70-487

Number: 70-487
Passing Score: 700
Time Limit: 120 min

General

QUESTION 1
The GetVendors() action in the ProcessedOrderController controller is querying the database each time it is run. The GetVendors() action must query the database only if the cache is null.

You need to add code to the action at line PC33 to cache the data. Which code segment can you use? (Each correct answer presents a complete solution. Choose all that apply.)

A. cache.Set(new CacheItem("vendorKey", vendors), GetVendorPolicy());
B. cache.Add("vendors", vendors, new CacheItemPolicy());
C. cache.Add(new CacheItem("vendorKey", vendors) , GetVendorPolicy());
D. cache.AddOrGetExisting("vendorKey", context, new CacheItemPolicy());

Correct Answer: AC
Section: (none)
Explanation

Explanation/Reference:

QUESTION 2
You need to regenerate the service proxies to include task-based asynchronous method signatures. Which command should you use?

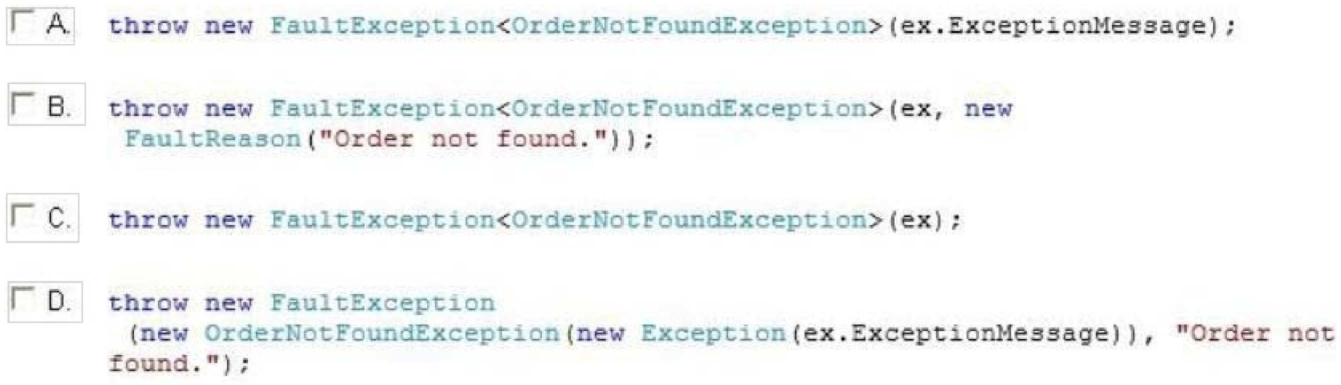
A. aspnet_regiis.exe /t:code http://localhost:62965/UploadCallbackService.svc
B. svcutil.exe /t:code http://localhost:62965/UploadCallbackService.svc
C. aspnet_compiler.exe /t:code http://localhost:62965/UploadCallbackService.svc
D. aspnet_regiis.exe /t:code http://localhost:62965/UploadService.svc
E. svcutil.exe /t:code http://localhost:62965/UploadService.svc

Correct Answer: B
Section: (none)
Explanation

Explanation/Reference:

QUESTION 3
The DeleteExternalOrder() method in the ExternalQueueService service is not throwing a FaultException exception as defined by the FaultContractAttribute attribute in the IExternalQueueService.cs file. You need to throw the FaultException exception. Which code segments can you insert at line EQ45 to achieve this goal? (Each correct answer presents a complete solution. Chose all that apply)

Hot Area:



