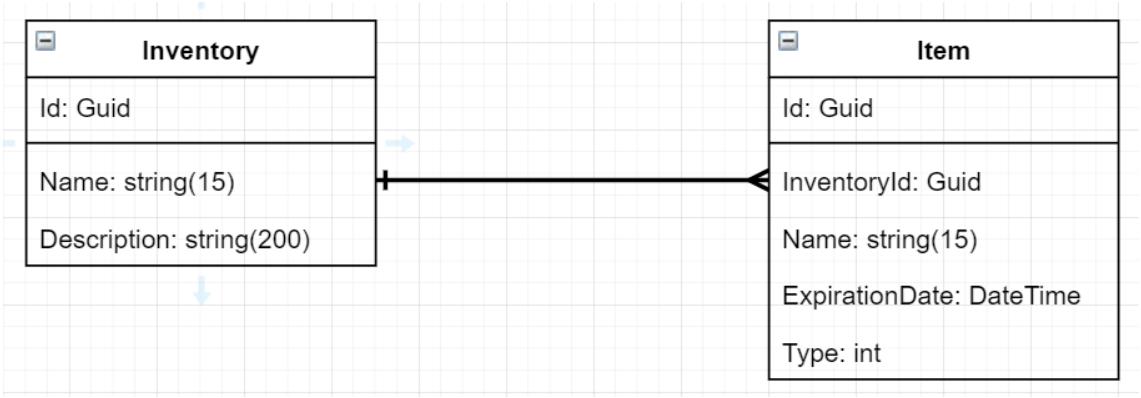
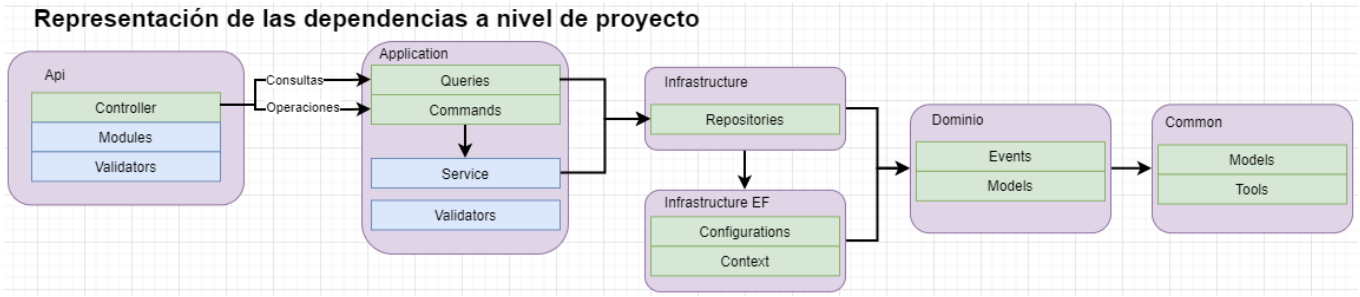
# Documento técnico Proyecto Goal Challenge

1. Modelo de datos

En base a la interpretación de la prueba, los ítems pertenecen a un inventario, por lo tanto, puede haber varios inventarios con sus ítems correspondientes.




2. Arquitectura




a. Basada en CQRS y DDD.

Las Queries y Comandos están separados. Las queries usan directamente EF Core para la devolución de los datos. Los comandos con acceso exclusivo por un mediador (MediatR) realizan las operaciones de creación, actualización y borrado.

```
[HttpGet]
0 references
public dynamic Get()
{
    return _itemQuery.GetAllItems();
}
```

```
1 reference
public class ItemQuery : IItemQuery
{
    private readonly IInventoryRepository _itemRepository;

    0 references
    public ItemQuery(IInventoryRepository itemRepository)
    {
        _itemRepository = itemRepository ?? throw new ArgumentNullException(nameof(itemRepository));
    }

    2 references
    public List<Inventory> GetAllItems()
    {
        return _itemRepository.GetAllItemsFromInventory();
    }
}
```

