Sales

View Models

csharp

```csharp
public class CategoryView
{
        public int CategoryID { get; set; }
        public string Description {get; set;}
}
```

csharp

```csharp
public class PartView
{
        public int PartID { get; set; }
        public string Description { get; set; }
        public decimal SellingPrice {get; set;}
}
```

csharp

```csharp
public class SaleView
{
        public int EmployeeId { get; set; }
        public decimal TaxAmount { get; set; }
        public decimal SubTotal { get; set; }
        public int CouponId { get; set; }
        public int DiscountPercent { get; set; }
        public string PaymentType { get; set; }
}
```

csharp

```csharp
public class SaleDetailView
{
        public int PartID { get; set; }
        public string Description { get; set; }
        public int Quantity { get; set; }
        public decimal SellingPrice { get; set; }
```

```
        public decimal Total { get; set; }
}
```

No model class, individual parameters

```csharp
public int GetCoupon(string coupons)
```

Business Processing Requirements

Sales

Query Service Methods

List of categories

```csharp
public List<CategoryView> Sale_GetCategories()
{

}
```

- Retrieve all categories

Get parts

```csharp
public List<PartView> Sale_GetParts(int categoryid)
{

}
```

- show only parts that are not discontinued

Get coupon amount

```csharp
public int GetCoupon(string coupons)
{

}
```

- check that coupon is valid
  - no
    - return zero (0)
  - yes
    - return discount amount

Command Methods

Checkout

```csharp
public int Checkout(SaleView sale, List<SaleDetailView> saleDetails)
{

}
```

- sales details count is greater than zero (0)
  - no
    - throw NullArgumentException (no sales details)
- foreach saleDetail in saleDetails
  - Quantity is greater than zero (0)
    - errorList.Add(new Exception($"Quantity for sale item () must have a positive value"));
- check for errors
  - Yes
    - throw the list of all collected exceptions
  - no
    - save all work to the database
      - create record in Sales
        - set SaleDate to today
        - update all fields

        - ensure that subtotal and tax are discounted based on coupon amount
      - create records in SaleDetails
      - update Parts QuantityOnHand (QuantityOnHand - saleDetail.Quantity)
    - return SaleID
    - **Disable Checkout Button**

Form Methods

Add Part

```csharp
public void AddPart(int partId)
{

}
```

- does part exist in sale details
  - yes
    - display message that part already exist **(Choice 1)**
    - add quantity to sale details part quantity **(Choice 2)**
  - no
    - add part and quantity to sale details
    - update subtotal and tax (including any discount)

Remove Part

```csharp
public void RemovePart(int partId)
{

}
```

- update subtotal and tax (including any discount)

Refresh Part

```csharp
public void RefreshPart(int partId)
{

}
```

- update sale detail total
- update subtotal and tax (including any discount)

Clear

```csharp
public void Clear()
{

}
```

- clear all values

Returns

View Models

```csharp
public class SaleRefundView
{
    public int SaleId { get; set; }
    public int EmployeeId { get; set; }
    public decimal TaxAmount { get; set; }
    public decimal SubTotal { get; set; }
    public int DiscountPercent { get; set; }
}
```

```csharp
public class SaleRefundDetailView
{
    public int PartID { get; set; }
    public string Description { get; set; }
```

```
    public int OrginialQuantity { get; set; }
    public decimal SellingPrice { get; set; }
    public int ReturnQuantity { get; set; }
    public bool Refundable { get; set; }
    public int Quantity { get; set; }
    public string Reason { get; set; }
}
```

Business Processing Requirements

Refund

Query Service Methods

csharp

```
public SaleRefundView SaleRefund_GetSaleRefund(int saleId)
{

}
```

csharp

```
public List<SaleRefundDetailView>
SaleRefund_GetSaleDetailsRefund(int saleId)
{

}
```

Command Methods

csharp

```
public SaleRefundView SaleRefund_Refund(SaleRefundView saleRefund,
List<SaleRefundDetailView>  SaleRefundDetails)
{

}
```

- foreach SaleRefundDetail in SaleRefundDetails

        - Quantity is less than zero (0)
            - errorList.Add(new Exception($"Quantity for item () is less than zero (0)"));
        - Quantity is greater than zero (0) and no reason given
            - errorList.Add(new Exception($"There is no reason given for item ());
        - Quantity return is larger than (original quantity - return quantity)
            - errorList.Add(new Exception($"Quantity for item () is greater than the (original quantity - return quantity)"));
        - Validate that the item can be refundable (Parts.Refundable)
            - no
                - errorList.Add(new Exception($"Item () is not refundable"));
- check for errors
    - Yes
        - throw the list of all collected exceptions
    - no
        - save all work to the database
            - SaleRefund
                - set SaleRefundDate to today
                - ensure that subtotal and tax are discounted based on coupon amount
            - SaleRefundDetails
            - Parts
                - Increase QuantityOnHand = QuantityOnHand + SaleRefundDetail.Quantity
- Return updated saleRefund
- Disable refund button

Form Methods

Lookup Sale

```csharp
public Void SaleRefund_GetRefund(int saleId)
{
    SaleRefund_GetSaleRefund(saleId);
    SaleRefund_GetSaleDetailsRefund(saleId);
}
```

- does sale ID exist

    - no
        - throw NullArgumentException (no sales id found)

Clear

```csharp
public void Clear()
```

- display warning
- clear all values
