Consuming Catalog prices in Trading

Script start

In order for our state machine to be able to calculate the purchase total, it needs to know the prices of the different Catalog items. So, let's update Trading so it can consume the item prices any time an item is created or updated.

In Trading repo

1. dotnet add package Play.Catalog.Contracts

2. Add the Entities directory

3. Add the Consumers directory

4. Open Play.Inventory in VS Code

5. Copy the CatalogItem entity from Inventory to Trading

6. Copy the 3 CatalogItem consumers from Inventory to Trading

7. Fix namespaces in copied classes

8. Update CatalogItem.cs:

```
public class CatalogItem : IEntity
{
    …
    public string Description { get; set; }

    public decimal Price { get; set; }
}
```

9. Update CatalogItemCreatedConsumer:

```
public class CatalogItemCreatedConsumer : IConsumer<CatalogItemCreated>
{
    …
    public async Task Consume(ConsumeContext<CatalogItemCreated> context)
    {
        …
        item = new CatalogItem
        {
```

```
            Id = message.ItemId,
            Name = message.Name,
            Description = message.Description,
            Price = message.Price
        };
        …
    }
}
```

10. Update CatalogItemUpdatedConsumer:

```
public class CatalogItemUpdatedConsumer : IConsumer<CatalogItemUpdated>
{
    …
    public async Task Consume(ConsumeContext<CatalogItemUpdated> context)
    {
        …
        if (item == null)
        {
            item = new CatalogItem
            {
                Id = message.ItemId,
                Name = message.Name,
                Description = message.Description,
                Price = message.Price
            };
            …
        }
        else
        {
            item.Name = message.Name;
            item.Description = message.Description;
            item.Price = message.Price;
            …
        }
    }
}
```

11. Update CatalogItemDeletedConsumer:

```
public class CatalogItemDeletedConsumer : IConsumer<CatalogItemDeleted>
{
    private readonly IRepository<CatalogItem> repository;
```

```
    public CatalogItemDeletedConsumer(IRepository<CatalogItem> repository)
    {
        this.repository = repository;
    }

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

        var item = await repository.GetAsync(message.ItemId);

        if (item == null)
        {
            return;
        }

        await repository.RemoveAsync(message.ItemId);
    }
}
```

12. Update Startup.cs:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMongo()
            .AddMongoRepository<CatalogItem>("catalogitems")
            .AddJwtBearerAuthentication();
    …
}

private void AddMassTransit(IServiceCollection services)
{
    services.AddMassTransit(configure =>
    {
        …
        var rabbitMQSettings = …

        configure.AddConsumers(Assembly.GetEntryAssembly());

        configure.UsingRabbitMq((context, configurator) =>
        {
            …
        });
        …
```

```
    });
    …
}
```

In the next lesson we will update the state machine so it can calculate the purchase total by using the new pricing information that's coming from Catalog.