Implementing idempotent consumers

Script start

Let's explore the message duplication issue firsthand and then let's use idempotent consumers to properly handle them.

In Inventory repo

1. Update GrantItemsConsumer:

```
public class GrantItemsConsumer : IConsumer<GrantItems>
{
   private static int retryCount = 0;

   …

   public async Task Consume(ConsumeContext<GrantItems> context)
   {
      …

      if (inventoryItem == null)
      {
         …
      }
      else
      {
         …
      }

      if (retryCount < 2)
      {
         retryCount++;
         throw new Exception("Something bad happened");
      }

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

2. Start Trading service

In Postman

3. Confirm the quantity of potions the user has

4. Perform the purchase

5. See how potion quantity increases with each retry

In Inventory repo

6. Update InventoryItem.cs:

```
public class InventoryItem : IEntity
{
   …
   public DateTimeOffset AcquiredDate { get; set; }

   public HashSet<Guid> MessageIds { get; set; } = new();
}
```

7. Update GrantItemsConsumer:

```
public class GrantItemsConsumer : IConsumer<GrantItems>
{
   …
   public async Task Consume(ConsumeContext<GrantItems> context)
   {
       …

      if (inventoryItem == null)
      {
         inventoryItem = new InventoryItem
         {
           …
         };

         inventoryItem.MessageIds.Add(context.MessageId.Value);
         await inventoryItemsRepository.CreateAsync(inventoryItem);
      }
      else
      {
         if (inventoryItem.MessageIds.Contains(context.MessageId.Value))
         {
            await context.Publish(new InventoryItemsGranted(message.CorrelationId));
            return;
```

```
        }

        inventoryItem.Quantity += message.Quantity;
        inventoryItem.MessageIds.Add(context.MessageId.Value);
        await inventoryItemsRepository.UpdateAsync(inventoryItem);
      }
      …
    }
}
```

8. Start Inventory service

In Postman

9. Confirm the quantity of potions the user has

10. Perform the purchase

11. Notice the operation is not retried

12. Confirm the item was granted only once

In Inventory repo

13. Remove the retry block from GrantItemsConsumer:

```
public class GrantItemsConsumer : IConsumer<GrantItems>
{
  private static int retryCount = 0;

  …

  public async Task Consume(ConsumeContext<GrantItems> context)
  {
    …

    if (retryCount < 2)
    {
      retryCount++;
      throw new Exception("Something bad happened");
    }
```

```
    …
  }
}
```

14.

In Identity repo

15. Update ApplicationUser:

```
public class ApplicationUser : MongoIdentityUser<Guid>
{
  public decimal Gil { get; set; }

  public HashSet<Guid> MessageIds { get; set; } = new();
}
```

16. Update DebitGilConsumer:

```
public class DebitGilConsumer : IConsumer<DebitGil>
{
   …

  public async Task Consume(ConsumeContext<DebitGil> context)
  {
    …

    if (user == null)
    {
      …
    }

    if (user.MessageIds.Contains(context.MessageId.Value))
    {
      await context.Publish(new GilDebited(message.CorrelationId));
      return;
    }

    …

    if (user.Gil < 0)
    {
      throw new InsufficientFundsException(message.UserId, message.Gil);
```

```
        }

        user.MessageIds.Add(context.MessageId.Value);

        await userManager.UpdateAsync(user);

        …
    }
}
```

17. Run Identity service

In Postman

18. Confirm how much gil the user has

19. Perform the purchase

20. Confirm how much gil the user has now

In the next module we will prepare our Trading microservice to participate in the new Frontent store experience.