Documentación resumida sobre mis CRUD Endpoints del proyecto backend

El documento original en el que fui poniendo el 90% de lo que fui haciendo es una barbaridad de largo porque ya supera las 150 páginas, pero pasa que tienes bastantes cosas desfasadas ya que a mi código backend para la bd hice muchos cambios.

Entonces en este resumen voy a mostrar la situación actual a 13 de noviembre de 2025.

Aclarar que voy con retraso respecto a la referencia de los entregables/semanas, pero considero que he conseguido esta mañana lograr lo esencial, que es lo siguiente:

Poder con Swagger insertar registros en la tabla de reservas con POST api reservas y consultarlos con GET api reservas sin problemas de "cyles" e incluyendo en el resultado de ambas operaciones las propiedades de navegación "obligatorias" de mi modelo de reservas, no con valor 'null', sino con el objeto que corresponda a la fk que yo paso al hacer el POST; dichas propiedades son Usuario e Instalacion:

```
// Propiedad de navegación:
5 references
public Usuario Usuario { get; set; } = null!;
```

```
// Propiedad de navegación:
5 references
public Instalacion Instalacion { get; set; } = null!;
```

Con "obligatorias" me refiero a que en su tipo de dato NO pongo '?' Y acordemente pongo = null! como diciendo "prometo que no serán null aunque así las inicialice" (esto estaba más pensado para un código antiguo que para el actual, pero por si acaso lo dejo).

En concreto, para lograr lo antedicho, redacté el siguiente código que emplea DTOs (la otra manera de evitar problemas de cycles es poniendo un código en Program.cs, pero entonces las consultas aparecen con bastantes $id y $ref que dificultan notablemente la compresión de la consulta, por eso opté por la solución de usar DTOs aunque fuese algo más compleja):

PÁGINA SIGUIENTE

```
namespace ReservasApi.Modelos.DTO
{
    1 reference
    public class ReservaCrearDto // this defines the fields Swagger will ask for when cr
    {
        // Primary key en EFC es por defecto autoincrementable al ser de tipo int.

        // Claves foráneas:
        2 references
        public string Dni { get; set; } = string.Empty; // tal cual está en Reserva.cs
        2 references
        public int InstalacionId { get; set; } // tal cual está en Reserva.cs

        // Campos "normales y corrientes" (i.e. ni son pk ni son fk)
        1 reference
        public DateTime FechaHora { get; set; } // tal cual en Reserva.cs. FechaHora es
        1 reference
        public int NumeroAsistentes { get; set; } // tal cual en Reserva.cs
    }
}
```

"**POST will use this DTO only, nav properties are set server-side**", es decir, que yo en el swagger POST no voy a asignar objetos a nav properties, a esta respecto solo pondré las fks.

En segundo lugar, **creo ReservaLeerDto** (for GET) i.e. "we create a separate DTO **for GET (el de reservas) to avoid cycles**:"

```
namespace ReservasApi.Modelos.DTO
{
    4 references
    public class ReservaLeerDto
    {
        // primary key:
        2 references
        public int ReservaId { get; set; } // esto (la pk de tabla de reservas) no lo puse en ReservaCrearDto.cs
        // normal fields:
        2 references
        public DateTime FechaHora { get; set; } // tal cual en ReservaCrearDto.cs
        2 references
        public int NumeroAsistentes { get; set; } // tal cual en ReservaCrearDto.cs
        // foreign keys:
        2 references
        public string Dni { get; set; } = string.Empty; // tal cual en ReservaCrearDto.cs
        2 references
        public int InstalacionId { get; set; } // tal cual en ReservaCrearDto.cs

        // Navigation objects (only basic info to avoid cycles):
        2 references
        public UsuarioDto? Usuario { get; set; }
        2 references
        public InstalacionDto? Instalacion { get; set; }

    }
}
```

Escribir esos 2 navigation objects me obligó a crear UsuarioDto e InstalacioniDto, lo hice:

PÁGINA SIGUIENTE

```
namespace ReservasApi.Modelos.DTO
{
    3 references
    public class UsuarioDto
    {
        2 references
        public string Dni { get; set; } = string.Empty;
        2 references
        public string NombreCompleto { get; set; } = string.Empty;
        // Add any other Usuario fields you want exposed
    }
}
```