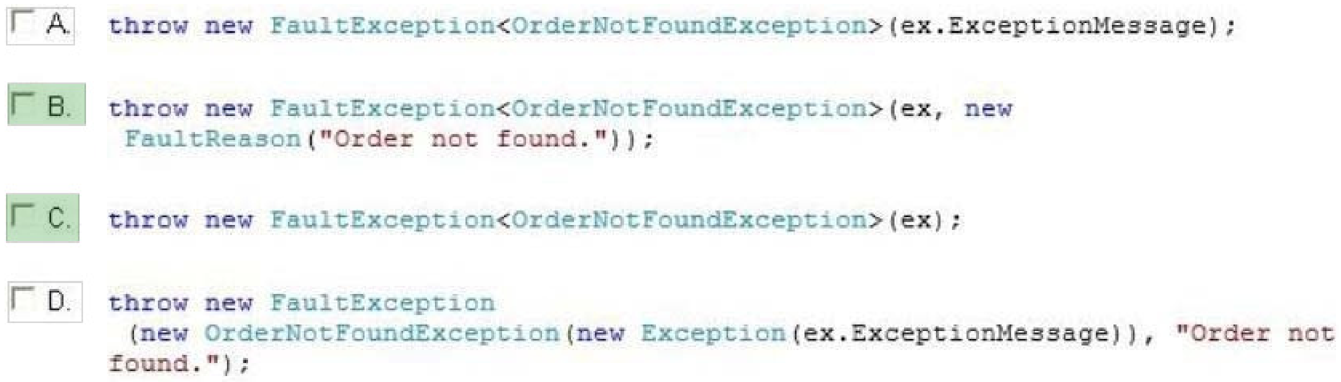
A.
```
throw new FaultException<OrderNotFoundException>(ex.ExceptionMessage);
```

B.
```
throw new FaultException<OrderNotFoundException>(ex, new
 FaultReason("Order not found."));
```

C.
```
throw new FaultException<OrderNotFoundException>(ex);
```

D.
```
throw new FaultException
 (new OrderNotFoundException(new Exception(ex.ExceptionMessage)), "Order not
found.");
```

Correct Answer:



A.
```
throw new FaultException<OrderNotFoundException>(ex.ExceptionMessage);
```

[✓ B.]
```
throw new FaultException<OrderNotFoundException>(ex, new
 FaultReason("Order not found."));
```

[✓ C.]
```
throw new FaultException<OrderNotFoundException>(ex);
```

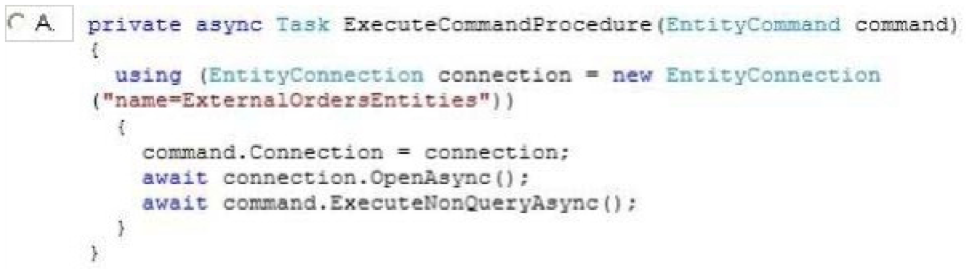
D.
```
throw new FaultException
 (new OrderNotFoundException(new Exception(ex.ExceptionMessage)), "Order not
found.");
```

Section: (none)
Explanation

Explanation/Reference:

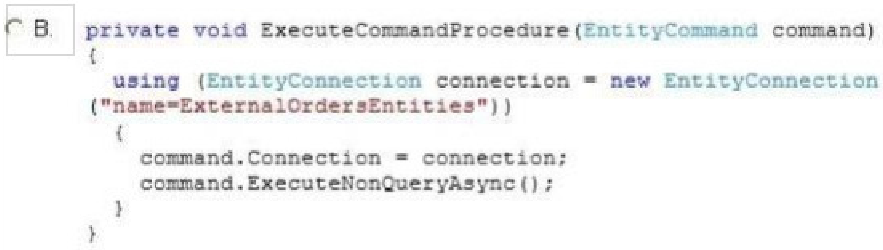
QUESTION 4
You need to modify the ExecuteCommandProcedure() method to meet the technical requirements. Which code segment should you use?

