всего проекта через docker-compose
- 2.5.7. Полезные команды для работы

2.5.0. Цель блока

В этом блоке студент должен:

- заменить InMemory-БД на PostgreSQL;
- настроить подключение через EF Core + Npgsql;
- настроить `docker-compose`, чтобы поднимать:
  - контейнер с API;
  - контейнер с PostgreSQL;
  - общий volume для данных.

2.5.1. Добавляем провайдер PostgreSQL в проект

Файл `ServiceDesk.Api.csproj` сейчас может выглядеть так:

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>net9.0</TargetFramework>
    <Nullable>enable</Nullable>
```


Помогу с учебным заданием без предоплаты. Оплата после выполнения

Нажмите на мессенджер, чтобы написать

Telegram ↗ WhatsApp ↗ MAX ↗

+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

```
    <ImplicitUsings>enable</ImplicitUsings>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Swashbuckle.AspNetCore" Version="10.0.1" />
    <PackageReference Include="Microsoft.EntityFrameworkCore" Version="8.0.0" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.InMemory" Version="8.0.0" />
  </ItemGroup>

</Project>
```

Добавим провайдер для PostgreSQL `Npgsql.EntityFrameworkCore.PostgreSQL`:

```
<ItemGroup>
  <PackageReference Include="Swashbuckle.AspNetCore" Version="10.0.1" />
  <PackageReference Include="Microsoft.EntityFrameworkCore" Version="8.0.0" />
  <PackageReference Include="Microsoft.EntityFrameworkCore.InMemory" Version="8.0.0" />
  <PackageReference Include="Npgsql.EntityFrameworkCore.PostgreSQL" Version="8.0.0" />
</ItemGroup>
```

Либо установить пакет через команду:

SCRIPT

```
dotnet add package Npgsql.EntityFrameworkCore.PostgreSQL --version 8.0.0
```

2.5.2. Настраиваем строку подключения

Сделаем так, чтобы в Docker строка подключения передавалась через переменную окружения, а в коде мы просто брали `DefaultConnection`.

В `Program.cs` мы будем вызывать:

SCRIPT

```
builder.Configuration.GetConnectionString("DefaultConnection")
```

ASP.NET Core умеет автоматически подтягивать это значение из переменной окружения с именем:

SCRIPT

```
ConnectionStrings__DefaultConnection=...
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

То есть в файле `appsettings.json` строку подключения можно не задавать — она будет переопределена в `docker-compose`. Если хочется, можно добавить её для наглядности:

appsettings.json (опционально)

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Host=localhost;Port=5432;Database=servicedeskdb;Username=servicedesk;Passwor
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  }
}
```

В контейнере эта строка подключения будет переписана через переменную окружения.

2.5.3 . Обновляем Program.cs под PostgreSQL

Ранее в `Program.cs` использовалась InMemory-БД (метод `UseInMemoryDatabase`). Теперь заменим её на PostgreSQL и `UseNpgsql`.

Пример полного `Program.cs` (минимальный вариант без лишнего кода):

```
using Microsoft.AspNetCore.Builder;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using ServiceDesk.Api.Data;

var builder = WebApplication.CreateBuilder(args);

// Контроллеры
builder.Services.AddControllers();

// Подключение PostgreSQL через EF Core
builder.Services.AddDbContext<AppDbContext>(options =>
{
    var connectionString = builder.Configuration.GetConnectionString("DefaultConnection");
    options.UseNpgsql(connectionString);
});
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

```
// Swagger
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

// Автоматическое создание БД (для учебного примера)
using (var scope = app.Services.CreateScope())
{
    var db = scope.ServiceProvider.GetRequiredService<AppDbContext>();
    db.Database.EnsureCreated();
}

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

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();
```

Важно:

- вверху должны быть `using Microsoft.EntityFrameworkCore;` и `using ServiceDesk.Api.Data;`;
- в `AppDbContext` изменений не требуется — он не привязан к конкретному провайдеру БД.

2.5.4. Dockerfile (сверка)

Dockerfile у тебя уже должен быть настроен. Ниже — сверка для проекта на `net9.0` (ASP.NET Core 9), аналогично предыдущему блоку.

```
# Этап 1: базовый рантайм ASP.NET Core 9
FROM mcr.microsoft.com/dotnet/aspnet:9.0 AS base
WORKDIR /app

EXPOSE 8080
ENV ASPNETCORE_URLS=http://+:8080
ENV ASPNETCORE_ENVIRONMENT=Development
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

```
# Этап 2: SDK для сборки .NET 9
FROM mcr.microsoft.com/dotnet/sdk:9.0 AS build
WORKDIR /src

COPY ["ServiceDesk.Api.csproj", "./"]
RUN dotnet restore "ServiceDesk.Api.csproj"

COPY . .
RUN dotnet publish "ServiceDesk.Api.csproj" -c Release -o /app/publish /p:UseAppHost=false

# Этап 3: финальный образ
FROM base AS final
WORKDIR /app

COPY --from=build /app/publish .

