Asynchronous communication

(Demo prep)

- Delete inventoryitems collection
- Delete Play.Common.1.0.2.nupkg

Using the mini catalog items DB

1. Now that our Inventory microservice has its own copy of the catalog items, it no longer needs to query the Catalog microservice to gather additional information about the items on inventory. So, lets update our Inventory controller to take advantage of its new Catalog items collection.

2. Open ItemsController

3. **Collapse the navigation pane**

4. Update ItemsController:

```
public class ItemsController : ControllerBase
{
   private readonly IRepository<InventoryItem> inventoryItemsRepository;
   private readonly IRepository<CatalogItem> catalogItemsRepository;

   public ItemsController(IRepository<InventoryItem> inventoryItemsRepository, IRepository<CatalogItem>
catalogItemsRepository)
   {
      this.inventoryItemsRepository = inventoryItemsRepository;
      this.catalogItemsRepository = catalogItemsRepository;
   }

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

      var inventoryItemEntities = await inventoryItemsRepository.GetAllAsync(item => item.UserId == userId);
      var itemIds = inventoryItemEntities.Select(item => item.CatalogItemId);
      var catalogItemEntities = await catalogItemsRepository.GetAllAsync(item => itemIds.Contains(item.Id));

      var inventoryItemDtos = inventoryItemEntities.Select(inventoryItem =>
      {
         var catalogItem = catalogItemEntities.Single(catalogItem => catalogItem.Id ==
inventoryItem.CatalogItemId);
```

```
            return inventoryItem.AsDto(catalogItem.Name, catalogItem.Description);
        });

        return Ok(inventoryItemDtos);
    }

    …
}
```

5. Start Catalog and Inventory

6. Grant an item to a user in Postman:

POST:

```
{
    "userId": "{{$guid}}",
    "catalogItemId": "11a6bab7-fafd-47f7-b43f-230a8f756e70",
    "quantity": 1
}
```

7. Get all the items in the user inventory in Postman

8. Update the description of the Potion:

PUT:

```
{
    "name": "Potion",
    "description": "Restores some of HP",
    "price": 5
}
```

9. Get all the items in the user inventory in Postman

10. Stop Catalog

11. Query for items in inventory again

12. Grant another item to users inventory:

```
{
    "userId": "671a25b3-40a8-40ae-a3e9-eb9d523add34",
    "catalogItemId": "5e460329-d9a3-41b9-a232-1ba479eaf6d1",
    "quantity": 3
}
```

13. Query for items in inventory again

This is the power of asynchronous communication and this is how you an enable very resilient communication between microservices. Both services are now really autonomous because they don't rely on each other to be available at any given time.

One more thing to consider is the case where one of our consumers is not able to consume the message properly, perhaps due to some issue while talking to its local database.

14. Add message retries to Masstransit configuration:

```
services.AddMassTransit(configure =>
{
   configure.AddConsumers(Assembly.GetEntryAssembly());

   configure.UsingRabbitMq((context, configurator) =>
   {
      var configuration = context.GetService<IConfiguration>();
      var serviceSettings = configuration.GetSection(nameof(ServiceSettings)).Get<ServiceSettings>();
      var rabbitMQSettings = configuration.GetSection(nameof(RabbitMQSettings)).Get<RabbitMQSettings>();
      configurator.Host(rabbitMQSettings.Host);
      configurator.ConfigureEndpoints(context, new
KebabCaseEndpointNameFormatter(serviceSettings.ServiceName, false));
      configurator.UseMessageRetry(retryConfigurator =>
      {
         retryConfigurator.Interval(3, TimeSpan.FromSeconds(5));
      });
   });
});
```

15. dotnet pack -p:PackageVersion=**1.0.2** -o ..\..\..\packages

16. Update package reference in Catalog and Inventory

That marks the end of this lesson and of this module. You now have two independent and resilient microservices that can collaborate asynchronously to manage a catalog of items and to grant and query for items in a user's inventory bag. You also have a reusable common library that any of your future microservices can take advantage of and that you can keep improving according to your needs.

And along the way you have learned how to use multiple microservices patterns and techniques, the .NET platform, VS Code, docker, MongoDB, NuGet, RabbitMQ and MassTransit to quickly and reliably standup new microservices to further expand the capabilities of your system.

I hope you have enjoyed the ride.