Adding a custom state machine activity

Script start

Let’s now update our state machine so that it can calculate the purchase total based on the Catalog prices that it is now consuming.

Now, arguably, pricing calculation would belong to yet another microservice fully specialized in pricing, but for the purposes of saving time, and since we want to focus more on the state machine capabilities, we will do a very simple calculation directly in the Trading microservice.

So, what we can use in this case is what we call a custom state machine activity.

In Trading repo

1. Add the Activities directory

2. Add CalculatePurchaseTotalActivity.cs:

```
public class CalculatePurchaseTotalActivity : Activity<PurchaseState, PurchaseRequested>
{
    private readonly IRepository<CatalogItem> repository;

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

    public void Probe(ProbeContext context)
    {
        // Provides information that could be used during visualization of the activity
        context.CreateScope("calculate-purchase-total");
    }

    public void Accept(StateMachineVisitor visitor)
    {
        visitor.Visit(this);
    }

    public async Task Execute(
        BehaviorContext<PurchaseState, PurchaseRequested> context,
        Behavior<PurchaseState, PurchaseRequested> next)
    {
        var message = context.Data;
```

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

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

            context.Instance.PurchaseTotal = item.Price * message.Quantity;
            context.Instance.LastUpdated = DateTimeOffset.UtcNow;

            await next.Execute(context).ConfigureAwait(false);
        }

        public Task Faulted<TException>(
            BehaviorExceptionContext<PurchaseState, PurchaseRequested, TException> context,
            Behavior<PurchaseState, PurchaseRequested> next) where TException : System.Exception
        {
            return next.Faulted(context);
        }
    }
```

3. Add the Exceptions directory

4. Add UnknownItemException.cs:

```
[Serializable]
internal class UnknownItemException : Exception
{
    public UnknownItemException(Guid itemId): base($"Unknown item {itemId}")
    {
        this.ItemId = itemId;
    }

    public Guid ItemId { get; }
}
```

5. Update PurchaseStateMachine.cs:

```
public class PurchaseStateMachine : MassTransitStateMachine<PurchaseState>
{
    …
    private void ConfigureInitialState()
    {
        Initially(
            When(PurchaseRequested)
                .Then(context =>
                {
                    ...
                })
                .Activity(x => x.OfType<CalculatePurchaseTotalActivity>())
                .TransitionTo(Accepted)
                .Catch<Exception>(ex => ex
                        .Then(context =>
                        {
                            context.Instance.ErrorMessage = context.Exception.Message;
                            context.Instance.LastUpdated = DateTimeOffset.UtcNow;
                        })
                        .TransitionTo(Faulted))
        );
    }
}
```

6. Update Startup.cs:

```
private void AddMassTransit(IServiceCollection services)
{
    services.AddMassTransit(configure =>
    {
        configure.AddConsumers(Assembly.GetEntryAssembly());
        configure.UsingPlayEconomyRabbitMQ(retryConfigurator =>
        {
            retryConfigurator.Interval(3, TimeSpan.FromSeconds(5));
            retryConfigurator.Ignore(typeof(UnknownItemException));
        });
        ...
    });
    ...
}
```

In Postman

7. Perform a purchase

8. Inspect the new properties in the created state in the database

9. Try purchasing an unexisting item

10. Notice the state is now **Faulted**

In the next lesson we will update the state machine so it grants of items to the user.