ENTRYPOINT ["dotnet", "ServiceDesk.Api.dll"]
```

2.5.5. Создаём docker-compose.yml с API и PostgreSQL

Файл `docker-compose.yml` можно положить рядом с Dockerfile (в `ServiceDesk.Api`) или на уровень выше. Важно, чтобы `build: .` указывал на папку, где лежат `Dockerfile` и `ServiceDesk.Api.csproj`.

Пример `docker-compose.yml`:

```
version: "3.9"

services:
  api:
    build: .
    container_name: servicedesk-api
    ports:
      - "8080:8080"
    environment:
      - ASPNETCORE_ENVIRONMENT=Development
      - ASPNETCORE_URLS=http://+:8080
      # Строка подключения для EF Core:
      - ConnectionStrings__DefaultConnection=Host=postgres;Port=5432;Database=servicedeskdb;Username=s
    depends_on:
      - postgres

  postgres:
    image: postgres:16-alpine
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

```
    container_name: servicedesk-postgres
    restart: always
    environment:
      - POSTGRES_DB=servicedeskdb
      - POSTGRES_USER=servicedesk
      - POSTGRES_PASSWORD=servicedeskpwd
    volumes:
      - db-data:/var/lib/postgresql/data

volumes:
  db-data:
```

Ключевые моменты

- Сервис `api`:
  - `build: .` — собирает образ из Dockerfile;
  - `ports: "8080:8080"` — публикует API наружу на порт 8080;
  - `ConnectionStrings__DefaultConnection=...` — переменная окружения, которая подхватывается в `builder.Configuration.GetConnectionString("DefaultConnection")`;
  - `Host=postgres` — имя хоста равно имени сервиса БД в `docker-compose`.
- Сервис `postgres`:
  - образ `postgres:16-alpine`;
  - параметры `POSTGRES_DB`, `POSTGRES_USER`, `POSTGRES_PASSWORD` совпадают с строкой подключения;
  - `volume db-data` для сохранения данных БД между перезапусками.

2.5.6. Запуск всего проекта через docker-compose

Из папки, где лежит `docker-compose.yml`, выполните:

SCRIPT

```
docker compose up -d
```

Если используется старая версия, можно:

SCRIPT

```
docker-compose up -d
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Проверка состояния сервисов:

```
SCRIPT
docker compose ps
```

Должны быть два сервиса: `api` и `postgres`.

Откройте в браузере:

```
SCRIPT
http://localhost:8080/swagger
```

Если всё настроено верно:

- API стартует в контейнере `api`;
- EF Core создаёт схему БД в PostgreSQL (через `EnsureCreated()`);
- Swagger показывает те же CRUD-эндпоинты, но данные теперь хранятся в PostgreSQL, а не в памяти.

2.5.7. Полезные команды для работы

Команды для управления всей связкой API + PostgreSQL:

- Остановить контейнеры (но оставить volume и данные):

```
SCRIPT
docker compose down
```

- Полностью удалить контейнеры и volume (данные БД будут удалены):

```
SCRIPT
docker compose down -v
```

- Логи API:

```
SCRIPT
docker compose logs api
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

- Логи БД:

```
SCRIPT

docker compose logs postgres
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

2.6. Выполнить обновление версии приложения в процессе сопровождения

ОГЛАВЛЕНИЕ РАЗДЕЛА 2.6

- 2.6.0. Логика блока
- 2.6.1. Цель блока
- 2.6.2. Внести небольшое изменение в код
- 2.6.3. Сборка новой версии Docker-образа
- 2.6.4. Обновление версии в docker-compose.yml
- 2.6.5. Перезапуск docker-compose
- 2.6.6. Проверка обновления версии
- 2.6.7. Методические тезисы для отчёта
- 2.6.8. Если произошла ошибка «port is already allocated»

2.6.0. Этот шаг обучает DevOps-логике

Процесс обновления приложения в контейнерной среде всегда включает такие шаги:

1. Обновление исходного кода.
2. Пересборка Docker-образа с новым тегом.
3. Обновление контейнера через `docker-compose`, при котором:
   - база данных сохраняется (она в volume);
   - API перезапускается в новой версии.

Цель — показать студентам, как выглядит реальная логика CI/CD в миниатюре.

2.6.1. Цель блока

Студент должен уметь:

- внести изменение в приложение (логика/Swagger/модель);

Помогу с учебным заданием без предоплаты. Оплата после выполнения

Нажмите на мессенджер, чтобы написать

Telegram ↗ WhatsApp ↗ MAX ↗

+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

- собрать новый Docker-образ с новым тегом;
- обновить docker-compose на новую версию образа;
- перезапустить приложение без потери данных PostgreSQL.

2.6.2. Внести небольшое изменение в код

Чтобы обновление было заметно визуально, внесём простое изменение в контроллер `EquipmentController`.

Например: добавим лог при вызове GET-метода.

Было:

```
public async Task<ActionResult<IEnumerable<Equipment>>> GetAll()
```

Стало:

```
[HttpGet]
public async Task<ActionResult<IEnumerable<Equipment>>> GetAll()
{
    Console.WriteLine(">>> API VERSION 2: GetAll() called");
    return await _context.Equipment.ToListAsync();
}
```

Также можно:

- изменить Swagger-описание;
- добавить поле в модель;
- изменить текстовое сообщение.

Главное — чтобы после обновления студент увидел разницу.

2.6.3. Сборка новой версии Docker-образа

Теперь собираем образ новой версии, не перезаписывая старый:

```
SCRIPT

docker build -t servicedesk-api:2.0 .
```

Где:

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

- `servicedesk-api` — имя образа;
- `2.0` — новая версия (можно: `1.1`, `v2`, `2025-02`).

2.6.4. Обновление версии в docker-compose.yml

Открой `docker-compose.yml` и поменяй тег образа.

Было:

```
api:
  image: servicedesk-api:dev
```

Стало:

```
api:
  image: servicedesk-api:2.0
```

При использовании сборки внутри compose можно оставить:

```
api:
  build: .
  image: servicedesk-api:2.0
```

2.6.5. Перезапуск docker-compose

Старый контейнер удалять вручную не требуется. Достаточно выполнить:

SCRIPT

```
docker compose down
```

SCRIPT

```
docker compose up -d --build
```

Это:

- остановит старый контейнер API,
- создаст контейнер новой версии,
- оставит PostgreSQL как есть,
- сохранит volume → данные не пропадут.

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

2.6.6. Проверка обновления

Теперь проверяем:

1. Открыть Swagger:

```
http://localhost:8080/swagger
```
SCRIPT

2. Выполнить запрос:

```
GET /api/equipment
```
SCRIPT

3. Посмотреть логи контейнера:

```
docker compose logs api
```
SCRIPT

В логах должно быть:

```
>>> API VERSION 2: GetAll() called
```
SCRIPT

Это подтверждает, что:

- контейнер обновился;
- работает новая версия API;
- данные PostgreSQL сохранились.

2.6.7. Методические тезисы (должны попасть в отчёт)

Студент должен описать следующие шаги:

Этап	Описание
1	Внести изменения в исходный код API
2	Собрать новый Docker-образ: docker build -t servicedesk-api:2.0 .
3	Обновить тег образа в `docker-compose.yml`
4	Перезапустить сервис:

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Этап	Описание
	docker compose down && docker compose up -d
5	Проверить обновление через Swagger и логи

2.6.8. Если произошла ошибка «port is already allocated»

При выполнении команды:

```
SCRIPT

docker compose up -d --build
```

в терминале может появиться ошибка:

```
Error response from daemon: failed to set up container networking:
driver failed programming external connectivity on endpoint servicedeskapi-api-1 (...):
Bind for 0.0.0.0:8080 failed: port is already allocated
```

Что это значит:

- порт `8080` уже занят;
- либо старый контейнер всё ещё слушает этот порт;
- либо другой сервис (не Docker) использует порт 8080.

Шаг 1. Проверяем, какие контейнеры используют порт 8080

Выполните:

```
SCRIPT

docker ps
```

Найдите в выводе контейнеры, у которых в колонке PORTS указано что-то вроде:

```
0.0.0.0:8080->8080/tcp
```

Например:

```
CONTAINER ID   NAMES
...            servicedesk-api-container
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

```
...            servicedeskapi-api-1
```

Любой контейнер с пробросом `8080:8080` будет блокировать запуск нового.

Шаг 2. Останавливаем «лишние» контейнеры API

Остановите контейнеры, которые занимают порт 8080 (имена см. в выводе `docker ps`):

```
SCRIPT

docker stop servicedesk-api-container
docker stop servicedeskapi-api-1
```

Если какого-то контейнера не существует, Docker выведет сообщение об ошибке — это нормально.

Шаг 3. Удаляем контейнеры, чтобы освободить порт

После остановки удаляем контейнеры:

```
SCRIPT

docker rm servicedesk-api-container
docker rm servicedeskapi-api-1
```

Если контейнеров несколько, удалите каждый, кто слушает порт 8080.