"you want exposed" = que quieres que aparezcan en el get api reservas de swagger.¶

```
namespace ReservasApi.Modelos.DTO
{
    3 references
    public class InstalacionDto
    {
        2 references
        public int InstalacionId { get; set; }
        2 references
        public string Localizacion { get; set; } = string.Empty;
        // Add any other Instalacion fields you want exposed.
    }
}
```

En tercer lugar, **editamos ReservasController.cs** en lo que respecta a **post api reservas** y **get api reservas**, quedando la cosa así:¶

```
[HttpPost] // matches POST /api/reservas
0 references
public async Task<ActionResult<ReservaLeerDto>> CrearReserva(ReservaCrearDto dto)// (Reserva nuevareserva)
{ // the body of the request must contain a JSON representation of a Reserva (de la que quieres crear, supongo).
    // comienzo del código propio del DTO:
    // Validate that the user and installation exist :
    var usuario = await _contexto.Usuarios.FindAsync(dto.Dni); // Usuarios es un DbSet en mi AppDbContext.cs
    var instalacion = await _contexto.Instalaciones.FindAsync(dto.InstalacionId); // Instalaciones es un DbSet en mi dbcontext

    if (usuario == null || instalacion == null) // validamos that the user and installation exist.
    {
        return BadRequest("Usuario o Instalación NO encontrados.");
    }

    // Create the new booking and link navigation properties (para que 'usuario' e 'instalacion' -navigation properties del modelo de reservas- se ac
    var nuevareserva = new Reserva // each time you create a new reservation object in memory inside your controller, that instance starts with Usuar
    {
        Dni = dto.Dni, InstalacionId = dto.InstalacionId, FechaHora = dto.FechaHora, NumeroAsistentes = dto.NumeroAsistentes,

        // Attach navigation properties (so that EF Core also attaches the Usuario and Instalacion entities based on fks 'dni' and 'instalacionId'):
        Usuario = usuario, // we're populating the existing nav props with the related entities ('usuario' and 'instalacion') fetched from the db.
        Instalacion = instalacion, // if you don't assign these nav props before saving, then when you later query the record, unless you use .Includ
    }; // explicitly assigning nav props affects writing/saving, whereas .Include(...) affects reading/querying. I should include both.

    // creates and saves a new Reserva en otras palabras Add and save:
    _contexto.Reservas.Add(nuevareserva); // EFCore adds it (la nueva reserva) to the database (Add + SaveChangesAsync()).
    await _contexto.SaveChangesAsync(); // after SaveChangesAsync(), the navigation properties are available and populated in memory. If you GET /api
```

¶

Sigue y finaliza en página siguiente

```
    // Map to DTO for response:
    var reservaDto = new ReservaLeerDto
    {
        ReservaId = nuevareserva.ReservaId,
        FechaHora = nuevareserva.FechaHora,
        NumeroAsistentes = nuevareserva.NumeroAsistentes,
        Dni = nuevareserva.Dni,
        InstalacionId = nuevareserva.InstalacionId,

        Usuario = new UsuarioDto
        {
            Dni = usuario.Dni,
            NombreCompleto = usuario.NombreCompleto
        },

        Instalacion = new InstalacionDto
        {
            InstalacionId = instalacion.InstalacionId,
            Localizacion = instalacion.Localizacion
        }
    };

    // (Optional but clean) reload navigation properties to ensure they are fully populated and also ensures Swagger shows them in the response:
    // await _contexto.Entry(nuevareserva).Reference(r => r.Usuario).LoadAsync();
    // await _contexto.Entry(nuevareserva).Reference(r => r.Instalacion).LoadAsync();

    // Publish event to Kafka topic "reservas.creadas"
    // awai _kafkaProducer.PublishReservaCreadaAsync(nuevareserva);

    // Return Created (201)
    return CreatedAtAction(nameof(GetReservas), new { id = nuevareserva.ReservaId }, reservaDto); // antes del 13 de nov aquí ponía nameof(GetReserva
} // returns '201 Created' with the new object (la nueva reserva) and a Location header pointing to /api/reservas/{id}
```