Hot Area:



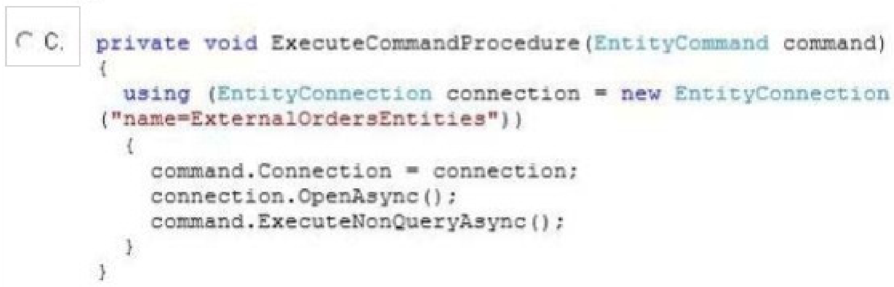
○ A.

```
private async Task ExecuteCommandProcedure(EntityCommand command)
{
  using (EntityConnection connection = new EntityConnection
("name=ExternalOrdersEntities"))
  {
    command.Connection = connection;
    await connection.OpenAsync();
    await command.ExecuteNonQueryAsync();
  }
}
```



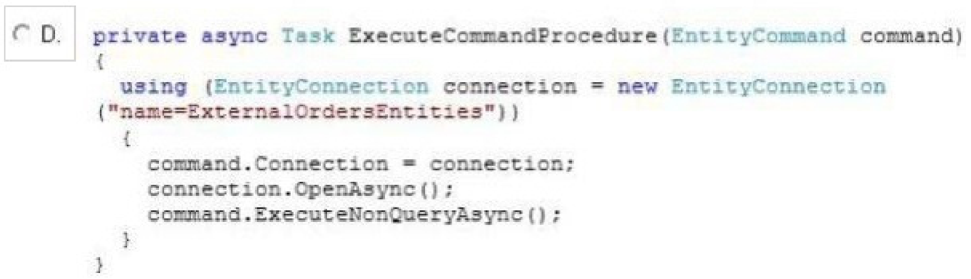
○ B.

```
private void ExecuteCommandProcedure(EntityCommand command)
{
  using (EntityConnection connection = new EntityConnection
("name=ExternalOrdersEntities"))
  {
    command.Connection = connection;
    command.ExecuteNonQueryAsync();
  }
}
```



○ C.

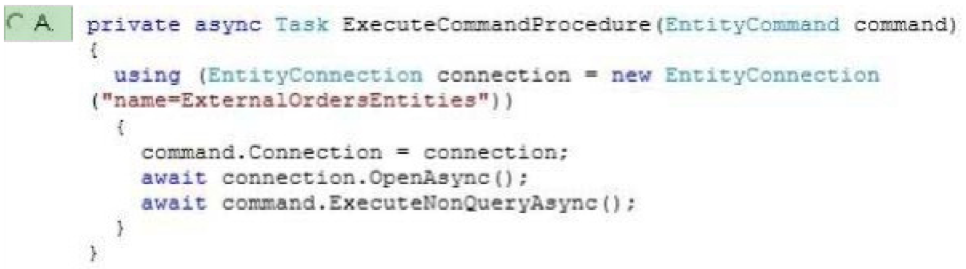
```
private void ExecuteCommandProcedure(EntityCommand command)
{
  using (EntityConnection connection = new EntityConnection
("name=ExternalOrdersEntities"))
  {
    command.Connection = connection;
    connection.OpenAsync();
    command.ExecuteNonQueryAsync();
  }
}
```



