Build an ASP.NET Core Service, and App with Core 2.2 Two-Day Hand-On Lab

Lab 9

This lab is the fourth in a series that builds the SpyStore RESTful service. This lab creates and configures the controllers for the service. Prior to starting this lab, you must have completed Lab 8.

Part 1: Update the ValuesController

Data can be returned from an action controller several ways.

1) Open the ValuesController.cs class, and add the following action methods to class:

```
[HttpGet("test1")]
public IActionResult Get1()
{
  return Ok(new string[] { "value1", "value2" });
}
[HttpGet("test2")]
public string[] Get2()
{
  return new string[] { "value1", "value2" };
}
[HttpGet("test3")]
public IActionResult Get3()
{
  return new JsonResult(new string[] { "value1", "value2" });
}
```

Part 2: Create the Controllers

Step 1: Add the Controllers

2) Add eight new controller classes into the `Controllers` directory of the `SpyStore.Service` project:

```
CategoryController.cs
CustomerController.cs
OrderDetailsController.cs
OrdersController.cs
ProductController.cs
SearchController.cs
ShoppingCartController.cs
ShoppingCartRecordController.cs
```

Step 2: Inherit from ControllerBase, Add Routing and the ApiController Attribute

1) For each of the controllers just added, ensure they are public, inherit from ControllerBase, add the ApiController attribute, and add the attribute route "api/[controller]". Here is the CategoryController as an example:

```
[Route("api/[controller]")]
[ApiController]
public class CategoryController : ControllerBase
{
}
```

Part 3: Update the Controllers

Step 1: Update the CategoryController

1) Update the using statements to the following:

```
using System.Collections.Generic;
using System.Linq;
using Microsoft.AspNetCore.Mvc;
using SpyStore.Hol.Dal.Repos.Interfaces;
using SpyStore.Hol.Models.Entities;
```

2) Add a constructor that takes an instance of `ICategoryRepo` and a private variable to hold the instance. This will be automatically populated by the DI container. The StoreContext needed by the repo is also automatically populated by the DI container.

```
private readonly ICategoryRepo _repo;
public CategoryController(ICategoryRepo repo)
{
  _repo = repo;
}
```

3) Create a method named `Get` that returns all of the Category records. Name the route `GetAllCategories`, and add attributes instructing the framework that the method produces content of "application/json" and potential response codes of 200 and 500:

```
[HttpGet(Name="GetAllCategories")]
[Produces("application/json")]
[ProducesResponseType(200)]
[ProducesResponseType(500)]
public ActionResult<IList<Category>> Get()
{
  IEnumerable<Category> categories = _repo.GetAll().ToList();
  return Ok(categories);
}
```

4) Create a method named `Get` that returns a single Category record. Update the route to include a token for the `Category Id` and name of `GetCategory`, then add the attributes for content type and response codes:

```
[HttpGet("{id}",Name = "GetCategory")]
[Produces("application/json")]
[ProducesResponseType(200)]
[ProducesResponseType(404)]
[ProducesResponseType(500)]
public ActionResult<Category> Get(int id)
{
  Category item = _repo.Find(id);
  if (item == null)
  {
    return NotFound();
  }
  return Ok(item);
}
```

5) Create a method named `GetProductsForCategory` that returns a list of Products for a given Category. The parameters include an instance of the `IProductRepo` (from the DI container) and the Category Id (from the route). When leveraging the DI container in a method (instead of the constructor), you use the `FromServices` attribute:

```
[HttpGet("{categoryId}/products", Name="GetCategoryProducts")]
[Produces("application/json")]
[ProducesResponseType(200)]
[ProducesResponseType(500)]
public ActionResult<IList<Product>> GetProductsForCategory(
    [FromServices] IProductRepo productRepo, int categoryId)
    => productRepo.GetProductsForCategory(categoryId).ToList();
```

Step 2: Update the CustomerController

1) Update the using statements to the following:

```
using System.Collections.Generic;
using System.Linq;
using Microsoft.AspNetCore.Mvc;
using SpyStore.Hol.Dal.Repos.Interfaces;
using SpyStore.Hol.Models.Entities;
```

2) Add a constructor that takes an instance of `ICustomerRepo` and a private variable to hold the instance.

```
private readonly ICustomerRepo _repo;
public CustomerController(ICustomerRepo repo)
{
  _repo = repo;
}
```

3) Create a method named `Get` that returns all of the Customer records. Name the route `GetAllCustomers`, and add the attributes for content type and response codes:

```
[HttpGet(Name = "GetAllCustomers")]
[Produces("application/json")]
[ProducesResponseType(200)]
[ProducesResponseType(500)]
public ActionResult<IEnumerable<Customer>> Get() => Ok(_repo.GetAll().ToList());
```

