Оглавление

[Создание проекта Web API на основе контроллера 2](#_Toc169203837)

[Просмотр составных частей решения 2](#_Toc169203838)

[Запуск проекта 5](#_Toc169203839)

[Создание и подключение к web API js 7](#_Toc169203840)

[Создание js 7](#_Toc169203841)

[Подключение к серверу 8](#_Toc169203842)

[Подключение БД к Web API 11](#_Toc169203843)

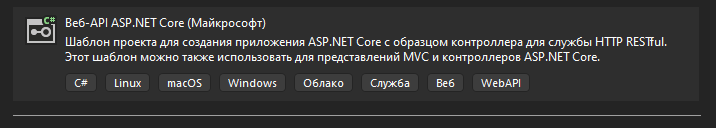
[Создание базы данных 11](#_Toc169203844)

[Подключение к БД 13](#_Toc169203845)

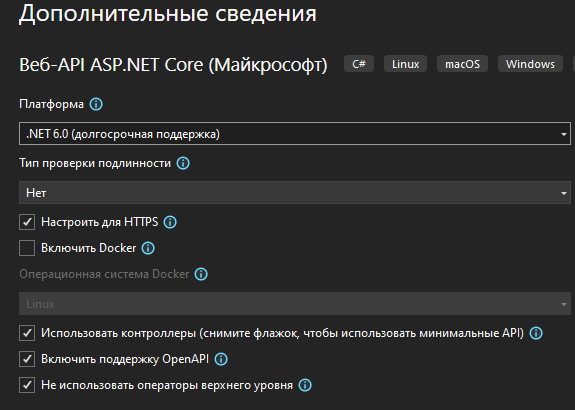
# Создание проекта Web API на основе контроллера

## Просмотр составных частей решения

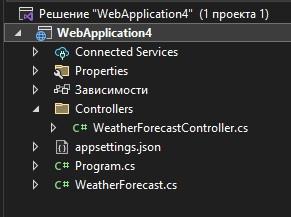
1. Выбираем проект в visual studio



1. Выбираем нужные параметры и создаем проект



В созданном проекте имеются элементы:



Папка контроллеры содержит контроллеры для управления логикой обработки запросов, они помогают разделить код на логические блоки и обеспечить модульность при разработке API.

Контроллер WeatherForecastController.cs:

using Microsoft.AspNetCore.Mvc;

namespace WebApplication4.Controllers

{

[ApiController]

[Route("[controller]")]

public class WeatherForecastController : ControllerBase

{

private static readonly string[] Summaries = new[]

{

"Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching"

};

private readonly ILogger<WeatherForecastController> \_logger;

public WeatherForecastController(ILogger<WeatherForecastController> logger)

{

\_logger = logger;

}

[HttpGet(Name = "GetWeatherForecast")]

public IEnumerable<WeatherForecast> Get()

{

return Enumerable.Range(1, 5).Select(index => new WeatherForecast

{

Date = DateTime.Now.AddDays(index),

TemperatureC = Random.Shared.Next(-20, 55),

Summary = Summaries[Random.Shared.Next(Summaries.Length)]

})

.ToArray();

}

}

}

appsettings.json в ASP.NET Core отвечает за хранение конфигурационных данных приложения. В этом файле обычно содержатся параметры настройки, такие как строки подключения к базе данных, ключи API, параметры логирования, и другие настройки, которые могут быть изменены без необходимости перекомпиляции приложения.

appsettings.json:

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*"

}

Program.cs служит входной точкой выполнения приложения.

Program.cs:

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();

// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();

WeatherForecast.cs используется в качестве модели данных для хранения информации о прогнозе погоды.

WeatherForecast.cs:

namespace WebApplication4

{

public class WeatherForecast

{

public DateTime Date { get; set; }

public int TemperatureC { get; set; }

public int TemperatureF => 32 + (int)(TemperatureC / 0.5556);

