**ASP.NET Core Web API**

1. Crear el proyecto con la siguiente configuración.

Captura de pantalla de un celular

Descripción generada automáticamente

1. Eliminar los archivos relacionados a los controladores creados por defecto.
2. En la carpeta controller agregar nuevo controlador. Escogemos controlador de API en blanco. Lo llamaremos VillaController.cs
3. Crear la carpeta modelos y agregar las propiedades. También importante crear los DTO porque hay cierta información que no se va a mostrar en el controlador.
4. Para simular la base de datos vamos a crear una carpeta datos. Donde crearemos VillaStore.

namespace MagicVilla\_API.Datos

{

public static class VillaStore

{

public static List<VillaDto> villaList = new List<VillaDto>

{

new VillaDto{Id=1, Nombre="Vista a la Piscina", Ocupantes=3, MetrosCuadrados=50},

new VillaDto{Id=2, Nombre="Vista a la Playa", Ocupantes=4, MetrosCuadrados=80}

};

}

}

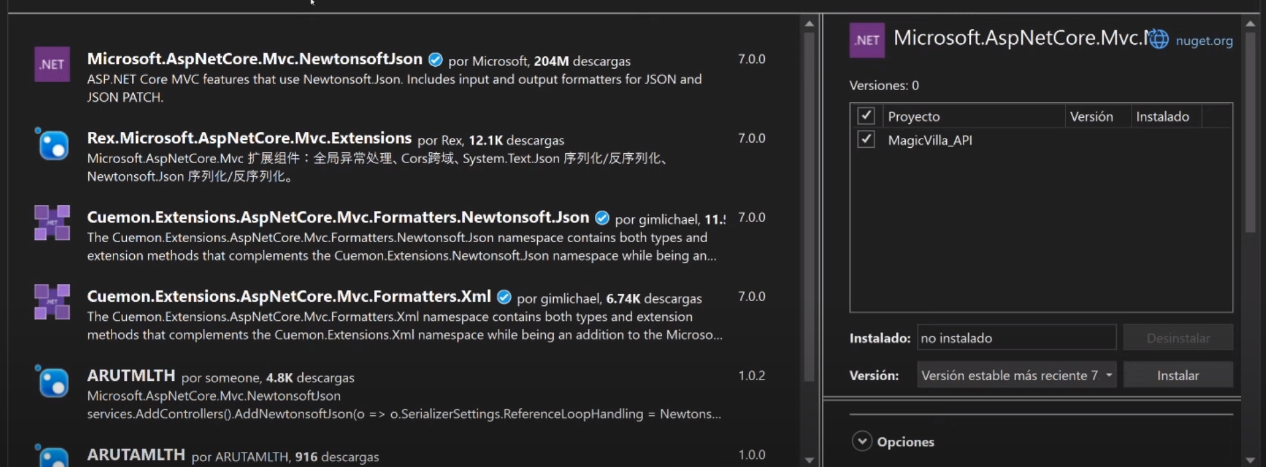
1. Agregar ActionResult para personalizar el status code de las respuestas.
2. Agregar validaciones como al momento de buscar id debe ser diferente de 0, de lo contrario devolver un BadRequest().
3. Si no encuentra la villa entonces se retorna un NotFound();
4. Debemos documentar todos los diferentes tipos de código de estado que vamos a manejar en cada endpoint.
5. Agregar las validaciones en el Dto.
6. Agregar Microsoft AspnetCore.JsonPach **(Para trabajar con PATCH) : 1:38:13**

Captura de pantalla de un celular

Descripción generada automáticamente

Nota: Escoger la misma versión de nuestro proyecto

1. Agregar **Microsoft.AspNetCore.Mvc.NewtonsoftJson**



Nota: Escoger la misma versión de nuestro proyecto

1. Ahora nos vamos al program.cs y agregamos los nuevos servicios.

// Add services to the container.

builder.Services.AddControllers().AddNewtonsoftJson();