Альтернативный вариант: остановить и удалить все контейнеры

> **Осторожно**
>
> Команды ниже остановят и удалят все контейнеры Docker, но не затронут volumes. Данные PostgreSQL в volume останутся.

```
SCRIPT

docker stop $(docker ps -aq)
docker rm $(docker ps -aq)
```

Используйте этот вариант, если в среде нет других важных контейнеров.

Шаг 4. Повторяем запуск docker-compose

После освобождения порта снова выполните:

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

```
SCRIPT

docker compose up -d --build
```

Ожидаемый результат:

- сервис API успешно поднимается;
- контейнер PostgreSQL продолжает работать;
- данные в базе не теряются, так как хранятся в `volume`.

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

2.7. Продемонстрировать работу с логами контейнеров

ОГЛАВЛЕНИЕ РАЗДЕЛА 2.7

- 2.7.0. Назначение блока
- 2.7.1. Проверка работающих контейнеров
- 2.7.2. Просмотр логов API
- 2.7.3. Просмотр логов в реальном времени
- 2.7.4. Логи PostgreSQL
- 2.7.5. Фильтрация логов
- 2.7.6. Получение последних N строк
- 2.7.7. Логи за определённый период
- 2.7.8. Методический итог

2.7.0. Назначение блока

Этот этап показывает студенту, как работать с логами контейнеров, что является важной частью сопровождения информационных систем.

Студент должен уметь:

- просматривать логи API;
- просматривать логи PostgreSQL;
- фильтровать вывод логов;
- анализировать логи в реальном времени;
- делать выводы на основе логов.

2.7.1. Проверка работающих контейнеров

Выполнить команду:

Помогу с учебным заданием без предоплаты. Оплата после выполнения

Нажмите на мессенджер, чтобы написать

Telegram ↗ WhatsApp ↗ MAX ↗

+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

```
SCRIPT

docker ps
```

Ожидаемый результат:

```
CONTAINER ID   IMAGE                    NAMES
abc123         servicedesk-api:2.0      servicedeskapi-api-1
def456         postgres:16-alpine       servicedesk-postgres
```

2.7.2. Просмотр логов API

Основная команда:

```
SCRIPT

docker logs servicedeskapi-api-1
```

(при необходимости имя контейнера можно уточнить через `docker ps`)

Пример вывода:

```
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://0.0.0.0:8080
>>> API VERSION 2: GetAll() called
info: Microsoft.EntityFrameworkCore.Update[30100]
      Database initialized using Npgsql
```

Студент должен увидеть свои сообщения (например: `API VERSION 2`), чтобы убедиться, что обновление контейнера прошло успешно.

2.7.3. Просмотр логов API в реальном времени

Аналог `tail -f` в Linux:

```
SCRIPT

docker logs -f servicedeskapi-api-1
```

Теперь студент открывает Swagger и отправляет запрос:

```
GET /api/equipment
SCRIPT
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Ожидаемый вывод в терминале:

```
>>> API VERSION 2: GetAll() called
```

Это доказывает, что логирование работает, а приложение корректно пишет в stdout.

2.7.4. Просмотр логов PostgreSQL

Команда:

```
SCRIPT

docker logs servicedesk-postgres
```

Пример вывода:

```
PostgreSQL init process complete; ready for start up.
database system is ready to accept connections
connection received: host=api user=servicedesk database=servicedeskdb
statement: SELECT "e"."Id", ...
```

Это показывает:

- запуск и инициализацию БД;
- входящие подключения от API;
- SQL-запросы (если включён verbose-режим);
- ошибки или проблемы.

2.7.5. Фильтрация логов по строке

PowerShell (Windows):

```
SCRIPT

docker logs servicedeskapi-api-1 | Select-String "GetAll"
```

Linux / macOS:

```
SCRIPT

docker logs servicedeskapi-api-1 | grep "GetAll"
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

2.7.6. Последние N строк логов

Например, последние 50 строк:

```
SCRIPT

docker logs --tail 50 servicedeskapi-api-1
```

2.7.7. Логи за определённый период времени

Логи за последние 30 минут:

```
SCRIPT

docker logs --since 30m servicedeskapi-api-1
```

Логи с определённого момента:

```
SCRIPT

docker logs --since "2025-11-23T20:00:00" servicedeskapi-api-1
```

2.7.8. Методический итог

Студент должен включить в отчёт:

1) Таблицу команд

Что делаем	Команда
Посмотреть все логи контейнера API	`docker logs servicedeskapi-api-1`
Просмотр логов в реальном времени	`docker logs -f servicedeskapi-api-1`
Последние 50 строк	`docker logs --tail 50 servicedeskapi-api-1`
Логи PostgreSQL	`docker logs servicedesk-postgres`
Фильтрация по ключевому слову	`docker logs servicedeskapi-api-1 | grep "GetAll"`
Логи с определённого времени	`docker logs --since "2025-11-23T20:00:00"`

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

2) Скриншоты:

- вывод `docker logs servicedeskapi-api-1`;
- вывод `docker logs -f` после запроса через Swagger;
- вывод `docker logs servicedesk-postgres`.