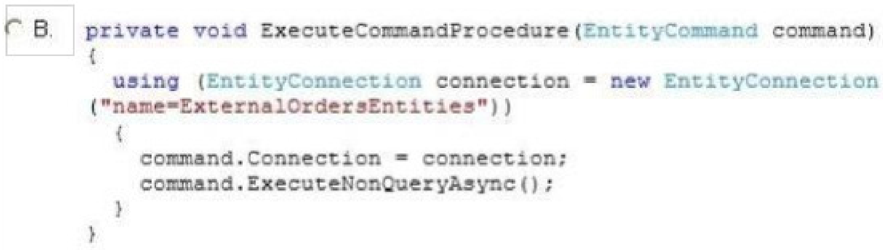
○ D.

```
private async Task ExecuteCommandProcedure(EntityCommand command)
{
  using (EntityConnection connection = new EntityConnection
("name=ExternalOrdersEntities"))
  {
    command.Connection = connection;
    connection.OpenAsync();
    command.ExecuteNonQueryAsync();
  }
}
```

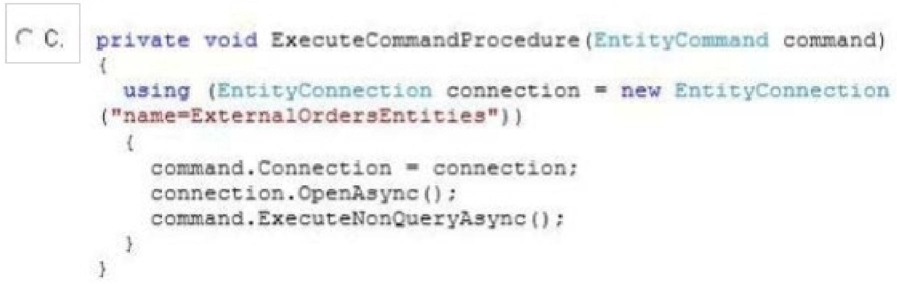
Correct Answer:


[✓ A.]

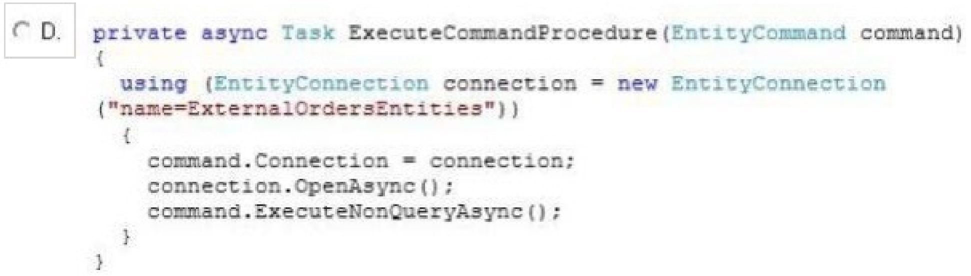
```
private async Task ExecuteCommandProcedure(EntityCommand command)
{
  using (EntityConnection connection = new EntityConnection
("name=ExternalOrdersEntities"))
  {
    command.Connection = connection;
    await connection.OpenAsync();
    await command.ExecuteNonQueryAsync();
  }
}
```


B.

```
private void ExecuteCommandProcedure(EntityCommand command)
{
  using (EntityConnection connection = new EntityConnection
("name=ExternalOrdersEntities"))
  {
    command.Connection = connection;
    command.ExecuteNonQueryAsync();
  }
}
```


C.

```
private void ExecuteCommandProcedure(EntityCommand command)
{
  using (EntityConnection connection = new EntityConnection
("name=ExternalOrdersEntities"))
  {
    command.Connection = connection;
    connection.OpenAsync();
    command.ExecuteNonQueryAsync();
  }
}
```


D.

```
private async Task ExecuteCommandProcedure(EntityCommand command)
{
  using (EntityConnection connection = new EntityConnection
("name=ExternalOrdersEntities"))
  {
    command.Connection = connection;
    connection.OpenAsync();
    command.ExecuteNonQueryAsync();
  }
}
```

Section: (none)
Explanation

Explanation/Reference:

QUESTION 5
Case Study: 2

Scenario 2

Background

You are developing an ASP.NET MVC application in Visual Studio 2012 that will be used to process orders.

