Implementing synchronous communication

(Demo prep)

Invoking Catalog from Inventory

Our Inventory microservice needs to receive data from the Catalog microservice. But before it can do that it first needs to define the DTO that represents the retrieved catalog items.

1. Add CatalogItemDto:

public record CatalogItemDto(Guid Id, string Name, string Description);

2. Add Clients directory

3. Add CatalogClient

```
public class CatalogClient
{
    private readonly HttpClient httpClient;

    public CatalogClient(HttpClient httpClient)
    {
        this.httpClient = httpClient;
    }

    public async Task<IReadOnlyCollection<CatalogItemDto>> GetCatalogItemsAsync()
    {
        var items = await httpClient.GetFromJsonAsync<IReadOnlyCollection<CatalogItemDto>>($"/items");
        return items;
    }
}
```

4. Update InventoryItemDto:

public record InventoryItemDto(Guid CatalogItemId, **string Name, string Description**, int Quantity, DateTimeOffset AcquiredDate);

5. Update Extensions:

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

```
        }
    }
```

6. Update ItemsController:

```
public class ItemsController : ControllerBase
{
    private readonly IRepository<InventoryItem> itemsRepository;
    private readonly CatalogClient catalogClient;

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

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

        var catalogItems = await catalogClient.GetCatalogItemsAsync();

        var inventoryItemEntities = await itemsRepository.GetAllAsync(item => item.UserId == userId);

        var inventoryItemDtos = inventoryItemEntities.Select(inventoryItem =>
        {
            var catalogItem = catalogItems.Single(catalogItem => catalogItem.Id == inventoryItem.CatalogItemId);
            return inventoryItem.AsDto(catalogItem.Name, catalogItem.Description);
        });

        return Ok(inventoryItemDtos);
    }

    [HttpPost]
    public async Task<ActionResult> PostAsync(GrantItemsDto grantItemsDto)
    {
        …
    }
}
```

7. Add to Startup → ConfigureServices:

```
    services.AddHttpClient<CatalogClient>(client =>
    {
        client.BaseAddress = new Uri("https://localhost:5001");
    });
```

8. Start Catalog and Inventory

9. Query for all user inventory

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

10. Grant another item to user via Postman:

11. Query for user inventory again

This seems to be working well. However, think about what would happen if the Catalog service starts having issues, at least temporarily.

In the next lesson we will learn about the problems that we might find when having a service communicate with another service.