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

2.8. Реализовать и проверить работоспособность системы через health-check

ОГЛАВЛЕНИЕ РАЗДЕЛА 2.8

- 2.8.0. Цель блока
- 2.8.1. Health-check в самом приложении (API)
- 2.8.2. Проверка health-check локально
- 2.8.3. HEALTHCHECK в Dockerfile
- 2.8.4. HEALTHCHECK в docker-compose (опционально)
- 2.8.5. Пересборка и запуск контейнеров
- 2.8.6. Проверка и демонстрация для отчёта

2.8.0. Цель блока

В этом блоке настраивается проверка работоспособности системы через **health-check** так, чтобы:

- был простой endpoint в API для проверки состояния;
- Docker мог использовать его как `HEALTHCHECK`;
- студент мог протестировать работоспособность через браузер и Docker.

2.8.1. Health-check в самом приложении (API)

Сделаем отдельный контроллер, который:

- проверяет возможность подключения к базе данных;
- возвращает `200 OK`, если всё работает;
- возвращает `503 Service Unavailable`, если БД недоступна.

Создание контроллера `HealthController`

Файл: `Controllers/HealthController.cs`

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using ServiceDesk.Api.Data;

namespace ServiceDesk.Api.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class HealthController : ControllerBase
    {
        private readonly AppDbContext _context;

        public HealthController(AppDbContext context)
        {
            _context = context;
        }

        [HttpGet]
        public async Task<IActionResult> Get()
        {
            var canConnect = await _context.Database.CanConnectAsync();

            if (canConnect)
                return Ok(new { status = "Healthy", db = "Up", timestamp = DateTime.UtcNow });
```


Помогу с учебным заданием без предоплаты. Оплата после выполнения

Нажмите на мессенджер, чтобы написать

Telegram ↗ WhatsApp ↗ MAX ↗

+7 (924) 305-23-08 · Пишите в мессенджер

disynergy.ru

```
            return StatusCode(503, new { status = "Unhealthy", db = "Down" });
        }
    }
}
```

Дополнительных изменений в `Program.cs` не требуется: контроллер будет обнаружен автоматически.

2.8.2. Проверка health-check локально

1. Запустить API (через `docker compose up` или `dotnet run`).
2. Открыть в браузере:

```
http://localhost:8080/api/health
```
SCRIPT

Если всё работает, ожидается ответ с кодом `200 OK` и JSON:

```
{
  "status": "Healthy",
  "db": "Up",
  "timestamp": "2025-11-23T20:01:34.867Z"
}
```

Пример вывода логов при запуске через `docker compose up`:

```
PS C:\Users\gvadoskr\Desktop\ppSIS\ServiceDesk.Api> docker compose up
time="2025-11-23T23:01:34+03:00" level=warning msg="C:\\Users\\gvadoskr\\Desktop\\ppSIS\\ServiceDesk.Api\\docker-compose.yml:
[+] Running 2/2
 ✓ Container servicedesk-postgres  Created   0.0s
 ✓ Container servicedeskapi-api-1  Created   0.0s
Attaching to servicedesk-postgres, api-1
servicedesk-postgres  |
servicedesk-postgres  | PostgreSQL Database directory appears to contain a database; Skipping initialization
servicedesk-postgres  |
servicedesk-postgres  | 2025-11-23 20:01:34.833 UTC [1] LOG:  starting PostgreSQL 16.11 ...
servicedesk-postgres  | 2025-11-23 20:01:34.867 UTC [1] LOG:  database system is ready to accept connections
api-1                 | info: Microsoft.EntityFrameworkCore.Database.Command[20101]
api-1                 |       Executed DbCommand (20ms) [...]
api-1                 | info: Microsoft.Hosting.Lifetime[14]
api-1                 |       Now listening on: http://[::]:8080
api-1                 | info: Microsoft.Hosting.Lifetime[0]
api-1                 |       Application started. Press Ctrl+C to shut down.
```

Если база данных недоступна, endpoint вернёт `503 Service Unavailable` и статус `Unhealthy`.

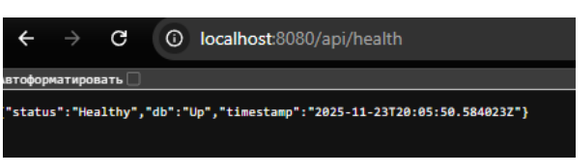

Рисунок 1 – Пример вызова `/api/health` в браузере

2.8.3. HEALTHCHECK в Dockerfile

Теперь настроим проверку состояния сервиса на уровне Docker. Для этого добавим директиву `HEALTHCHECK` в `Dockerfile`.

Открываем `Dockerfile` и добавляем в конец (перед `ENTRYPOINT` или сразу перед ним):