Business Requirements

The application contains the following three pages.

· A page that queries an external database for orders that are ready to be processed. The user can then process the order.

· A page to view processed orders.

· A page to view vendor information.

The application consumes three WCF services to retrieve external data.

Technical Requirements

Visual Studio Solution:

The solution contains the following four projects.

· ExternalQueue: A WCF service project used to communicate with the external order database.

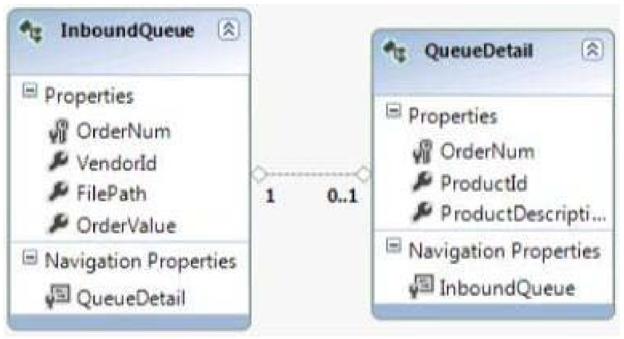
· OrderProcessor: An ASP.NET MVC project used for order processing and logging order metadata.

· OrderUpload: A WCF service project used to submit order data to an external data source.

· Shipping: A WCF service project used to acquire shipping information.

ExternalQueue Project:

Entity Framework is used for data access. The entities are defined in the ExternalOrders.edmx file as shown in the following diagram.



The project contains two services defined in the following files.

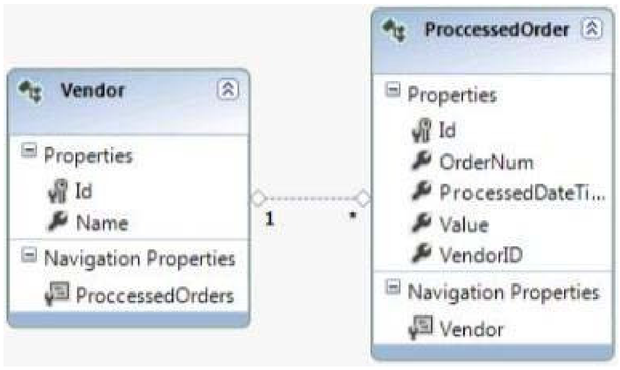
· IExternalQueueService.cs

· ExternalQueueService.svc.

The ExternalQueue.Helpers namespace contains a definition for a class named OrderNotFound Exception.

OrderProcessor Project:

Entity Framework is used for data access. The entities are defined in the ProcessedOrders.edmx file as shown in the following diagram.



The classes are contained in the OrderProcessor.Entities namespace. The project contains the following two controllers.

· InboundQueueController.cs

· ProcessedOrderController.cs

WCF service proxies to the ExternalQueue, Shipping and OrderUpload services have been generated by using the command prompt. The ExecuteCommandProcedure() method in the ExternalQueueService.svc file must run asynchronously.

The ProcessedOrderController controller has the following requirements.

The GetVendorPolicy() method must enforce a 10 minute absolute cache expiration policy.

The GetProcessedOrders() method must return a view of the 10 most recently processed orders.

OrderUpload Project:

The project contains two services defined in the following files.