En PÁGINA SIGUIENTE el get api reservas

Y·el·**get·api·reservas**·queda·ahora·así:¶

```
[HttpGet]
1 reference
public async Task<ActionResult<IEnumerable<ReservaLeerDto>>> GetReservas()
{
    var reservas = await _contexto.Reservas // Reservas viene de public DbSet<Reserva> Reservas { ge
        .Include(r => r.Usuario) // "eager loading with .Include()
        .Include(r => r.Instalacion) // so EF Core brings in the related objects automatically ("rel
        .ToListAsync(); // sin .Include este GET method would retrieve all bookings without includin
    // ToListAsync() asynchronously fetches all rows (de la tabla de reservas)

    // esto lo añado con intención de poder consultar get api reservas en swagger sin cycle problem:
    var resultado = reservas.Select(r => new ReservaLeerDto
    {
        ReservaId = r.ReservaId,
        FechaHora = r.FechaHora,
        NumeroAsistentes = r.NumeroAsistentes,
        Dni = r.Dni,
        InstalacionId = r.InstalacionId,

        Usuario = r.Usuario == null ? null : new UsuarioDto
        {
            Dni = r.Usuario.Dni,
            NombreCompleto = r.Usuario.NombreCompleto
        },

        Instalacion = r.Instalacion == null ? null : new InstalacionDto
        {
            InstalacionId = r.Instalacion.InstalacionId,
            Localizacion = r.Instalacion.Localizacion
        }
    }).ToList();

    return Ok(resultado); // el return "sends HTTP 200 with a JSON array of reservations".
```

¶

}·//·fin·del·GET·/api/reservas¶

Habiendo·hecho·esos·cambios·y·para·que·surtan·efecto,·ejecuto·**docker-compose·up·--build·backend**·desde·carpeta·padre·del·docker-compose.yml·(no·hace·falta·parar·los·contenedores)·y·entonces·al·hacer·un·**get·api·reservas·con·swagger**·obtengo·lo·que·quería·(i.e.·no·cycle·problem·y·no·nulos·en·propds·de·navegacion·usuario·e·instalacion)·(el·POST·ya·lo·había·hecho·en·el·pasado,·con·código·algo·más·antiguo).¶

En·NEXT·PAGE·el·get·api·reservas·que·logro·con·esos·nuevos·cambios.¶

PÁGINA SIGUIENTE

Reservas

POST /api/Reservas

GET /api/Reservas

Parameters

No parameters

Execute

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:5000/api/Reservas' \
  -H 'accept: text/plain'
```

Request URL

```
http://localhost:5000/api/Reservas
```

Server response

Code	Details
200	Response body

```
[
  {
    "reservaId": 1,
    "fechaHora": "2025-11-25T15:00:00.768",
    "numeroAsistentes": 6,
    "dni": "98761234Z",
    "instalacionId": 1,
    "usuario": {
      "dni": "98761234Z",
      "nombreCompleto": "Laura Gomez"
    },
    "instalacion": {
      "instalacionId": 1,
      "localizacion": "Avenida de la Fuerza, Barcelona"
    }
  }
]
```

Response headers

```
content-type: application/json; charset=utf-8
date: Thu,13 Nov 2025 09:46:06 GMT
server: Kestrel
transfer-encoding: chunked
```

PÁGINA SIGUIENTE

Prueba a postear una segunda reserva para ver si **el post** sigue funcionando correctamente después de esos cambios, lo que observo es que no solo sigue **funciona**ndo bien, sino **mejor**, ya que ahora en el propio resultado del post ya no pone null en propds de navegacion Usuario e Instalacion, sino que pone los objetos correspondientes a los valores de las fk que yo pasé al hacer el swagger POST. Muestro este nuevo POST:

Clico en Execute y entonces se ve esto:

PÁGINA SIGUIENTE

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:5000/api/Reservas' \
  -H 'accept: text/plain' \
  -H 'Content-Type: application/json' \
  -d '{
  "dni": "98761234Z",
  "instalacionId": 1,
  "fechaHora": "2025-11-30T10:00:00.287Z",
  "numeroAsistentes": 2
}'
```

Request URL

```
http://localhost:5000/api/Reservas
```

Server response

Code	Details
201 *Undocumented*	Response body

```
{
  "reservaId": 2,
  "fechaHora": "2025-11-30T10:00:00.287Z",
  "numeroAsistentes": 2,
  "dni": "98761234Z",
  "instalacionId": 1,
  "usuario": {
    "dni": "98761234Z",
    "nombreCompleto": "Laura Gomez"
  },
  "instalacion": {
    "instalacionId": 1,
    "localizacion": "Avenida de la Fuerza, Barcelona"
  }
}
```

Response headers

```
content-type: application/json; charset=utf-8
date: Thu,13 Nov 2025 10:20:25 GMT
location: http://localhost:5000/api/Reservas?id=2
server: Kestrel
transfer-encoding: chunked
```