public string? Summary { get; set; }

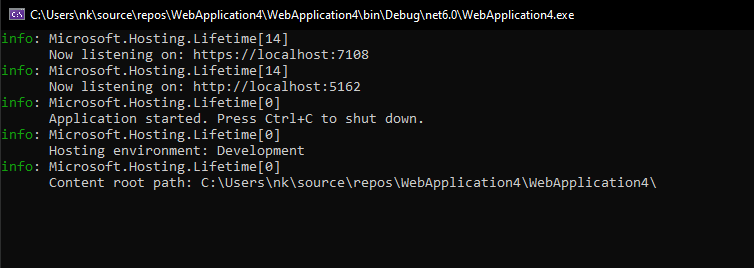
}

}

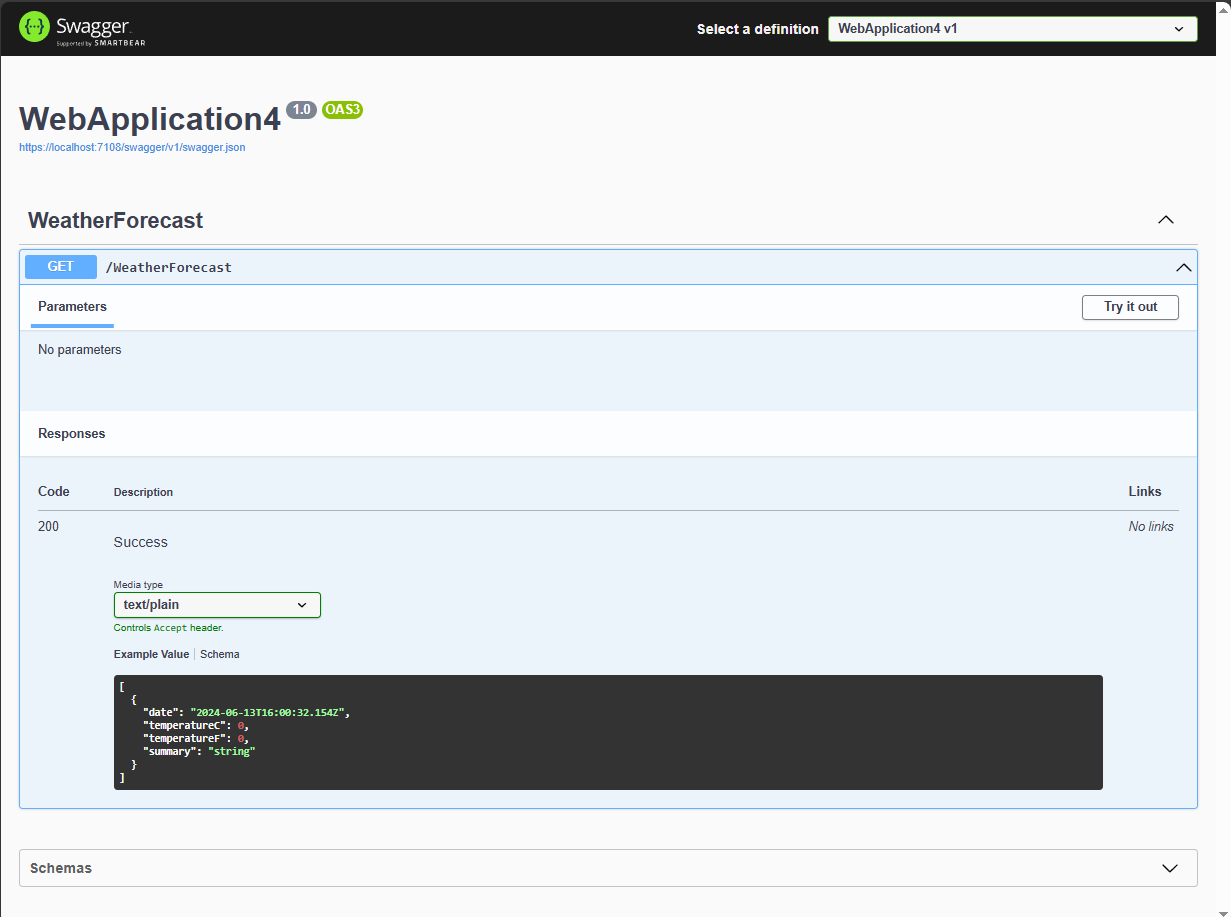
## Запуск проекта

При запуске проекта открываются:

1. Консольное окно

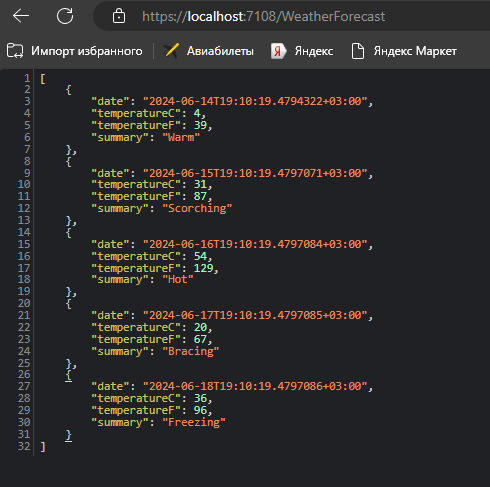


1. Swagger — инструмент для подготовки документации к API и проведения тестов API.



Добавим к https://localhost:7108 запросу, название контроллера WeatherForecast:

Получим случайные данные сгенерированные контроллером



# Создание и подключение к web API js

## Создание js

Для создания JavaScript проекта нужно:

1. Создаем папку на рабочем столе
2. Открываем ее с помощью VS Code
3. В VS Code создаем app.js ­– файл который содержит JavaScript-код для добавления интерактивности и функциональности веб-приложению, index.html – файл является стартовой точкой веб-приложения. Он содержит структуру веб-страницы, такие как разметка HTML, ссылки на внешние ресурсы и другие элементы, main.css – файл обычно содержит каскадные таблицы стилей, которые определяют внешний вид веб-приложения.
4. html файл:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <script src="app.js"></script>

    <title>Document</title>

</head>

<body>

    <button onclick="qwerty()">get</button>

</body>

</html>

1. js файл:

*// Объявление асинхронной функции logMovies и присвоение её переменной qwerty*

const qwerty = async function logMovies() {

*// Выполняется запрос fetch к URL "https://localhost:7108/WeatherForecast"*

    const response = await fetch("https://localhost:7108/WeatherForecast")

*// Сохранение ответа от сервера в переменную movies*

    const movies = await response;

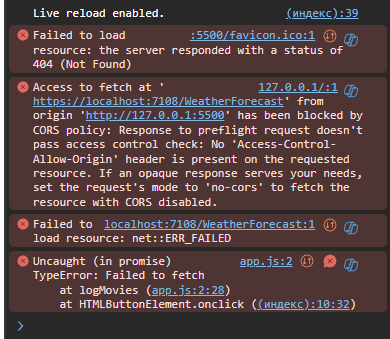
*// Извлечение JSON данных из ответа и вывод их в консоль*

    console.log(await movies.json());

}

## Подключение к серверу

По нажатию на кнопку get на странице, после запуска Web API и js происходит ошибка:



Для решения этой ошибки требуется разрешение CORS

Обновим код js:

const qwerty = async function logMovies() {

*// Выполняется запрос fetch к URL "https://localhost:7108/WeatherForecast"*

    const response = await fetch("https://localhost:7108/WeatherForecast", {

*// Установка режима запроса 'cors' для работы с CORS*

        mode: 'cors',

        headers: {

*// Установка заголовка 'Access-Control-Allow-Origin' со значением '\*', что не имеет смысла, так как это должно быть настроено на сервере*

            'Access-Control-Allow-Origin': '\*',

        }

    });

*// Сохранение ответа от сервера в переменную movies*

    const movies = await response;

*// Извлечение JSON данных из ответа и вывод их в консоль*

    console.log(await movies.json());

}

И обновим код Program:

namespace WebApplication3

{

public class Program

{

public static void Main(string[] args)

{

var builder = WebApplication.CreateBuilder(args); //Создает экземпляр построителя веб-приложения.

builder.Configuration.AddJsonFile("appsettings.json"); // эта строка для загрузки конфигурации из файла appsettings.json

builder.Services.AddControllers(); //Регистрирует контроллеры MVC как сервисы.

builder.Services.AddEndpointsApiExplorer(); // Добавляет службы исследователя API для обнаружения конечных точек.

builder.Services.AddSwaggerGen(); //Добавляет службы Swagger для генерации документации API.

builder.Services.AddCors(p => p.AddPolicy("corsapp", builder =>

{

builder.WithOrigins("\*").AllowAnyMethod().AllowAnyHeader();

})); //Добавляет службы Cross-Origin Resource Sharing (CORS) для разрешения кросс-доменных запросов.

var app = builder.Build();

if (app.Environment.IsDevelopment()) //Проверяет, работает ли приложение в среде разработки.

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection(); //Перенаправляет запросы HTTP на HTTPS.

app.UseRouting();

app.UseCors("corsapp"); // Настраивает политику CORS для приложения.

app.UseAuthorization();

app.MapControllers(); //Отображает контроллеры для обработки входящих HTTP-запросов.