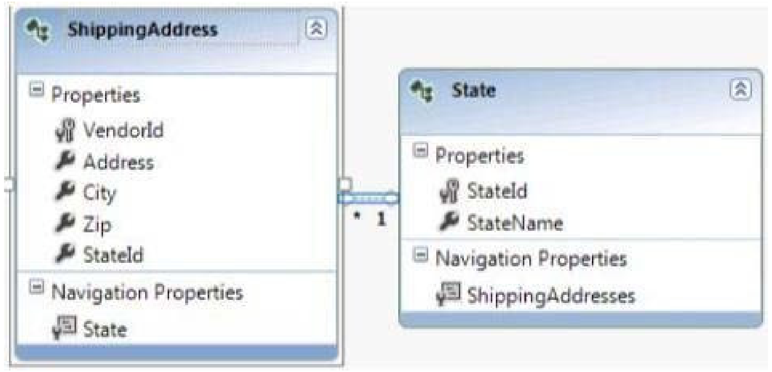
· IUploadCallbackService.cs

· UploadCallbackService.svc

Data Access is maintained in a file named UploadOrder.es.

Shipping Project:

Entity Framework is used for data access. The entities are defined in the ExternalOrders.edmx file as shown in the following diagram.



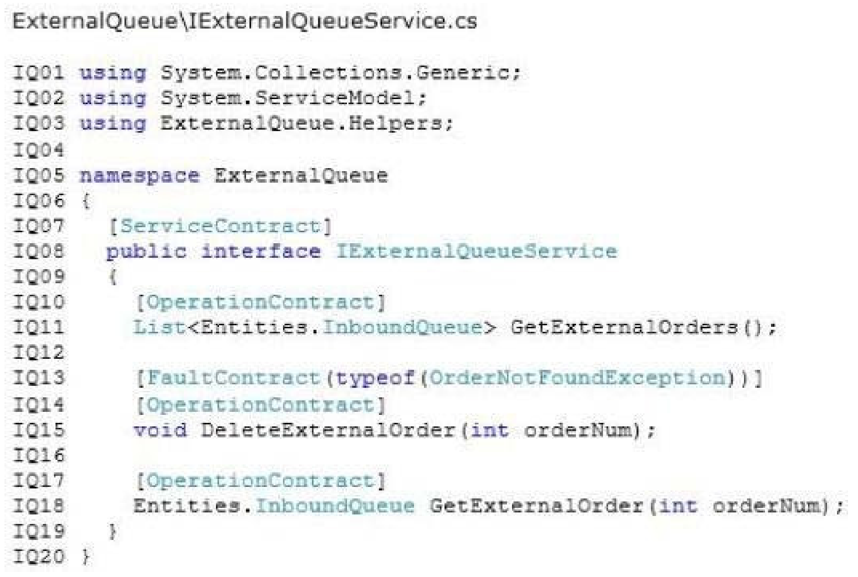
The Custom Tool property for ExternalOrders.edmx has been removed. POCO classes for the Entity Model are located in the ShippingAddress.cs file. The POCO entity must be loaded by using lazy loading. The project contains two services defined in the following files.

· IShippingService.cs

· ShippingService.svc.

The IShippingService contract must contain an operation that receives an order number as a parameter. The operation must return a class named ShippingInfo that inherits from a class named State.

Application Structure



ExternalQueue\IExternalQueueService.cs

```
IQ01 using System.Collections.Generic;
IQ02 using System.ServiceModel;
IQ03 using ExternalQueue.Helpers;
IQ04
IQ05 namespace ExternalQueue
IQ06 {
IQ07   [ServiceContract]
IQ08   public interface IExternalQueueService
IQ09   {
IQ10     [OperationContract]
IQ11     List<Entities.InboundQueue> GetExternalOrders();
IQ12
IQ13     [FaultContract(typeof(OrderNotFoundException))]
IQ14     [OperationContract]
IQ15     void DeleteExternalOrder(int orderNum);
IQ16
IQ17     [OperationContract]
IQ18     Entities.InboundQueue GetExternalOrder(int orderNum);
IQ19   }
IQ20 }
```

