**Criar uma aplicação ->**  dotnet new webapi -n Proagil

**Rodando um projeto ->** dotnet run

Dotnet watch run -> **estarta o projeto deixado livre a modificação**

A utilização do controller só é possível, pois adicionamos no services o MVC no startup

**Cirando a classe dataContext**

Cria uma nova pasta chamada de data, dentro dela coloca uma classe c# chamada de dataContext, vale lembrar que ela herda de de DbContext

Ele ficará assim:

public class DataContext : DbContext

    {

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

    }

Já no startup podemos configurar o servisse

services.AddDbContext<DataContext>(x => x.UseSqlite(Configuration.GetConnectionString("DefaultConnection")));

Vale lembrar que precisa baixar o Bd que você vai usar, no meu Caso:

Microsoft.EntityFrameworkCore.Sqlite, para entrar no nuget aperta crtl+shift+p

Esse defaultCOnnection vai ser configurada no appsetings

**Para criar uma nova migration**

dotnet ef migrations add initial

Quando se trabalha com IActionResult deve-se enviar as respostas com OK e circular com tryCatch

[HttpGet]

        public IActionResult Get()

        {

            try

            {

                var result = \_context.Eventos.ToList();

                return Ok(result);

            }

            catch (System.Exception)

            {

                return this.StatusCode(StatusCodes.Status500InternalServerError, "BD falhou");

            }

        }

**Chamadas assíncronas**

[HttpGet]

        public async Task<IActionResult> Get()

        {

            try

            {

                var result = await \_context.Eventos.ToListAsync();

                return Ok(result);

            }

            catch (System.Exception)

            {

                return this.StatusCode(StatusCodes.Status500InternalServerError, "BD falhou");

            }

        }

A cada chamada é aberta uma nova thread e o await faz esperar, pois se não trava o recurso.