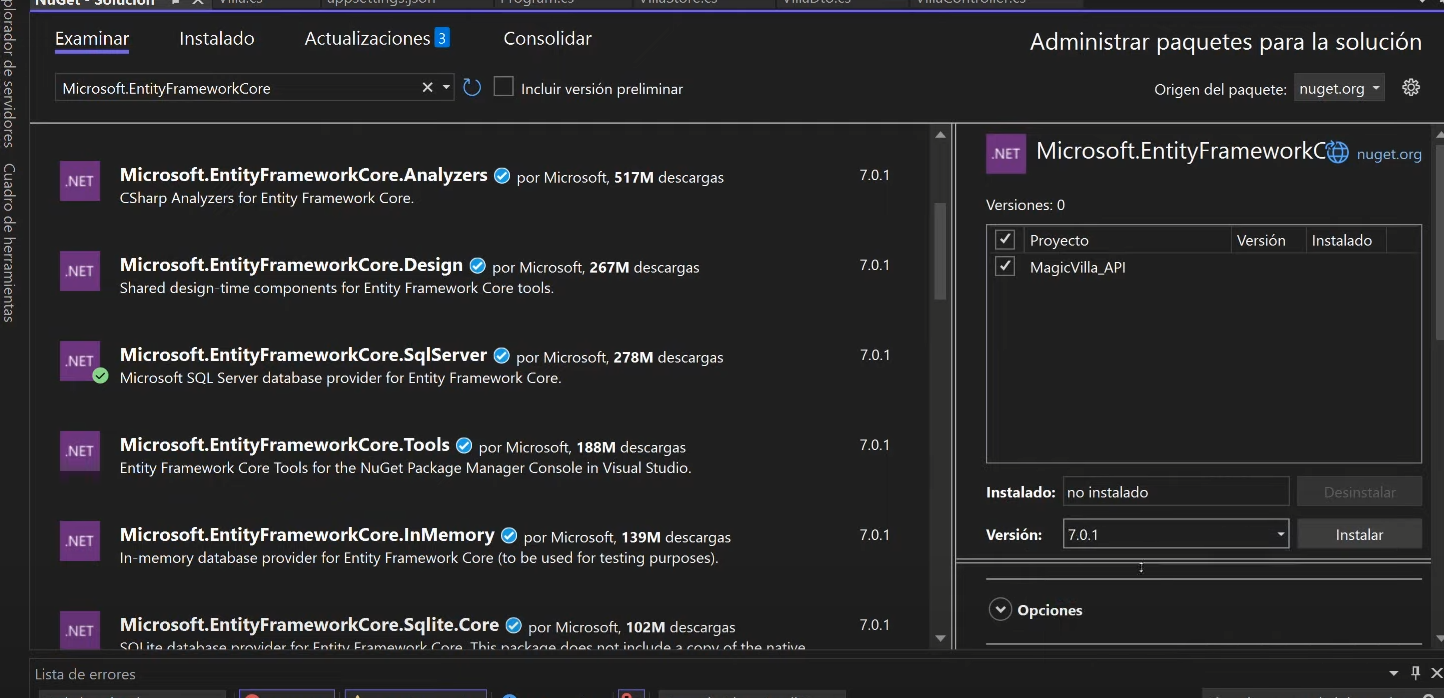
1. Para trabajar con patch necesitamos agregar esos paquetes nuget. JsonPath y NewtonsoftJson.
2. Trabajar con sql Server. Deberíamos instalar el relational también, pero ya al instalar instalar el **.SqlServer** ya viene instalado el rest. Ya viene incluido el EnitityFrameWork.Core y el Relational.

Interfaz de usuario gráfica, Texto, Sitio web

Descripción generada automáticamente

**Nota: Escoger la misma versión de nuestro proyecto**

1. Agregamos al **entityFramework.Tool** que esta relacionado a la creación de las tablas.



**Nota: Misma versión que la del proyecto.**

1. Modificando appsettings.json

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*",

"ConnectionStrings": {

"DefaultConnection": "Server=DESKTOP-V4RF20S;Database=MagicVilla;TrustServerCertificate=True;Trusted\_Connection=true;MultipleActiveResultSets=true"

}

}

1. Modificando Program.cs

builder.Services.AddDbContext<ApplicationDbContext>(option =>

{

option.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection"));

});

1. Ahora vamos a consola de administrador de paquetes Nuggets que está en herramientas.

add-migration AgregarBaseDeDatos 🡪 Crear un archivo de migración

update-database 🡪 Ejecuta el script creado en la carpeta de migración.

Si la conexión no se ejecuta correctamente puede ser porque la conexión no es segura. Esto se soluciona agregando el TrustServerCertificate=True

"DefaultConnection": "Server=DESKTOP-V4RF20S;Database=MagicVilla;TrustServerCertificate=True;Trusted\_Connection=true;MultipleActiveResultSets=true"

1. AsNoTracking se utiliza para indicar que no realice seguimiento de los objetos recuperados en la base de datos.
2. Crear los DTO para cada tipo de endpoint.
3. Crear los métodos asíncronos para los métodos del dbContext. Para eso se debe envolver el ActionsResult con un Task, y en el return colocar el await.

[HttpGet]

[ProducesResponseType(StatusCodes.Status200OK)]

public async Task<ActionResult<APIResponse>> GetVillas()

{

try

{

\_logger.LogInformation("Obtener las villas");

//return Ok(VillaStore.villaList);

IEnumerable<Villa> villaList = await \_villaRepo.ObtenerTodos();

\_response.Resultado = \_mapper.Map<IEnumerable<VillaDto>>(villaList);

\_response.StatusCode = HttpStatusCode.OK;

return Ok(\_response);

} catch (Exception ex)

{

\_response.IsExitoso = false;

\_response.ErrorMessages = new List<string>() { ex.ToString() };

}

return \_response;

}

1. Instalar AutoMapper y AutoMapper.Exntensions.Microsoft.DependencyInjection **(2:48:00).** Ayuda a la conversión de objetos.

Captura de pantalla de un celular

Descripción generada automáticamente

**Nota: Instalar la última versión, no una versión parecida a nuestro proyecto.**

1. Crear la clase MappingConfig

Texto

Descripción generada automáticamente

1. Crear el servicio en Program.cs

builder.Services.AddAutoMapper(typeof(MappingConfig));

1. Ahora debo crear las interfaces del repositorio
2. Agregar en el program.cs el AddScoped. Creamos el servicio y ya con eso podemos inyectarlo en cualquier lugar.

builder.Services.AddAutoMapper(typeof(MappingConfig));

builder.Services.AddScoped<IVillaRepositorio, VillaRepositorio>();