Sales Tax Report

The reference application is mostly order-focused. A lot of what was built was around placing an order. There are some pieces of the backend system, but much of it is left open. As an example, we don't have a call chain to show how much sales tax we have collected.

Step 1. Place an order. To create an order run the website project. The WebStore project is a console application. To place an order you will need to run the following commands. 1, 5, 12, 13, and 16 (in that order). This will place an order and store that order in the database.

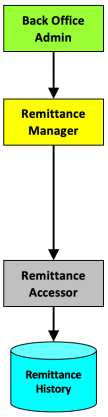
SubTotal	TaxAmount	Total	Auth
3238	226.66	3464.66	615cc6c

This order has a tax amount of 226.66 and occurred in the zip code 68508.

```
select ShippingPostal, TaxAmount, CreatedAt
from Orders
where ShippingPostal = 68508
```

	ShippingPostal	TaxAmount	CreatedAt
1	68508	226.66	2021-06-28 10:10:43.107235-05:00

The call chain we will implement for this activity is as below.



For this activity we will not remit sales tax but build a report that shows how much sales tax we must remit for each zip code. This report should show us the sales tax for a given zip code and date range.

Step 2, create Gherkin scenario for this requirement.

Step 3, create a new Accessor method that will return sales tax given a date range. If we were to write this method as SQL, it might look something like below.

```
select TaxAmount
from Orders
where ShippingPostal = 68508
and CreatedAt >= '2021-06-01' and CreatedAt < '2021-07-01'
```

We will write this method using the entity framework, but the code will end up looking similar.

Find IRemittanceAccessor, we will add a new method to the interface.

```
decimal SalexTax(string zipCode, DateTime start, DateTime end)
```

Navigate to the RemittanceAccessor class, we will need to implement this new method we added.

```
public decimal SalexTax(string zipCode, DateTime start, DateTime end)
{
    using var db = eCommerce.Accessors.EntityFramework.eCommerceDbContext.Create();

    return (YOUR LINQ STATEMENT).Sum();
}
```

Step 4, lets write an automated test. (RemittanceAccessorTests)

```
public void RemittanceAccessor_TaxAmount()
{
    var order = new Order()
    {
        BillingAddress = new Address()
        {
            First = "Bob",
            Last = "Smith",
            EmailAddress = "bob.smith@dontpaniclabsl.com",
            Addr1 = "address1",
            City = "Lincoln",
            State = "Nebraska",
            Postal = "68508",
        },
        ShippingAddress = new Address()
        {
            First = "Bob",
            Last = "Smith",
            EmailAddress = "bob.smith@dontpaniclabsl.com",
            Addr1 = "address1",
            City = "Lincoln",
            State = "Nebraska",
            Postal = "68508",
        },
        OrderLines = new OrderLine[]
        {
            new OrderLine()
            {
```

```
                ProductId = 1,
                UnitPrice = 10.0M,
                ExtendedPrice = 10.0M,
                Quantity = 1,
            },
        },
        TaxAmount = 0.70M,
        Total = 10.0M,
    };

    const int sellerId = 1;
    const int catalogId = 2;

    var orderAccessor = CreateOrderAccessor();
    var remittanceAccessor = CreateRemittanceAccessor();

    var saved = orderAccessor.SaveOrder(catalogId, order);

    var taxAmount = remittanceAccessor.SalexTax("68508", DateTime.Now.AddDays(-30),
DateTime.Now.AddDays(1));
    Assert.IsTrue(taxAmount > 0);
}
```

Step 5. Add a method to MockRemittanceAccessor to return a Sales Tax Amount.

```
public decimal SalexTax(string zipCode, DateTime start, DateTime end)
{
    return 5.00M;
}
```

Step 6. Add a method to IRemittanceManager.

```
decimal RecentSalesTax(string zipCode);
```

Step 7. Add a method to RemittanceManager to call this new method.

```
decimal BackOffice.IBackOfficeRemittanceManager.RecentSalesTax(string zipCode)
{
    if
(UtilityFactory.CreateUtility<ISecurityUtility>().BackOfficeAdminAuthenticated())
    {
        return AccessorFactory.CreateAccessor<IRemittanceAccessor>()
            .SalexTax(zipCode, DateTime.Now.AddDays(-30), DateTime.Now.AddDays(1));
    }

    return 0.0M;
}
```

Step 8. Write a unit test to verify that your RemittanceManager method works. (RemittanceManagerTests)

```
[TestMethod]
[TestCategory("Managers-WebStore")]
public void RemittanceManager_BackOffice_SalesTax()
{
    var mgr = GetBackOfficeManager();

    var tax = mgr.RecentSalesTax("68508");
```

```
    Assert.AreEqual(5.0M, tax);
}
```

Run the new tests and verify that your new code works.

Bonus

Bonus 1: Update the unit tests to actually assert something.

Bonus 2: Create a unit test that produce 0 tax amount but has seed orders.