Los servicios o casos de uso están enteramente relacionado con los comandos, siendo estos últimos el caso de uso por diseño. El servicio lo interpreto como el algoritmo en sí mismo para realizar los pasos necesarios para ejecutar el caso de uso. Como otro punto de vista, en el pdf que

aporto al proyecto llamado “Arquitectura Otra.pdf” sugiero distintas factorías para ayudar en la complejidad de cada paso en el algoritmo (Servicio), dando más intención al principio de responsabilidad única, ya que cada método de la factoría hace una sola cosa, además que es reutilizable por otros servicios que lo requieran.

```
5 references
public void AddItem(Item addItem)
{
    Tools.ArgumentNull(addItem, nameof(addItem));

    Items.Add(addItem);
    Events.Add(new ItemAddInInventoryDomainEvent(addItem));
}

5 references
public void RemoveItem(Item removeItem)
{
    Tools.ArgumentNull(removeItem, nameof(removeItem));

    Items.Remove(removeItem);
    Events.Add(new ItemRemovedFromInventoryDomainEvent(removeItem, Name));
}
```

Esta arquitectura recae finalmente sobre el dominio, será quien cambiará de estado o de propósito ejecutando intrínsecamente los eventos de dominio.
El lanzamiento de los eventos se realiza en el propio contexto, ya que, es cuando es consciente del cambio de estado del dominio y obtiene el listado de eventos listo para ser lanzados.

```
3 references
public override async Task<int> SaveChangesAsync(CancellationToken cancellationToken = new CancellationToken())
{
    int result = await base.SaveChangesAsync(cancellationToken).ConfigureAwait(false);

    var entitiesWithEvents = ChangeTracker.Entries<EntityBase>()
        .Select(e => e.Entity)
        .Where(e => e.Events.Any());

    foreach (var entity in entitiesWithEvents)
    {
        var events = entity.Events;

        foreach (var domainEvent in events)
        {
            await _mediator.Publish(domainEvent).ConfigureAwait(false);
        }

        entity.Events.Clear();
    }

    return result;
}
```

Los eventos, particulares de los cambios producidos en el dominio, también son accedidos a través del mediador para desacoplar la funcionalidad.

He implementado una capa Common que me ayuda a reutilizar código como un control de argumentos

```
11 references
public static void ArgumentNull<T>(T argument, string operation) => _ = argument ?? throw new ArgumentNullException(operation);
```

Una pequeña función para extraer las diferencias de dos listas de elementos

```
2 references
public static List<T> ExceptionLists<T>(List<T> first, List<T> second) where T : class => first.Except(second).ToList();
```

Y un método de extensión del ValidationResult de Fluent Validation para un código más limpio y menos repetitivo.

```
2 references
public static void Result(this ValidationResult validationResult)
{
    if (validationResult != null && !validationResult.IsValid)
    {
        throw new Exception(validationResult.ToString("#"));
    }
}
```

También he implementado un par de modelos input que los usa tanto el controller para recogida de datos como los comandos para su uso, todo ello gobernado por el control de validación de Fluent Validator

```
1 reference
public class InventoryInputValidator : AbstractValidator<InventoryInput>
{
    0 references
    public InventoryInputValidator()
    {
        RuleFor(x => x.Name)
            .NotEmpty()
            .MaximumLength(15);

        RuleFor(x => x.Description)
            .MaximumLength(200);

        RuleFor(x => x.Items)
            .NotNull()
            .NotEmpty();
    }
}
public class ItemInputValidator : AbstractValidator<ItemInput>
{
    /// <summary>
    /// Validator item inputs
    /// </summary>
    0 references
    public ItemInputValidator()
    {
        RuleFor(c => c.Name)
            .NotEmpty()
            .MaximumLength(15);

        RuleFor(c => c.Type)
            .NotEmpty();
    }
}
```

En el apartado del control de dependencias, he optado por Autofac, un potente contendor de inversión de control, donde el código que se implementaes claro y limpio y su mantenibilidad es muy alta, destaco la forma de identificar el ensamblado para que Autofac haga el trabajo de registrarlo todo. En pocas líneas de código, tienes registrado todo lo necesario para hacer funcionar la aplicación, en contra partida con otras opciones, en cada registro se declara cada implementación con su interfaz, en proyectos muy grande, es una gran desventaja el nivel de mantenimiento que aporta.

```
protected override void Load(ContainerBuilder builder)
{
    var assemblies = new System.Reflection.Assembly[]
    {
        //Application
        typeof(GoalChallenge.Application.Commands.Items.AddItemsToInvetoryCommand).Assembly,

        // Infrastructure
        typeof(Infrastructure.Data.Repositories.Items.IInventoryRepository).Assembly,

        // Domain
        typeof(Domain.Events.Base.BaseDomainEvent).Assembly,

       // Api.Query
        typeof(Api.Query.Queries.Items.IItemQuery).Assembly,

       System.Reflection.Assembly.GetExecutingAssembly()
    };

    builder.RegisterAssemblyTypes(typeof(IMediator).GetTypeInfo().Assembly).AsImplementedInterfaces();

    builder.RegisterAssemblyTypes(assemblies).AsImplementedInterfaces().InstancePerDependency();

    builder.Register<Serilog.ILogger>((c, p) =>
    {
        return new LoggerConfiguration()
            .WriteTo.Console().
            CreateLogger();
    }).InstancePerLifetimeScope();

    //builder.Register(c => new SqlConnection(connectionString)).As<IDbConnection>().InstancePerLifetimeScope();

    var dbContextOptionsBuilder = new DbContextOptionsBuilder<EFContext>().UseInMemoryDatabase(databaseName: "Test").UseLazyLo

    builder.RegisterType<EFContext>()
        .WithParameter("options", dbContextOptionsBuilder.Options)
        .InstancePerLifetimeScope();
}
```

Para el apartado de logs, he usado Serilog, un sistema de logging estructurado. En esta aplicación, lo he usado principalmente para mostrar a través de la consola, los mensajes de los eventos producidos por los cambios de estado del dominio, insertar y eliminar ítems.
Sinceramente, no uso demasiado loggers en mis aplicaciones, por el contrario, suelo usar un sistema de mensajería a través de un Service Bus, con el consumidor de turno, para almacenar mensajes de auditoria o cualquier cosa que tenga interés. Seguramente Serilog me pueda resolver muchas complejidades que he tenido en el pasado y ahora con la inclusión de Serilog, he visto su potencial.

b. Queries desacopladas

En este apartado quería mencionar en base a CQRS la intención de desacoplar las consultas del resto de operaciones. Una de las ideas que surge al separar, entre otras, en casos especiales que se produzcan muchísimas consultas, es el poder trasladar a servidores, contenedores o similares la ejecución exclusiva de las queries, aumentando o disminuyendo los recursos necesarios para el buen rendimiento del sistema.
Ahora bien, para implementar esta capa, personalmente me decanto por un micro ORM llamado Dapper. Antiguamente, el uso de Entity Framework, con sus edmx maravillosos (ironía), hacían que las consultas eran fáciles de crear, pero difíciles optimizar y mantenimiento complejo etc…
Hará como 8 años atrás, conocí este micro ORM y empecé a usarlo en mis aplicaciones, el rendimiento fue exponencial y su mantenimiento en igual medida, nos deshicimos de generalizaciones, eliminamos los IQueryables asesinos de rendimiento y todo cambio a partir de ese momento, pero esto no quiere decir que Entity Framework no se use, todo lo contrario, se ha aprendido mucho, se ha reescrito por completo en las versiones NET y su rendimiento ha cambiado muchísimo, pero para las consultas, habrá que analizar y diseñar bien para sugerir un ORM o un micro ORM en cada caso.

En este proyecto reto, no he podido usarlo como me gustaría, pero debo mencionarlo y dejar unas trazas de su uso con ayuda de una implementación de métodos encapsulados para hacer más fácil al desarrollador su uso (GoalChallenge.Data.Dapper).
Se ha hecho uso de SonarLint para el análisis de código limpio en Visual Studio 2022.

3. Requerimientos no implementados.

    a. Implementación de un Frontend con VUE3 o REACT.

No he realizado este desarrollo porque no es mi fuerte, con más tiempo podría montarme un entorno para montar estas tecnologías u otras, pero en mi trayectoria profesional me he decantado más por la arquitectura backend, rendimiento y optimización.

    b. API Segura.

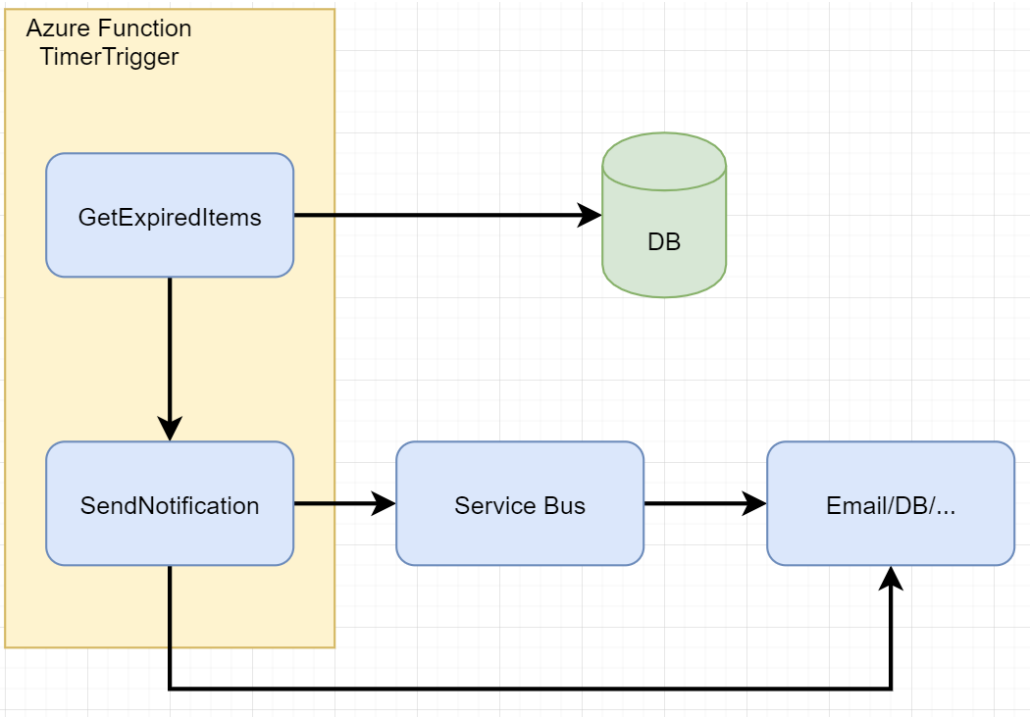
Al no implementar la parte Frontend he dedicado el tiempo a mejorar el resto de la arquitectura, aunque hay otros elementos que quería implementar, pero no lo hecho, como Middlewares de excepciones, atributos de control de auditoría para endpoints etc, siempre queda algo en el tintero que quieres introducir.

    c. Requisito de notificación cuando expira un item.

No lo he implementado, pero expongo el análisis y el diseño de como lo haría.

Crearía un Azure Function Timer Trigger, obtendría los ítems expirados vía query y a la hora de enviar las notificaciones usaría un Service bus para encolar las notificaciones y organizar los envíos o directamente si la estimación de notificaciones no es muy alta.

Como canal de comunicación he elegido el email, pero podría ser cualquier cosa, como la escritura en alguna tabla para que aparezca en un dashboard, inclusive usar SignalR para una actualización en tiempo real.



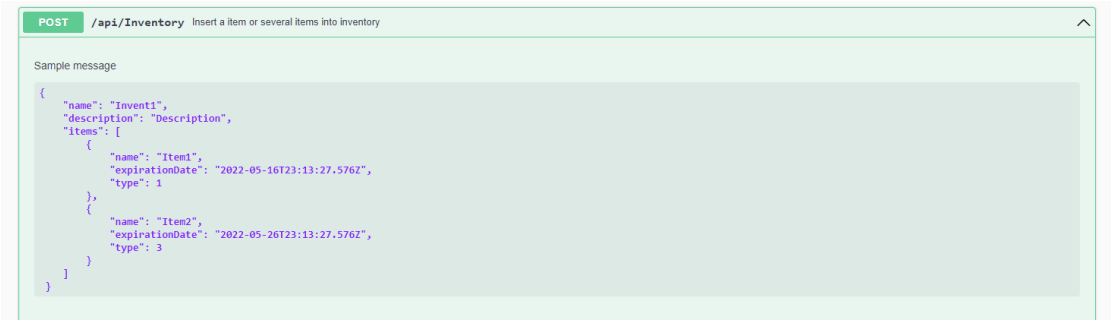
4. Instrucciones uso de la aplicación

Abrir la solución con Visual Studio 2022

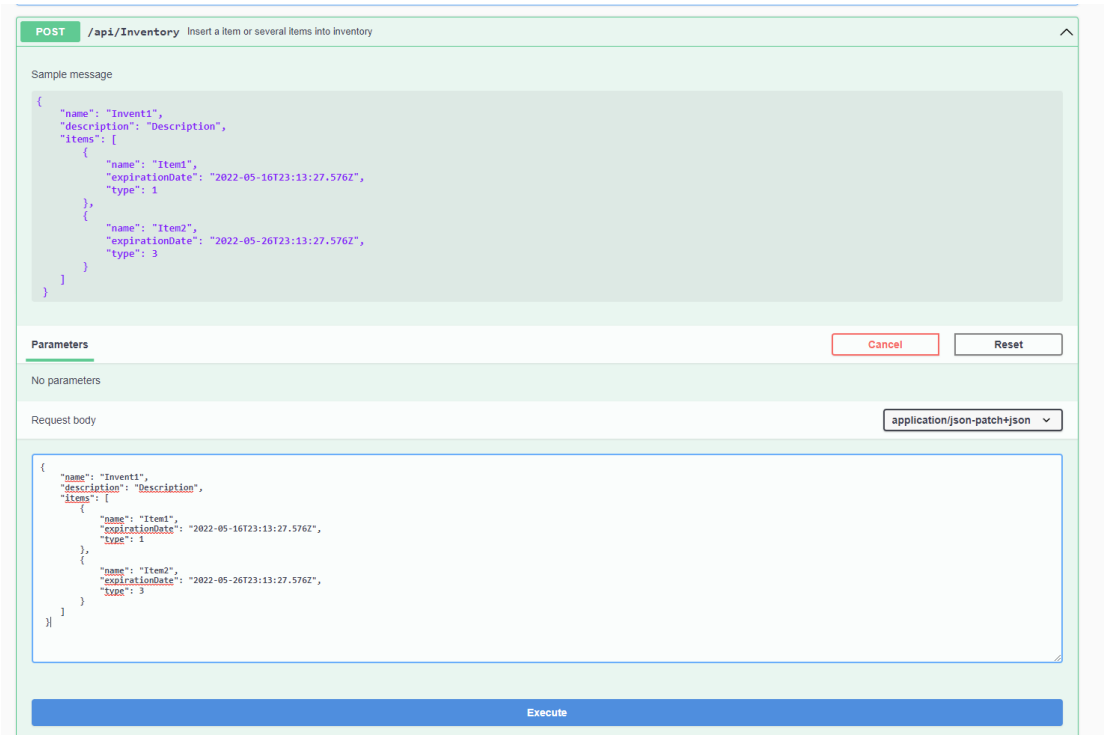
Ejecutar la solución

Se abre el navegador mostrando el swagger UI con las siguientes operaciones

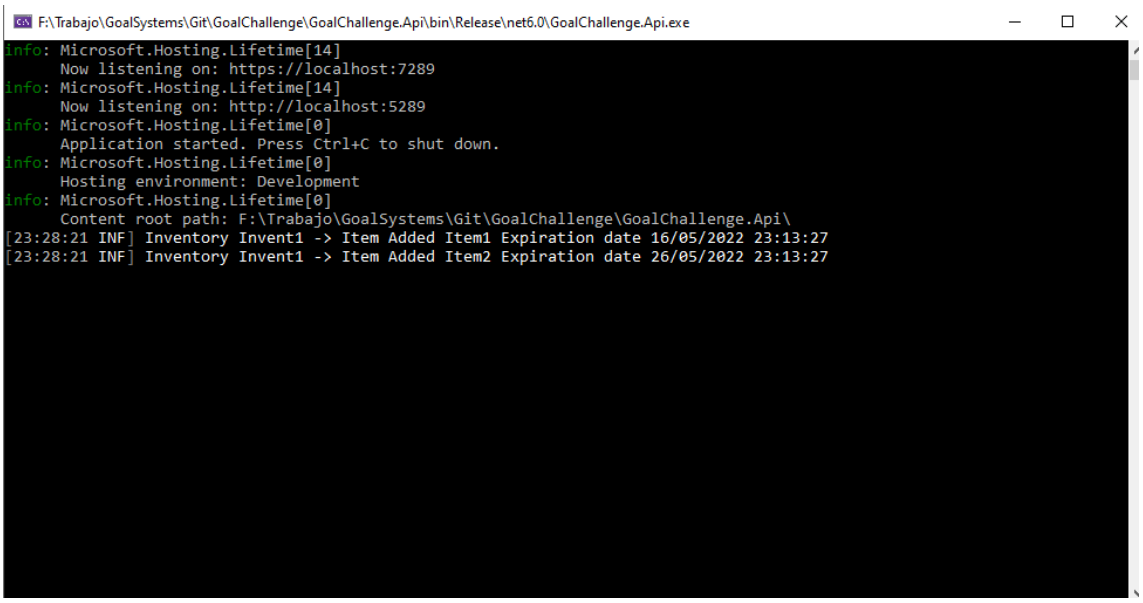

Para poblar la base de datos con ítems usar la operación POST




Aparece un mensaje de prueba en formato json listo para usarlo.Se pulsa el botón Try it out, eliminar el contenido que aparece y copiar el mensaje de prueba y pulsar Execute



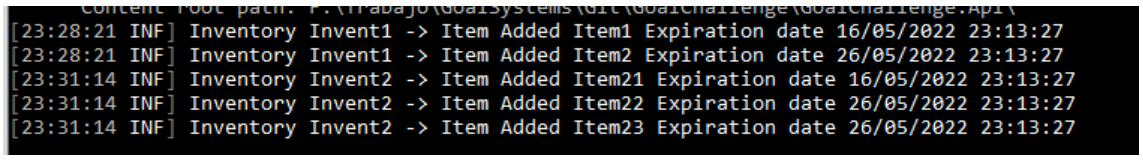
En la consola de la aplicación aparecen los mensajes de los eventos de dominio relacionado con los ítems.




Volver a repetir los pasos anteriores para añadir ítems al inventario actual, o crear nuevos ítems en otro inventario.

Ejemplo1: Crear nuevo inventario con sus items

```
{
    "name": "Invent2",
    "description": "Description",
    "items": [
        {
            "name": "Item21",
            "expirationDate": "2022-05-16T23:13:27.576Z",
            "type": 1
        },
        {
            "name": "Item22",
            "expirationDate": "2022-05-26T23:13:27.576Z",
            "type": 3
        },
        {
            "name": "Item23",
            "expirationDate": "2022-05-26T23:13:27.576Z",
            "type": 3
        }
```



Ejemplo2: Crear y actualizar nuevos ítems en el inventario 1. Elimina aquellos que no estén en la nueva lista

```
{
    "name": "Invent1",
    "description": "Description",
    "items": [
        {
            "name": "Item2",
            "expirationDate": "2022-05-16T23:13:27.576Z",
            "type": 1
        },
        {
            "name": "Item4",
            "expirationDate": "2022-05-26T23:13:27.576Z",
            "type": 3
        },
        {
            "name": "Item5",
            "expirationDate": "2022-05-26T23:13:27.576Z",
            "type": 3
        }
    ]
}
```

```
[23:28:21 INF] Inventory Invent1 -> Item Added Item1 Expiration date 16/05/2022 23:13:27
[23:28:21 INF] Inventory Invent1 -> Item Added Item2 Expiration date 26/05/2022 23:13:27
[23:31:14 INF] Inventory Invent2 -> Item Added Item21 Expiration date 16/05/2022 23:13:27
[23:31:14 INF] Inventory Invent2 -> Item Added Item22 Expiration date 26/05/2022 23:13:27
[23:31:14 INF] Inventory Invent2 -> Item Added Item23 Expiration date 26/05/2022 23:13:27
[23:33:00 INF] Inventory Invent1 -> Item Added Item4 Expiration date 26/05/2022 23:13:27
[23:33:00 INF] Inventory Invent1 -> Item Added Item5 Expiration date 26/05/2022 23:13:27
[23:33:00 INF] Inventory Invent1 -> Item Removed: Item1
```

