Granting and subtracting items via asynchronous messages

Script start

At this point we have a way to grant items to users via the Inventory REST API. However, as we get the service ready to participate in the upcoming purchase state machine, we want to have a way to grant items via asynchronous messages.

So, let's add the required contracts and consumers to enable that.

In Inventory repo

1. Open Terminal in src directory.

2. Create Contracts project:

```
dotnet new classlib -n Play.Inventory.Contracts
```

3. Switch to Play.Inventory.Service and reference contracts project:

```
dotnet add reference ..\Play.Inventory.Contracts\Play.Inventory.Contracts.csproj
```

4. Rename Class1.cs to Contracts.cs

5. Update Contracts.cs:

```
namespace Play.Inventory.Contracts
{
    public record GrantItems(Guid UserId, Guid CatalogItemId, int Quantity, Guid CorrelationId);
    public record InventoryItemsGranted(Guid CorrelationId);
    public record SubtractItems(Guid UserId, Guid CatalogItemId, int Quantity, Guid CorrelationId);
    public record InventoryItemsSubtracted(Guid CorrelationId);
}
```

6. Add GrantItemsConsumer.cs under Consumers:

```
namespace Play.Inventory.Service.Consumers
{
    public class GrantItemsConsumer : IConsumer<GrantItems>
    {
        private readonly IRepository<InventoryItem> inventoryItemsRepository;
        private readonly IRepository<CatalogItem> catalogItemsRepository;

        public GrantItemsConsumer(
```

```
        IRepository<InventoryItem> inventoryRepository,
        IRepository<CatalogItem> catalogRepository)
    {
        this.inventoryItemsRepository = inventoryRepository;
        this.catalogItemsRepository = catalogRepository;
    }

    public async Task Consume(ConsumeContext<GrantItems> context)
    {
        var message = context.Message;

        var item = await catalogItemsRepository.GetAsync(message.CatalogItemId);

        if (item == null)
        {
            throw new UnknownItemException(message.CatalogItemId);
        }

        var inventoryItem = await inventoryItemsRepository.GetAsync(
            item => item.UserId == message.UserId && item.CatalogItemId == message.CatalogItemId);

        if (inventoryItem == null)
        {
            inventoryItem = new InventoryItem
            {
                CatalogItemId = message.CatalogItemId,
                UserId = message.UserId,
                Quantity = message.Quantity,
                AcquiredDate = DateTimeOffset.UtcNow
            };

            await inventoryItemsRepository.CreateAsync(inventoryItem);
        }
        else
        {
            inventoryItem.Quantity += message.Quantity;
            await inventoryItemsRepository.UpdateAsync(inventoryItem);
        }

        await context.Publish(new InventoryItemsGranted(message.CorrelationId));
    }
  }
}
```

7. Create the Exceptions directory

8. Add UnknownItemException.cs:

```
namespace Play.Inventory.Service.Exceptions
{
    [Serializable]
    internal class UnknownItemException : Exception
    {
        public UnknownItemException(Guid catalogItemId) : base($"Unknown item '{catalogItemId}'")
        {
            this.ItemId = catalogItemId;
        }

        public Guid ItemId { get; }
    }
}
```

9. Add SubtractItemsConsumer.cs:

```
namespace Play.Inventory.Service.Consumers
{
    public class SubtractItemsConsumer : IConsumer<SubtractItems>
    {
        private readonly IRepository<InventoryItem> inventoryItemsRepository;
        private readonly IRepository<CatalogItem> catalogItemsRepository;

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

        public async Task Consume(ConsumeContext<SubtractItems> context)
        {
            var message = context.Message;

            var item = await catalogItemsRepository.GetAsync(message.CatalogItemId);

            if (item == null)
            {
                throw new UnknownItemException(message.CatalogItemId);
```

```
        }

        var inventoryItem = await inventoryItemsRepository.GetAsync(
            item => item.UserId == message.UserId && item.CatalogItemId == message.CatalogItemId);

        if (inventoryItem != null)
        {
            inventoryItem.Quantity -= message.Quantity;
            await inventoryItemsRepository.UpdateAsync(inventoryItem);
        }

        await context.Publish(new InventoryItemsSubtracted(message.CorrelationId));
      }
    }
}
```

10. Bump the version of Play.Common in Play.Inventory.Services.csproj

```
  <PackageReference Include="Play.Common" Version="1.0.4" />
```

11. Update Startup.cs:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMongo()
        .AddMongoRepository<InventoryItem>("inventoryitems")
        .AddMongoRepository<CatalogItem>("catalogitems")
        .AddMassTransitWithRabbitMq(retryConfigurator =>
        {
            retryConfigurator.Interval(3, TimeSpan.FromSeconds(5));
            retryConfigurator.Ignore(typeof(UnknownItemException));
        })
        .AddJwtBearerAuthentication();
        …
}
```

12. Switch to Play.Inventory.Contracts in Terminal

13. Create the Play.Inventory.Contracts NuGet package:

```
dotnet pack -o ..\..\..\packages
```

In the next lesson we will also update our Identity microservice to be able to debit gil via asynchronous messages.