Página siguiente

Vuelvo a ejecutar get api reservas, veo las 2 reservas insertadas:

Code | Details

200

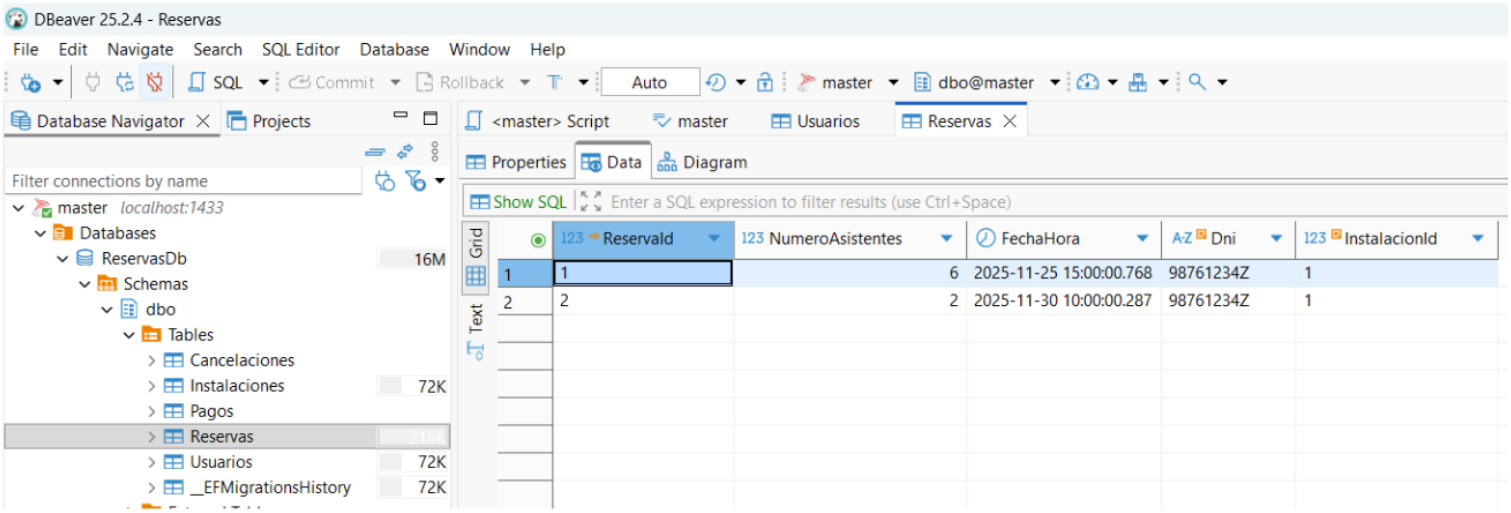
Response body


```
[
  {
    "reservaId": 1,
    "fechaHora": "2025-11-25T15:00:00.768",
    "numeroAsistentes": 6,
    "dni": "98761234Z",
    "instalacionId": 1,
    "usuario": {
      "dni": "98761234Z",
      "nombreCompleto": "Laura Gomez"
    },
    "instalacion": {
      "instalacionId": 1,
      "localizacion": "Avenida de la Fuerza, Barcelona"
    }
  },
  {
    "reservaId": 2,
    "fechaHora": "2025-11-30T10:00:00.287",
    "numeroAsistentes": 2,
    "dni": "98761234Z",
    "instalacionId": 1,
    "usuario": {
      "dni": "98761234Z",
      "nombreCompleto": "Laura Gomez"
    },
    "instalacion": {
```



```
    "instalacion": {
      "instalacionId": 1,
      "localizacion": "Avenida de la Fuerza, Barcelona"
    }
  }
]
```


La misma consulta en DBeaver en página siguiente



En·DBeaver·no·aparecen·las·propds·de·navgcón·en·la·consulta,·solo·sus·fk.¶

Tengo pendiente:

- Lograr lo mismo (i.e. get y post api reservas sin cycle problems e incluyendo propds de navegación con valores correspondientes a las fk que se pasen por POST) en las demás tablas (que son 4), amén de probar en todas que PUT osease update y DELETE funcionan bien (quizás tenga que cambiar aquí código al haber hecho tantos cambios a lo demás).

- También tengo pendiente hacer las mismas pruebas con POSTMAN y con TDD enfocado en test unitarios realizados con xUnit (.NET).