4) Create a method named `Get` that returns a single Customer record. Update the route to include a token for the `Customer Id` and name of `GetCustomer`, then add the attributes for content type and response codes:

```
[HttpGet("{id}", Name = "GetCustomer")]
[Produces("application/json")]
[ProducesResponseType(200)]
[ProducesResponseType(404)]
[ProducesResponseType(500)]
public ActionResult<Customer> Get(int id)
{
  var item = _repo.Find(id);
  if (item == null)
  {
    return NotFound();
  }
  return Ok(item);
}
```

Step 3: Update the OrderDetailsController

1) Update the using statements to the following:

```
using Microsoft.AspNetCore.Mvc;
using SpyStore.Hol.Dal.Repos.Interfaces;
using SpyStore.Hol.Models.ViewModels;
```

2) Add a constructor that takes an instance of `IOrderRepo` and a private variable to hold the instance.

```
private readonly IOrderRepo _repo;
public OrderDetailsController(IOrderRepo repo)
{
  _repo = repo;
}
```

3) Create a method named `GetOrderWithDetailsForCustomer` that returns all of the `OrderWithDetailAndProductInfo` records for a `Customer`. Add a token for the `Order Id` and name the route `GetOrderDetails`, and add the attributes for content type and response codes. Note that this method can also return a 404 (Not Found):

```
[HttpGet("{orderId}", Name = "GetOrderDetails")]
[Produces("application/json")]
[ProducesResponseType(200)]
[ProducesResponseType(404)]
[ProducesResponseType(500)]
public IActionResult GetOrderWithDetailsForCustomer(int orderId)
{
  OrderWithDetailsAndProductInfo orderWithDetails = _repo.GetOneWithDetails(orderId);
  return orderWithDetails == null
    ? (IActionResult) NotFound()
    : new ObjectResult(orderWithDetails);
}
```

Step 4: Update the OrdersController

1) Update the using statements to the following:

```
using System.Collections.Generic;
using Microsoft.AspNetCore.Mvc;
using SpyStore.Hol.Dal.Repos.Interfaces;
using SpyStore.Hol.Models.Entities;
```

2) Add a constructor that takes an instance of `IOrderRepo` and a private variable to hold the instance.

```
private readonly IOrderRepo _repo;
public OrdersController(IOrderRepo repo)
{
  _repo = repo;
}
```

3) Create a method named `GetOrderHistory` that returns all of the `Order` records for a `Customer`. Add a token for the `Order Id` and name the route `GetOrderHistory`, and add the attributes for content type and response codes:

```
[HttpGet("{customerId}", Name = "GetOrderHistory")]
[Produces("application/json")]
[ProducesResponseType(200)]
[ProducesResponseType(404)]
[ProducesResponseType(500)]
public IActionResult GetOrderHistory(int customerId)
{
  _repo.Context.CustomerId = customerId;
  IList<Order> orderWithTotals = _repo.GetOrderHistory();
  return orderWithTotals == null
    ? (IActionResult) NotFound()
    : new ObjectResult(orderWithTotals);
}
```

Step 5: Update the ProductController

1) Update the using statements to the following:

```
using System.Collections.Generic;
using System.Linq;
using Microsoft.AspNetCore.Mvc;
using SpyStore.Hol.Dal.Repos.Interfaces;
using SpyStore.Hol.Models.Entities;
```

2) Add a constructor that takes an instance of `IProductRepo` and a private variable to hold the instance.

```
private readonly IProductRepo _repo;
public ProductController(IProductRepo repo)
{
  _repo = repo;
}
```

3) Create a method named `Get` that returns a single Product record. Update the route to include a token for the `Product Id` and name of `GetProduct`, then add the attributes for content type and response codes:

```
[HttpGet("{id}", Name = "GetProduct")]
[Produces("application/json")]
[ProducesResponseType(200)]
[ProducesResponseType(404)]
[ProducesResponseType(500)]
public ActionResult<Product> Get(int id)
{
  Product item = _repo.GetOneWithCategoryName(id);
  if (item == null)
  {
    return NotFound();
  }
  return Ok(item);
}
```

4) Create a method named `GetFeatured` that returns a list of the featured Products.

```
[HttpGet("featured", Name = "GetFeaturedProducts")]
[Produces("application/json")]
[ProducesResponseType(200)]
[ProducesResponseType(500)]
public ActionResult<IList<Product>> GetFeatured()
  => Ok(_repo.GetFeaturedWithCategoryName().ToList());
```

Step 6: Update the SearchController

1) Update the using statements to the following:

```
using System.Collections.Generic;
using System.Linq;
using Microsoft.AspNetCore.Mvc;
using SpyStore.Hol.Dal.Repos.Interfaces;
using SpyStore.Hol.Models.Entities;
```