OrderProcessor\IExternalQueueService.svc

```
EQ01 using System;
EQ02 using System.Collections.Generic;
EQ03 using System.Linq;
EQ04 using System.Data.EntityClient;
EQ05 using System.Data;
EQ06 using ExternalQueue.Entities;
EQ07 using System.Data.Objects;
EQ08 using ExternalQueue.Helpers;
EQ09 using System.ServiceModel;
EQ10 using System.Threading.Tasks;
EQ11
EQ12 namespace ExternalQueue
EQ13 {
EQ14   public class ExternalQueueService : IExternalQueueService
EQ15   {
EQ16     public List<Entities.InboundQueue> GetExternalOrders()
EQ17     {
EQ18       List<InboundQueue> queueItems = new List<InboundQueue>();
EQ19       return queueItems;
EQ20     }
EQ21
EQ22     public void DeleteExternalOrder(int orderNum)
EQ23     {
EQ24       using (var context = new ExternalOrdersEntities())
EQ25       {
EQ26         var orders = context.InboundQueues.Where(i => i.OrderNum ==
orderNum).ToList();
EQ27         if (orders.Count() > 0)
EQ28         {
EQ29           using (EntityCommand cmd = new EntityCommand())
EQ30           {
EQ31             cmd.CommandText = "ExternalOrdersEntities.uspInboundQueueDelete";
EQ32             cmd.CommandType = CommandType.StoredProcedure;
EQ33             EntityParameter param = new EntityParameter();
EQ34             param.Value = orderNum;
EQ35             param.ParameterName = "orderNum";
EQ36             cmd.Parameters.Add(param);
EQ37             ExecuteCommandProcedure(cmd);
EQ38           }
EQ39         }
EQ40         else
EQ41         {
EQ42           OrderNotFoundException ex = new OrderNotFoundException();
EQ43           ex.OrderNum = orderNum;
EQ44           ex.ExceptionMessage = "Order not found...Cannot delete";
EQ45
EQ46         }
EQ47       }
EQ48     }
EQ49
EQ50     private void ExecuteCommandProcedure(EntityCommand command)
EQ51     {
EQ52       using (EntityConnection connection = new EntityConnection
("name=ExternalOrdersEntities"))
EQ53       {
EQ54         command.Connection = connection;
EQ55         connection.Open();
EQ56         command.ExecuteNonQuery();
EQ57       }
EQ58     }
EQ59
EQ60     public InboundQueue GetExternalOrder(int orderNum)
EQ61     {
EQ62       using (var context = new ExternalOrdersEntities())
EQ63       {
EQ64         string queryString = string.Empty;
EQ65         ObjectQuery<InboundQueue> query = context.CreateQuery<InboundQueue>
EQ66           (queryString, new ObjectParameter("orderNum", orderNum));
EQ67         return query.First();
```

ExternalQueue\ProcessedOrderController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Caching;
using System.Web.Mvc;
using OrderProcessor.Entities;
using OrderProcessor.Helpers;
using System.Configuration;

namespace OrderProcessor.Controllers
{
  public class ProcessedOrderController : Controller
  {
    public ActionResult GetProcessedOrders()
    {
      using (var context = new ProcessedOrders())
      {
        List<Entities.ProccessedOrder> orders = new List<ProccessedOrder>();
        return View(orders);
      }
    }

    private ObjectCache cache {get { return MemoryCache.Default; }}

    public ActionResult GetVendors()
     {
       List<Entities.Vendor> vendors = cache.Get
("vendorKey") as List<Entities.Vendor>;
       if (vendors == null)
       {
         using (var context = new ProcessedOrders())
         {
           vendors = context.Vendors.ToList();

         }
       }
       return View(vendors);
     }

     private CacheItemPolicy GetVendorPolicy()
     {
       CacheItemPolicy vendorPolicy = new CacheItemPolicy();

       return vendorPolicy;
     }

     private List<string> GetTriggerPaths()
     {
       List<string> triggerPath = new List<string>();
       triggerPath.Add(@"c:\triggers\vendortrigger.txt");
       return triggerPath;
     }
  }
}
```