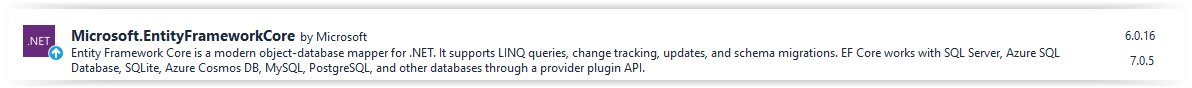
**AWS LAMBDA API GATEWAY PERSONAJES**

Lo primero que haremos será crear un Api Gateway para consumir los datos de una base de datos RDS.

Creamos un nuevo proyecto de tipo **AWS Serverless Application**

Instalamos los siguiente **Nuget**



Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Sobre **Models** creamos una nueva clase llamada **Personaje**

**PERSONAJE**

Sobre el proyecto, creamos una carpeta llamada **Data** y una clase llamada **PersonajesContext**

**PERSONAJESCONTEXT**

public class PersonajesContext : DbContext

{

public PersonajesContext(DbContextOptions<PersonajesContext> options)

: base(options) { }

public DbSet<Personaje> Personajes { get; set; }

}

Sobre el proyecto, creamos una carpeta llamada **Repositories** y una clase llamada **RepositoryPersonajes**

**REPOSITORYPERSONAJES**

public class RepositoryPersonajes

{

private PersonajesContext context;

public RepositoryPersonajes(PersonajesContext context)

{

this.context = context;

}

public async Task<List<Personaje>> GetPersonajesAsync()

{

return await this.context.Personajes.ToListAsync();

}

public async Task<Personaje> FindPersonajeAsync(int id)

{

return await

this.context.Personajes

.FirstOrDefaultAsync(x => x.IdPersonaje == id);

}

private int GetMaxIdPersonaje()

{

return this.context.Personajes.Max(z => z.IdPersonaje) + 1;

}

public async Task CreatePersonaje(string nombre

, string imagen, string descripcion)

{

Personaje personaje = new Personaje();

personaje.IdPersonaje = this.GetMaxIdPersonaje();

personaje.Nombre = nombre;

personaje.Imagen = imagen;

personaje.Descripcion = descripcion;

this.context.Personajes.Add(personaje);

await this.context.SaveChangesAsync();

}

}

Incluimos la cadena de conexión a RDS dentro de **appsettings.json**

**APPSETTINGS.JSON**

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

Sobre **Controllers**, creamos un nuevo controlador de tipo **Api Empty** llamado **PersonajesController**

**PERSONAJESCONTROLLER**

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

[ApiController]

public class PersonajesController : ControllerBase

{

private RepositoryPersonajes repo;

public PersonajesController(RepositoryPersonajes repo)

{

this.repo = repo;

}

[HttpGet]

public async Task<ActionResult<List<Personaje>>> GetPersonajes()

{

return await this.repo.GetPersonajesAsync();

}

[HttpGet("{id}")]

public async Task<ActionResult<Personaje>> FindPersonaje(int id)

{

return await this.repo.FindPersonajeAsync(id);

}

[HttpPost]

public async Task<ActionResult> CreatePersonaje(Personaje personaje)

{

await this.repo.CreatePersonaje(personaje.Nombre

, personaje.Imagen, personaje.Descripcion);

return Ok();

}

}

Resolvemos la inyección de dependencias y agregamos Swagger para la documentación dentro de **Startup**

**STARTUP**

public void ConfigureServices(IServiceCollection services)

{

string connectionString =

this.Configuration.GetConnectionString("MySql");

services.AddTransient<RepositoryPersonajes>();

services.AddDbContext<PersonajesContext>

(options => options.UseMySql(connectionString

, ServerVersion.AutoDetect(connectionString)));

services.AddSwaggerGen(options =>

{

options.SwaggerDoc("v1", new OpenApiInfo

{

Title = "Api Gateway Personajes AWS",

Version = "v1"

});

});

services.AddCors(p => p.AddPolicy("corsenabled", builder =>

{

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

}));

services.AddControllers();

}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseCors("corsenabled");

app.UseSwagger();

app.UseSwaggerUI(options =>

{

options.SwaggerEndpoint

("swagger/v1/swagger.json", "Api Gateway Personajes AWS v1");

options.RoutePrefix = "";

});

app.UseHttpsRedirection();

app.UseRouting();

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

endpoints.MapGet("/", async context =>

{

await context.Response.WriteAsync("Welcome to running ASP.NET Core on AWS Lambda");

});

});

}

Una vez que lo tenemos montado, probamos la funcionalidad en LOCAL

Interfaz de usuario gráfica, Aplicación, Teams

Descripción generada automáticamente

A continuación, vamos a publicar nuestro Api Gateway.

Para ello utilizamos un usuario de IAM y un Bucket para que la plantilla de **CloudFormation** pueda desplegar el proyecto

Abrimos IAM y creamos un usuario con permisos de administración

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Abrimos S3 y creamos un bucket (con nombre único) para poder incluir nuestros Artifacts. Simplemente lo creamos, no tenemos que dar ningún tipo de permisos.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Sobre **Visual Studio** incluimos nuestras nuevas claves de administrador que hemos descargado.

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Y publicamos con el menú contextual sobre el proyecto.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Esperamos a su publicación y probamos los métodos en Producción.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente