Creating the Inventory service

In this lesson we will create our Inventory microservice, which owns the player's inventory of purchased items.

Inventory REST API

Just like with did with the Catalog service we will define a REST API for Inventory with a couple of operations, which are:

POST /items: Which will be used to add an item to the player's inventory bag, and
GET /items by user id: Which should retrieve all the items for the specified player or user

Talking about users you may have noticed that we have not properly introduced the concept of players or, in general, users into our system. And this is fine. The management of the actual user's database will be the responsibility of the Identity service, which we will create in a future module, along with all the microservices security infrastructure. For now, we use random GUIDs to identify the users that own the inventory items.

Let's get started.

(Demo prep)

Creating the Inventory microservice

Let's create our Inventory microservice. I'll start by opening a new instance of VS Code.

1. In a Terminal, switch to D:\projects
2. md Play.Inventory
3. CD Play.Inventory
4. code . -r
5. Crate **src** directory
6. Right click on src and **Open Terminal**
7. dotnet new webapi -n Play.Inventory.Service
8. Select any C# file so OmniSharp starts and prompts for adding files for build/run
9. Add to **tasks.json** (type it):

```
"group": {
  "kind": "build",
  "isDefault": true
}
```

10. Remove **serverReadyAction** from launch.json.

11. Update ports in launchSettings.json:

    "applicationUrl": "https://localhost:**5005**;http://localhost:**5004**",

12. Delete WeatherForecastController and WeatherForecast

13. Create Dtos.cs

```
namespace Play.Inventory.Service.Dtos
{
  public record GrantItemsDto(Guid UserId, Guid CatalogItemId, int Quantity);

  public record InventoryItemDto(Guid CatalogItemId, int Quantity, DateTimeOffset AcquiredDate);
}
```

14. dotnet add package Play.Common

15. Add the **Entities** directory

16. Add the InventoryItem entity:

```
namespace Play.Inventory.Service.Entities
{
  public class InventoryItem : IEntity
  {
    public Guid Id { get; set; }

    public Guid UserId { get; set; }

    public Guid CatalogItemId { get; set; }

    public int Quantity { get; set; }

    public DateTimeOffset AcquiredDate { get; set; }
  }
}
```

17. Add Extensions.cs:

```
public static class Extensions
{
    public static InventoryItemDto AsDto(this InventoryItem item)
    {
        return new InventoryItemDto(item.CatalogItemId, item.Quantity, item.AcquiredDate);
    }
}
```

18. Add the ItemsController:

```
namespace Play.Inventory.Service.Controllers
{
    [ApiController]
    [Route("items")]
    public class ItemsController : ControllerBase
    {
        private readonly IRepository<InventoryItem> itemsRepository;

        public ItemsController(IRepository<InventoryItem> itemsRepository)
        {
            this.itemsRepository = itemsRepository;
        }

        [HttpGet]
        public async Task<ActionResult<IEnumerable<InventoryItemDto>>> GetAsync(Guid userId)
        {
            if (userId == Guid.Empty)
            {
                return BadRequest();
            }

            var items = (await itemsRepository.GetAllAsync(item => item.UserId == userId))
                        .Select(item => item.AsDto());

            return Ok(items);
        }


        [HttpPost]
        public async Task<ActionResult> PostAsync(GrantItemsDto grantItemsDto)
        {
            var inventoryItem = await itemsRepository.GetAsync(item => item.UserId == grantItemsDto.UserId &&
item.CatalogItemId == grantItemsDto.CatalogItemId);

            if (inventoryItem == null)
            {
                inventoryItem = new InventoryItem
                {
                    CatalogItemId = grantItemsDto.CatalogItemId,
```

```
                UserId = grantItemsDto.UserId,
                Quantity = grantItemsDto.Quantity,
                AcquiredDate = DateTimeOffset.UtcNow
            };

            await itemsRepository.CreateAsync(inventoryItem);
        }
        else
        {
            inventoryItem.Quantity += grantItemsDto.Quantity;
            await itemsRepository.UpdateAsync(inventoryItem);
        }

        return Ok();
    }
  }
}
```

19. Update appsettings.json:

```
 "ServiceSettings": {
   "ServiceName": "Inventory"
 },
 "MongoDbSettings": {
   "Host": "localhost",
   "Port": 27017
 },
```

20. Update Startup.cs:

```
public void ConfigureServices(IServiceCollection services)
{
   services.AddMongo()
         .AddMongoRepository<InventoryItem>("inventoryitems");

   services.AddControllers();
   services.AddSwaggerGen(c =>
   {
      c.SwaggerDoc("v1", new OpenApiInfo { Title = "Play.Inventory.Service", Version = "v1" });
   });
}
```

21. Import Inventory API in Postman

22. Set collection baseurl as **https://localhost:5005**

23. POST an inventory item:

```
{
  "userId": "{{$guid}}",
  "catalogItemId": "16ab846e-807f-4512-9c0e-3ec0170adc6b",
  "quantity": 1
}
```

24. Query for inventory items (grab user id from Postman Console):

```
GET {{baseUrl}}/items?userId=5420090a-6f7c-4cae-8a70-f6228208bb74
```

25. POST more of the same item:

```
{
  "userId": "{{$guid}}",
  "catalogItemId": "16ab846e-807f-4512-9c0e-3ec0170adc6b",
  "quantity": 4
}
```

26. Query for inventory items again

At this point our Inventory service can store and report the items that each user has in their inventory bag. However, notice that the DTO returned by the GET operation only offers the ids of the items in the user inventory. Without names or descriptions it is hard to tell which items are in the user inventory. In the next lesson we will learn about the microservice communication styles that our Inventory service can use to retrieve more details about each item from our Catalog service.