Ejemplo3: Insertar los mismos ítems anteriores en el inventario 2. Los ítems iniciales del inventario 2, al no coincidir, se eliminan

```
{
    "name": "Invent2",
    "description": "Description",
    "items": [
        {
            "name": "Item2",
            "expirationDate": "2022-05-16T23:13:27.576Z",
            "type": 1
        },
        {
            "name": "Item4",
            "expirationDate": "2022-05-26T23:13:27.576Z",
            "type": 3
        },
        {
            "name": "Item5",
            "expirationDate": "2022-05-26T23:13:27.576Z",
            "type": 3
        }
    ]
}
```

```
[23:28:21 INF] Inventory Invent1 -> Item Added Item1 Expiration date 16/05/2022 23:13:27
[23:28:21 INF] Inventory Invent1 -> Item Added Item2 Expiration date 26/05/2022 23:13:27
[23:31:14 INF] Inventory Invent2 -> Item Added Item21 Expiration date 16/05/2022 23:13:27
[23:31:14 INF] Inventory Invent2 -> Item Added Item22 Expiration date 26/05/2022 23:13:27
[23:31:14 INF] Inventory Invent2 -> Item Added Item23 Expiration date 26/05/2022 23:13:27
[23:33:00 INF] Inventory Invent1 -> Item Added Item4 Expiration date 26/05/2022 23:13:27
[23:33:00 INF] Inventory Invent1 -> Item Added Item5 Expiration date 26/05/2022 23:13:27
[23:33:00 INF] Inventory Invent1 -> Item Removed: Item1
[23:36:25 INF] Inventory Invent2 -> Item Added Item2 Expiration date 16/05/2022 23:13:27
[23:36:25 INF] Inventory Invent2 -> Item Added Item4 Expiration date 26/05/2022 23:13:27
[23:36:25 INF] Inventory Invent2 -> Item Added Item5 Expiration date 26/05/2022 23:13:27
[23:36:25 INF] Inventory Invent2 -> Item Removed: Item21
[23:36:25 INF] Inventory Invent2 -> Item Removed: Item22
[23:36:25 INF] Inventory Invent2 -> Item Removed: Item23
```

En cualquier momento se puede usar la operación GET que obtendrá la lista completa de todos los ítems en cada inventario.

```
[
  {
    "events": [],
    "items": [
      {
        "events": [],
        "name": "Item2",
        "expirationDate": "2022-05-26T23:13:27.576Z",
        "type": 3,
        "inventoryId": "122f7453-f9fa-449e-b2a9-014a4e886079",
        "id": "bc2cefdb-cd59-40f2-b4f4-452b70c90a27",
        "creationDate": "2022-05-08T21:28:20.4083968Z",
        "modificationDate": "2022-05-08T21:28:20.4083969Z"
      },
      {
        "events": [],
        "name": "Item4",
        "expirationDate": "2022-05-26T23:13:27.576Z",
        "type": 3,
        "inventoryId": "122f7453-f9fa-449e-b2a9-014a4e886079",
        "id": "48370414-ed5f-4ac3-8f9f-ac97a05cf3cc",
        "creationDate": "2022-05-08T21:33:00.4575716Z",
        "modificationDate": "2022-05-08T21:33:00.4575717Z"
      },
      {
        "events": [],
        "name": "Item5",
        "expirationDate": "2022-05-26T23:13:27.576Z",
```


La operación DELETE elimina el o los ítems de cualquier inventario que coincida con el nombre introducido. En el ejemplo, introducimos el valor “Item5” y se eliminará tanto del inventario1 como del inventario2

```
[23:28:21 INF] Inventory Invent1 -> Item Added Item1 Expiration date 16/05/2022 23:13:27
[23:28:21 INF] Inventory Invent1 -> Item Added Item2 Expiration date 26/05/2022 23:13:27
[23:31:14 INF] Inventory Invent2 -> Item Added Item21 Expiration date 16/05/2022 23:13:27
[23:31:14 INF] Inventory Invent2 -> Item Added Item22 Expiration date 26/05/2022 23:13:27
[23:31:14 INF] Inventory Invent2 -> Item Added Item23 Expiration date 26/05/2022 23:13:27
[23:33:00 INF] Inventory Invent1 -> Item Added Item4 Expiration date 26/05/2022 23:13:27
[23:33:00 INF] Inventory Invent1 -> Item Added Item5 Expiration date 26/05/2022 23:13:27
[23:33:00 INF] Inventory Invent1 -> Item Removed: Item1
[23:36:25 INF] Inventory Invent2 -> Item Added Item2 Expiration date 16/05/2022 23:13:27
[23:36:25 INF] Inventory Invent2 -> Item Added Item4 Expiration date 26/05/2022 23:13:27
[23:36:25 INF] Inventory Invent2 -> Item Added Item5 Expiration date 26/05/2022 23:13:27
[23:36:25 INF] Inventory Invent2 -> Item Removed: Item21
[23:36:25 INF] Inventory Invent2 -> Item Removed: Item22
[23:36:25 INF] Inventory Invent2 -> Item Removed: Item23
[23:40:38 INF] Inventory Invent2 -> Item Removed: Item5
[23:40:38 INF] Inventory Invent1 -> Item Removed: Item5
```