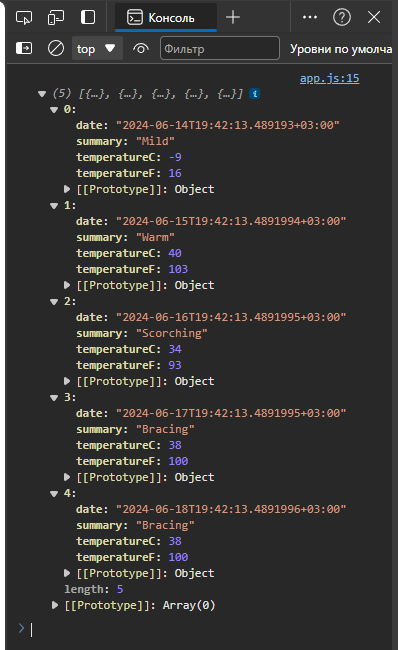
app.Run();

}

}

}

Теперь по нажатию на get в консоли будет:



# Подключение БД к Web API

## Создание базы данных

Создадим запрос на создание БД в SQL:  
create table [Raspesanie].dbo.Kafedra

(

id\_kafedra INT IDENTITY(1,1) PRIMARY KEY,

Name nvarchar(100),

Сode nvarchar(100),

Zaveduchi nvarchar(100)

);

create table [Raspesanie].dbo.Groups

(

id\_group INT IDENTITY(1,1) PRIMARY KEY,

id\_kafedra INT,

Name nvarchar(100),

Сode nvarchar(100),

Rykovod nvarchar(100),

FOREIGN KEY (id\_kafedra) REFERENCES [Raspesanie].dbo.Kafedra(id\_kafedra)

);

create table [Raspesanie].dbo.Users

(

id\_user INT IDENTITY(1,1) PRIMARY KEY,

Login nvarchar(100),

Pass nvarchar(100),

Familia nvarchar(100),

Name nvarchar(100),

Otchestvo nvarchar(100),

Post nvarchar(100)

);

create table [Raspesanie].dbo.Prepods

(

id\_teach INT IDENTITY(1,1) PRIMARY KEY,

id\_kafedra INT,

Familia nvarchar(100),

Name nvarchar(100),

Otchestvo nvarchar(100),

FOREIGN KEY (id\_kafedra) REFERENCES [Raspesanie].dbo.Kafedra(id\_kafedra)

);

create table[Raspesanie].dbo. Discipline

(

id\_discipline INT IDENTITY(1,1) PRIMARY KEY,

id\_kafedra INT,

id\_teach INT,

Name nvarchar(100),

Сode nvarchar(100),

number\_hours INT,

FOREIGN KEY (id\_kafedra) REFERENCES [Raspesanie].dbo.Kafedra(id\_kafedra),

FOREIGN KEY (id\_teach) REFERENCES [Raspesanie].dbo.Prepods(id\_teach)

);

create table [Raspesanie].dbo.Kabinet

(

id\_office INT IDENTITY(1,1) PRIMARY KEY,

Number INT,

Korpus nvarchar(100)

);

create table [Raspesanie].dbo.Raspes

(

id\_raspes INT IDENTITY(1,1) PRIMARY KEY,

id\_discipline INT,

id\_teach INT,

id\_group INT,

id\_office INT,

id\_user INT,

DayNedel nvarchar(100),

hours\_passed INT,

FOREIGN KEY (id\_discipline) REFERENCES [Raspesanie].dbo.Discipline(id\_discipline),

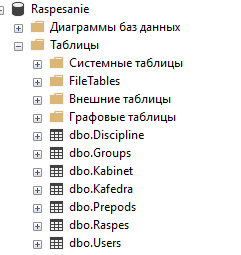
FOREIGN KEY (id\_teach) REFERENCES [Raspesanie].dbo.Prepods(id\_teach),

FOREIGN KEY (id\_group) REFERENCES [Raspesanie].dbo.Groups(id\_group),

FOREIGN KEY (id\_office) REFERENCES [Raspesanie].dbo.Kabinet(id\_office),

FOREIGN KEY (id\_user) REFERENCES [Raspesanie].dbo.Users(id\_user)

);



## Подключение к БД

Для подключения к базе данных изменим appsettings.json:  
{

"ConnectionStrings": {

"DefaultConnection": "Server=localhost;Database=Raspesanie;Trusted\_Connection=True;Encrypt=True;TrustServerCertificate=True;"

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*"

}

Trusted\_Connection=True: Указывает на использование Windows-аутентификации для аутентификации в SQL Server. Если задано значение True, то при подключении будет использоваться сведения об учетной записи Windows текущего пользователя для аутентификации.

