Consuming Inventory and Identity events in the Trading microservice

Script start

Let's now update our Trading microservice to consume the new events published by Inventory and Identity.

In Trading repo

1. Update the NuGet package references:

```
<ItemGroup>
  …
  <PackageReference Include="Play.Identity.Contracts" Version="1.0.1" />
  <PackageReference Include="Play.Inventory.Contracts" Version="1.0.1" />
  …
</ItemGroup>
```

2. Add the InventoryItem entity:

```
namespace Play.Trading.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; }
    }
}
```

3. Add the ApplicationUser entity:

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

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

```
}
```

4. Add the InventoryItemUpdatedConsumer:

```
namespace Play.Trading.Service.Consumers
{
   public class InventoryItemUpdatedConsumer : IConsumer<InventoryItemUpdated>
   {
      private readonly IRepository<InventoryItem> repository;

      public InventoryItemUpdatedConsumer(IRepository<InventoryItem> repository)
      {
         this.repository = repository;
      }

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

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

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

            await repository.CreateAsync(inventoryItem);
         }
         else
         {
            inventoryItem.Quantity = message.NewTotalQuantity;
            await repository.UpdateAsync(inventoryItem);
         }
      }
   }
}
```

5. Add the UserUpdatedConsumer

```
namespace Play.Trading.Service.Consumers
{
    public class UserUpdatedConsumer : IConsumer<UserUpdated>
    {
        private readonly IRepository<ApplicationUser> repository;

        public UserUpdatedConsumer(IRepository<ApplicationUser> repository)
        {
            this.repository = repository;
        }

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

            var user = await repository.GetAsync(message.UserId);

            if (user == null)
            {
                user = new ApplicationUser
                {
                    Id = message.UserId,
                    Gil = message.NewTotalGil
                };

                await repository.CreateAsync(user);
            }
            else
            {
                user.Gil = message.NewTotalGil;

                await repository.UpdateAsync(user);
            }
        }
    }
}
```

6. Update Startup:

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

```
        .AddMongoRepository<ApplicationUser>("users")
        .AddJwtBearerAuthentication();
        …
}
```

7. Start Inventory, Identity and Trading services

In Postman

8. Perform a purchase

9. Verify the new collections have been created in Trading DB

In the next lesson we will create a new controller that can take advantage of these new database collections to provide data to the new store experience.