```
FROM base AS final
WORKDIR /app

COPY --from=build /app/publish .
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

```
# Health-check: Docker будет дергать этот URL внутри контейнера
HEALTHCHECK --interval=30s --timeout=5s --retries=3 \
  CMD wget -qO- http://localhost:8080/api/health || exit 1

ENTRYPOINT ["dotnet", "ServiceDesk.Api.dll"]
```

Здесь используется `wget`, который обычно доступен в базовом образе. Если в образе его нет, можно заменить на `curl -f` и при необходимости установить его в слое сборки.

2.8.4. HEALTHCHECK в docker-compose (опционально)

Дополнительно можно описать `healthcheck` в файле `docker-compose.yml` для сервиса API.

```
services:
  api:
    build: .
    image: servicedesk-api:2.0
    container_name: servicedesk-api
    ports:
      - "8080:8080"
    environment:
      - ASPNETCORE_ENVIRONMENT=Development
      - ASPNETCORE_URLS=http://+:8080
      - ConnectionStrings__DefaultConnection=Host=postgres;Port=5432;Database=servicedeskdb;Username=servicedesk;Password=serv
    depends_on:
      - postgres
    healthcheck:
      test: ["CMD", "wget", "-qO-", "http://localhost:8080/api/health"]
      interval: 30s
      timeout: 5s
      retries: 3
```

В этом случае Docker Compose будет учитывать состояние здоровья сервиса API при управлении контейнерами.

2.8.5. Пересборка и запуск контейнеров

После внесения изменений в `Dockerfile` и `docker-compose.yml` необходимо пересобрать и перезапустить проект:

SCRIPT

```
docker compose down
```

SCRIPT

```
docker compose up -d --build
```

Проверка статуса контейнера:

SCRIPT

```
docker ps
```

Через несколько секунд в колонке **STATUS** для контейнера API должно появиться что-то вроде:

```
Up 20 seconds (healthy)
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

```
PS C:\Users\gvadoskr\Desktop\ppSIS\ServiceDesk.Api> docker compose down
time="2025-11-23T23:04:26+03:00" level=warning msg="C:\\Users\\gvadoskr\\Desktop\\ppSIS\\ServiceDesk.Api\\docker-
[+] Running 3/3
 ✔Container servicedeskapi-api-1  Removed
 ✔Container servicedesk-postgres  Removed
 ✔Network servicedeskapi_default  Removed
PS C:\Users\gvadoskr\Desktop\ppSIS\ServiceDesk.Api>
PS C:\Users\gvadoskr\Desktop\ppSIS\ServiceDesk.Api> docker compose up -d --build
time="2025-11-23T23:04:31+03:00" level=warning msg="C:\\Users\\gvadoskr\\Desktop\\ppSIS\\ServiceDesk.Api\\docker-
Compose can now delegate builds to bake for better performance.
 To do so, set COMPOSE_BAKE=true.
[+] Building 5.0s (17/17) FINISHED
 => [api internal] load build definition from Dockerfile
 => => transferring dockerfile: 1.42kB
 => [api internal] load metadata for mcr.microsoft.com/dotnet/sdk:9.0
 => [api internal] load metadata for mcr.microsoft.com/dotnet/aspnet:9.0
 => [api internal] load .dockerignore
 => => transferring context: 149B
 => [api build 1/6] FROM mcr.microsoft.com/dotnet/sdk:9.0@sha256:584cdb686f8ff7d2ffd636581b9bba8c9de38df1e751e927
 => => resolve mcr.microsoft.com/dotnet/sdk:9.0@sha256:584cdb686f8ff7d2ffd636581b9bba8c9de38df1e751e9275438cddecd
 => [api internal] load build context
 => => transferring context: 2.27kB
 => [api base 1/2] FROM mcr.microsoft.com/dotnet/aspnet:9.0@sha256:925a86ec775fe737f1d65371ad7b2e01e81390c3c327ce
 => => resolve mcr.microsoft.com/dotnet/aspnet:9.0@sha256:925a86ec775fe737f1d65371ad7b2e01e81390c3c327ce3c06866a8
 => CACHED [api build 2/6] WORKDIR /src
 => CACHED [api build 3/6] COPY [ServiceDesk.Api.csproj, ./]
 => CACHED [api build 4/6] RUN dotnet restore "ServiceDesk.Api.csproj"
 => [api build 5/6] COPY . .
 => [api build 6/6] RUN dotnet publish "ServiceDesk.Api.csproj" -c Release -o /app/publish /p:UseAppHost=false
 => CACHED [api base 2/2] WORKDIR /app
 => CACHED [api final 1/2] WORKDIR /app
 => [api final 2/2] COPY --from=build /app/publish .
 => [api] exporting to image
 => => exporting layers
 => => exporting manifest sha256:04dbe9e816089fb5669dc7f71c0fa2efbe07ebc9a3358450404cb2731fd3c78e
 => => exporting config sha256:1034cc996f7989bb9216abddb9414349697bbcb58e2eae5013b95a9dba45b981
 => => exporting attestation manifest sha256:be5690394e95612a5421cdedb3f7afb9099d9731d67af091524b7fe0d24ab399
 => => exporting manifest list sha256:1e4f3e46555d954d11dd6af00720e75c373ecaa4def1bf186f1503f0ea91727a
 => => naming to docker.io/library/servicedesk-api:2.0
 => => unpacking to docker.io/library/servicedesk-api:2.0
 => [api] resolving provenance for metadata file
[+] Running 4/4
 ✔api                           Built
 ✔Network servicedeskapi_default  Created
 ✔Container servicedesk-postgres  Started
 ✔Container servicedeskapi-api-1  Started
PS C:\Users\gvadoskr\Desktop\ppSIS\ServiceDesk.Api> docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED         STATUS                            P
124b0ed3f3a1   servicedesk-api:2.0    "dotnet ServiceDesk.…"   8 seconds ago   Up 7 seconds (health: starting)   0
14248a8b1f47   postgres:16-alpine     "docker-entrypoint.s…"   8 seconds ago   Up 7 seconds                      5
PS C:\Users\gvadoskr\Desktop\ppSIS\ServiceDesk.Api>
```

Рисунок 2 – Пример вывода `docker ps` с пометкой `(healthy)`

2.8.6. Проверка и демонстрация для отчёта

В отчёте студент должен продемонстрировать работу health-check:

1. Скриншот или пример вызова:
   - `GET http://localhost:8080/api/health` с кодом ответа `200 OK` и JSON-статусом.
2. Скриншот вывода `docker ps` с пометкой `(healthy)` у контейнера API.
3. Пример команды для детального просмотра состояния health-check:

```
SCRIPT

docker inspect servicedesk-api --format='{{json .State.Health}}'
```

   или (в PowerShell):

```
SCRIPT

docker inspect servicedesk-api | Select-String "Health"
```

4. Краткое объяснение:
   - что такое health-check;

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

- какую проверку выполняет `/api/health` (подключение к PostgreSQL);
- как Docker использует `HEALTHCHECK` для мониторинга контейнера.

Помогу с учебным заданием без предоплаты. Оплата после выполнения

Нажмите на мессенджер, чтобы написать

Telegram ↗ WhatsApp ↗ MAX ↗

+7 (924) 305-23-08 · Пишите в мессенджер

disynergy.ru

2.9. Выполнить базовые процедуры резервного копирования и восстановления данных (PostgreSQL)

ОГЛАВЛЕНИЕ РАЗДЕЛА 2.9

- 2.9.0. Цель и виды резервного копирования
- 2.9.1. Резервное копирование средствами PostgreSQL (SQL-дамп)
- 2.9.2. Восстановление базы данных из дампа
- 2.9.3. Полное физическое резервное копирование через Docker volume
- 2.9.4. Восстановление Docker volume
- 2.9.5. Таблица команд для отчёта
- 2.9.6. Методическое описание для отчёта

2.9.0. Цель и виды резервного копирования

Так как система развернута в Docker и база PostgreSQL работает в контейнере с подключённым volume, можно использовать два подхода к резервному копированию:

1. **Логический backup (SQL-дамп)** — с помощью `pg_dump`.
2. **Физический backup** — копия Docker volume с файлами базы.

Для учебной практики достаточно освоить:

- создание SQL-дампа и восстановление из него;
- базовые операции с Docker volume как альтернативный способ.

2.9.1. Резервное копирование средствами PostgreSQL (SQL-дамп)

Это основной, переносимый и рекомендуемый способ резервного копирования.

Помогу с учебным заданием без предоплаты. Оплата после выполнения

Нажмите на мессенджер, чтобы написать

Telegram ↗ WhatsApp ↗ MAX ↗

+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru

Шаг 1. Узнать имя контейнера PostgreSQL

Выполнить команду:

```
SCRIPT

docker ps
```

Имя контейнера базы данных, например:

```
servicedesk-postgres
```

Шаг 2. Создать SQL-дамп

Команда для создания резервной копии:

```
SCRIPT

docker exec -t servicedesk-postgres pg_dump -U servicedesk servicedeskdb > backup.sql
```

Расшифровка:

- `servicedesk-postgres` — имя контейнера PostgreSQL;
- `pg_dump` — встроенная утилита резервного копирования PostgreSQL;
- `-U servicedesk` — пользователь БД;
- `servicedeskdb` — имя базы данных;
- `> backup.sql` — перенаправление вывода в файл `backup.sql` на хосте.

После выполнения в текущей папке появится файл:

```
backup.sql
```

Это полный логический дамп базы данных.

2.9.2. Восстановление базы данных из дампа

Перед восстановлением необходимо:

- остановить сервис API (чтобы не было активных подключений к базе);
- очистить или пересоздать базу данных.

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Шаг 1. Удалить старую базу и создать новую

```
SCRIPT

docker exec -it servicedesk-postgres psql -U servicedesk -c "DROP DATABASE
servicedeskdb;"
```

```
SCRIPT

docker exec -it servicedesk-postgres psql -U servicedesk -c "CREATE DATABASE
servicedeskdb;"
```

Шаг 2. Восстановить базу из дампа

```
SCRIPT

docker exec -i servicedesk-postgres psql -U servicedesk servicedeskdb < backup.sql
```

Если в дампе есть таблицы и данные, в выводе появятся сообщения вида:

```
CREATE TABLE
INSERT 0 1
INSERT 0 1
...
```

2.9.3. Полное физическое резервное копирование через Docker volume

Этот способ сохраняет физическое содержимое volume с файлами базы данных.

Он полезен, когда:

- нужно показать альтернативный способ резервирования;
- база данных очень большая;
- важно сохранить всё, включая служебные данные PostgreSQL.

Шаг 1. Узнать имя volume

```
SCRIPT

docker volume ls
```

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

Пример результата:

```
DRIVER    VOLUME NAME
local     servicedesk-postgres
```

Шаг 2. Создать архив volume

Создадим архив с содержимым volume в текущей папке:

```
SCRIPT

docker run --rm \
  -v servicedesk-postgres:/db \
  -v $(pwd):/backup \
  alpine tar czvf /backup/db-volume.tar.gz /db
```

После выполнения в текущем каталоге появится файл:

```
db-volume.tar.gz
```

Это полная физическая копия содержимого volume с базой данных.

2.9.4. Восстановление Docker volume

Шаг 1. Остановить контейнеры

```
SCRIPT

docker compose down
```

Шаг 2. Удалить существующий volume

```
SCRIPT

docker volume rm servicedesk-postgres
```

Шаг 3. Восстановить volume из архива

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

```
SCRIPT

docker run --rm \
  -v servicedesk-postgres:/db \
  -v $(pwd):/backup \
  alpine tar xzvf /backup/db-volume.tar.gz -C /
```

Шаг 4. Запустить docker-compose

```
SCRIPT

docker compose up -d
```

После этого контейнер PostgreSQL будет использовать восстановленный volume с данными, а API сможет продолжать работу с уже восстановленной базой.

2.9.5. Таблица команд для отчёта

Для отчёта удобно оформить основные операции в виде таблицы.

Операция	Команда
Создать SQL-дамп	`docker exec -t servicedesk-postgres pg_dump -U servicedesk servicedeskdb > backup.sql`
Восстановить SQL-дамп	`docker exec -i servicedesk-postgres psql -U servicedesk servicedeskdb < backup.sql`
Удалить базу данных	`DROP DATABASE servicedeskdb;`
Создать базу данных	`CREATE DATABASE servicedeskdb;`
Посмотреть список volumes	`docker volume ls`
Создать архив volume	`docker run --rm -v servicedesk-postgres:/db -v $(pwd):/backup alpine tar czvf /backup/db-volume.tar.gz /db`
Восстановить volume из архива	`docker run --rm -v servicedesk-postgres:/db -v $(pwd):/backup alpine tar xzvf /backup/db-volume.tar.gz -C /`

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер
disynergy.ru

2.9.6. Методическое описание для отчёта

В отчёте студент должен:

1. Выполнить логическое резервное копирование базы данных (SQL-дамп) и зафиксировать команду и результат (наличие файла `backup.sql`).
2. Выполнить восстановление базы данных из дампа и показать, что данные появились в системе после восстановления.
3. Продемонстрировать физическое копирование volume (создание и восстановление `db-volume.tar.gz`) как альтернативный метод.
4. Показать, что данные сохраняются после перезапуска контейнеров и/или обновления версии приложения.
5. Добавить скриншоты выполнения ключевых команд:
   - создание `backup.sql`;
   - восстановление из дампа;
   - создание архива `db-volume.tar.gz`;
   - успешный запуск `docker compose up -d` после восстановления.

Помогу с учебным заданием без предоплаты. Оплата после выполнения
Нажмите на мессенджер, чтобы написать
Telegram ↗ WhatsApp ↗ MAX ↗
+7 (924) 305-23-08 · Пишите в мессенджер disynergy.ru