Encrypt=True: Указывает, что подключение к серверу SQL Server должно использовать шифрование для обеспечения безопасности передаваемых данных между клиентом и сервером.

TrustServerCertificate=True: Указывает, что клиент доверяет сертификату сервера SQL Server при установлении защищенного соединения. Включение этого параметра позволяет клиенту принимать сертификат сервера как доверенный без проверки подлинности.

Создадим в решении класс Person в котором будут столбцы из бд:

namespace WebApplication3

{

public class Person

{

public int id\_raspes { get; set; }

public int id\_discipline { get; set; }

public int id\_teach { get; set; }

public int id\_group { get; set; }

public int id\_office { get; set; }

public int id\_user { get; set; }

public string DayNedel { get; set; }

public int hours\_passed { get; set; }

}

}

Изменим WeatherForecastController.cs:

using Microsoft.AspNetCore.Mvc;

using Microsoft.Data.SqlClient;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.Logging;

using System;

namespace WebApplication3.Controllers

{

[ApiController]

[Route("[controller]")]

public class WeatherForecastController : ControllerBase

{

private readonly IConfiguration \_configuration;

private readonly ILogger<WeatherForecastController> \_logger;

public WeatherForecastController(IConfiguration configuration, ILogger<WeatherForecastController> logger)

{

\_configuration = configuration; // Инициализация поля конфигурации

\_logger = logger; // Инициализация поля журналирования

}

[HttpGet("GetPersons", Name = "GetPersons")]

public IActionResult GetPersons()

{

string connectionString = \_configuration.GetConnectionString("DefaultConnection"); // Получение строки подключения к базе данных из конфигурации

try

{

using (var connection = new SqlConnection(connectionString))

{

connection.Open();

var commandText = "SELECT id\_raspes, id\_discipline, id\_teach, id\_group, id\_office, id\_user, DayNedel, hours\_passed FROM Raspes";

var command = new SqlCommand(commandText, connection);

var reader = command.ExecuteReader();

var persons = new List<Person>();

while (reader.Read())

{

var person = new Person

{

id\_raspes = reader.GetInt32(0),

id\_discipline = reader.GetInt32(1),

id\_teach = reader.GetInt32(2),

id\_group = reader.GetInt32(3),

id\_office = reader.GetInt32(4),

id\_user = reader.GetInt32(5),

DayNedel = reader.GetString(6),

hours\_passed = reader.GetInt32(7)

};

persons.Add(person); // Добавление объекта Person в список

}

return Ok(persons); // Возврат списка объектов Person в формате JSON

}

}

}

Изменим js:

const displayModal = (data) => {

    const modal = document.getElementById("myModal");

    const modalContent = modal.querySelector(".form-group");

    document.getElementById("getBtn").addEventListener("click", qwerty);

    document.querySelector(".close").addEventListener("click", closeModal);

    modalContent.innerHTML = "";

    data.forEach(item => {

        const dataElement = document.createElement("div");

        dataElement.innerHTML = `

            <p>id\_raspes: ${item.id\_raspes}</p>

            <p>id\_discipline: ${item.id\_discipline}</p>

            <p>id\_teach: ${item.id\_teach}</p>

            <p>id\_group: ${item.id\_group}</p>

            <p>id\_office: ${item.id\_office}</p>

            <p>id\_user: ${item.id\_user}</p>

            <p>DayNedel: ${item.DayNedel}</p>

            <p>hours\_passed: ${item.hours\_passed}</p>

            <hr>

        `;

        modalContent.appendChild(dataElement);

    });

    modal.style.display = "block"; *// Показываем модальное окно*

};

const closeModal = () => {

    const modal = document.getElementById("myModal");

    modal.style.display = "none";

};

*// Функция для загрузки данных по клику на кнопку*

const qwerty = async function () {

    try {

        const response = await fetch("https://localhost:7108/WeatherForecast/GetPersons");

        const weatherData = await response.json();

        displayModal(weatherData);

    } catch (error) {

        console.error("Error fetching data:", error);

    }

};

Все запускаем и нажимаем get:

