Debiting gil via asynchronous messages

Script start

Just like we did with the Inventory microservice, this time we want to update the Identity microservice so that gil can be debited via asynchronous messages.

In Identity repo

1. Open Terminal in src directory.

2. Create Contracts project:

```
dotnet new classlib -n Play.Identity.Contracts
```

3. Switch to Play.Inventory.Service and reference contracts project:

```
dotnet add reference ..\Play.Identity.Contracts\Play.Identity.Contracts.csproj
```

4. Rename Class1.cs to Contracts.cs

5. Update Contracts.cs:

```
namespace Play.Identity.Contracts
{
   public record DebitGil(Guid UserId, decimal Gil, Guid CorrelationId);
   public record GilDebited(Guid CorrelationId);
}
```

6. Add the Consumers directory

7. Add DebitGilConsumer.cs:

```
namespace Play.Identity.Service.Consumers
{
   public class DebitGilConsumer : IConsumer<DebitGil>
   {
      private readonly UserManager<ApplicationUser> userManager;

      public DebitGilConsumer(UserManager<ApplicationUser> userManager)
      {
         this.userManager = userManager;
      }
```

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

            var user = await userManager.FindByIdAsync(message.UserId.ToString());

            if (user == null)
            {
                throw new UnknownUserException(message.UserId);
            }

            user.Gil -= message.Gil;

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

            await userManager.UpdateAsync(user);

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

8. Create the Exceptions directory

9. Add UnknownUserException.cs:

```
namespace Play.Identity.Service.Exceptions
{
    [Serializable]
    internal class UnknownUserException : Exception
    {
        public UnknownUserException(Guid userId): base($"Unknown user '{userId}'.")
        {
            this.UserId = userId;
        }

        public Guid UserId { get; }
    }
}
```

10. Add InsuficientFundsException.cs:

```
namespace Play.Identity.Service.Exceptions
{
    [Serializable]
    internal class InsufficientFundsException : Exception
    {
        public InsufficientFundsException(Guid userId, decimal gilToDebit)
        : base($"Not enough gil to debit {gilToDebit} from user '{userId}'.")
        {
            this.UserId = userId;
            this.GilToDebit = gilToDebit;
        }

        public Guid UserId { get; }
        public decimal GilToDebit { get; }
    }
}
```

11. Update appsettings.json:

```
{
  …
  "MongoDbSettings": {
    …
  },
  "RabbitMQSettings": {
    "Host": "localhost"
  },
  "IdentitySettings": {
    …
  },
  …
}
```

12. Update appsettings.Development.json:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Debug",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  …
}
```

13. Bump Play.Common version in Play.Identity.Service.csproj:

```
<PackageReference Include="Play.Common" Version="1.0.4" />
```

14. Update Startup.cs:

```
public void ConfigureServices(IServiceCollection services)
{
    …
    services.Configure<IdentitySettings>(Configuration.GetSection(nameof(IdentitySettings)))
        …
        .AddMongoDbStores<ApplicationUser, ApplicationRole, Guid>
        (
            …
        );

    services.AddMassTransitWithRabbitMq(retryConfigurator =>
    {
        retryConfigurator.Interval(3, TimeSpan.FromSeconds(5));
        retryConfigurator.Ignore(typeof(UnknownUserException));
        retryConfigurator.Ignore(typeof(InsufficientFundsException));
    });

    services.AddIdentityServer(options =>
    {
        …
    })
    …
}
```

15. Switch to Play.Identity.Contracts in Terminal

16. Create the Play.Identity.Contracts NuGet package:

```
dotnet pack -o ..\..\..\packages
```

In the next module we will start building our Trading microservice.