Handling out of order messages in the state machine

Script start

Let's see how to fix our state machine to properly handle the case where it receives messages that are not in the correct order with respect to the current state.

In Trading repo

1. Update PurchaseStateMachine.cs:

```
public class PurchaseStateMachine : MassTransitStateMachine<PurchaseState>
{
    …

    public PurchaseStateMachine()
    {
        …
        ConfigureItemsGranted();
        ConfigureCompleted();
    }

    …

    private void ConfigureAccepted()
    {
        During(Accepted,
            Ignore(PurchaseRequested),
            When(InventoryItemsGranted)
            …
        );
    }

    private void ConfigureItemsGranted()
    {
        During(ItemsGranted,
            Ignore(PurchaseRequested),
            Ignore(InventoryItemsGranted),
            When(GilDebited)
            …
    }

    private void ConfigureCompleted()
```

```
  {
    During(Completed,
      Ignore(PurchaseRequested),
      Ignore(InventoryItemsGranted),
      Ignore(GilDebited)
    );
  }

  …
}
```

2. Start Trading service

In Postman

3. Perform purchase with same idempotencyid

4. Notice no errors in VS Code console

5. Notice Gil has not been debited again

But then, our REST API and our state machine are not the only places where we need to deal with idempotency properly. What about our Inventory and Identity microservices?

In the next lesson we will learn about message duplication and idempotency in the context of message consumers.