2) Add a constructor that takes an instance of `IProductRepo` and a private variable to hold the instance.

```
private readonly IProductRepo _repo;
public SearchController(IProductRepo repo)
{
  _repo = repo;
}
```

3) Create a method named `Search` that returns the search results. Update the route to include a token for the `searchstring` and name of `SearchProduct`, then add the attributes for content type and response codes:

```
[HttpGet("{searchString}", Name = "SearchProducts")]
[Produces("application/json")]
[ProducesResponseType(200)]
[ProducesResponseType(204)]
[ProducesResponseType(500)]
public ActionResult<IList<Product>> Search(string searchString)
{
  var results = _repo.Search(searchString).ToList();
  if (results.Count == 0)
  {
    return NoContent();
  }
  return results;
}
```

Step 7: Update the ShoppingCartController

The ShoppingCartController has one Post method, used to purchase the items in the cart.

1) Update the using statements to the following:

```
using Microsoft.AspNetCore.Mvc;
using SpyStore.Hol.Dal.Repos.Interfaces;
using SpyStore.Hol.Models.Entities;
using SpyStore.Hol.Models.ViewModels;
```

2) Add a constructor that takes an instance of `IProductRepo` and a private variable to hold the instance.

```
private readonly IShoppingCartRepo _repo;
public ShoppingCartController(IShoppingCartRepo repo)
{
  _repo = repo;
}
```

3) Create a method named `GetShoppingCart` that returns the Customer information and ShoppingCart records for a customer. Update the route with the name of `GetShoppingCart`, then add the attributes for content type and response codes:

```
[HttpGet(Name = "GetShoppingCart")]
[Produces("application/json")]
[ProducesResponseType(200)]
[ProducesResponseType(204)]
[ProducesResponseType(500)]
public ActionResult<CartWithCustomerInfo> GetShoppingCart(int customerId)
  => _repo.GetShoppingCartRecordsWithCustomer(customerId);
```

4) Create a method named `Purchase` that processes the pretend purchase process. It returns an HTTP 201 when successful (referring to the created Order and Order Details). returns the Customer information and ShoppingCart records for a customer. Update the route with the name of `GetShoppingCart`, then add the attributes for content type and response codes:

```
[HttpPost("buy", Name = "Purchase")]
[ProducesResponseType(201)]
[ProducesResponseType(500)]
public IActionResult Purchase(int customerId, Customer customer)
{
  if (customer == null || customer.Id != customerId || !ModelState.IsValid)
  {
    return BadRequest();
  }
  int orderId;
  orderId = _repo.Purchase(customerId);
  //Location: http://localhost:8477/api/OrderDetails/2
  return CreatedAtRoute("GetOrderDetails",
    routeValues: new {orderId = orderId},null);
}
```

Step 8: Update the ShoppingCartRecordController

The ShoppingCartRecordController Has HttpGet, HttpPist, HttpPut, and HttpDelete methods.

1) Update the using statements to the following:

```
using System;
using Microsoft.AspNetCore.Mvc;
using SpyStore.Hol.Dal.Repos.Interfaces;
using SpyStore.Hol.Models.Entities;
using SpyStore.Hol.Models.ViewModels;
```

2) Add a constructor that takes an instance of `IProductRepo` and a private variable to hold the instance.

```
private readonly IShoppingCartRepo _repo;
public ShoppingCartRecordController(IShoppingCartRepo repo)
{
  _repo = repo;
}
```

3) Create a method named `GetShoppingCartRecord` that returns a CartRecordWithProductInfo view model. Update the route with the name of `GetShoppingCart`, then add the attributes for content type and response codes:

```
[HttpGet("{recordId}",Name = "GetShoppingCartRecord")]
[Produces("application/json")]
[ProducesResponseType(200)]
[ProducesResponseType(404)]
[ProducesResponseType(500)]
public ActionResult<CartRecordWithProductInfo> GetShoppingCartRecord(int recordId)
{
  CartRecordWithProductInfo cartRecordWithProductInfo = _repo.GetShoppingCartRecord(recordId);
  return cartRecordWithProductInfo ?? (ActionResult<CartRecordWithProductInfo>) NotFound();
}
```

4) Create a method named `AddShoppingCartRecord` that returns an HTTP 201 (created). Update the route to take a token for the `customerId` and route name of `AddCartRecord`, then add the attributes for response codes:

```
[HttpPost("{customerId}",Name = "AddCartRecord")]
[ProducesResponseType(201)]
[ProducesResponseType(400)]
[ProducesResponseType(500)]
public ActionResult AddShoppingCartRecord(int customerId, ShoppingCartRecord record)
{
  if (record == null || customerId != record.CustomerId || !ModelState.IsValid)
  {
    return BadRequest();
  }
  record.DateCreated = DateTime.Now;
  record.CustomerId = customerId;
  _repo.Context.CustomerId = customerId;
  _repo.Add(record);
  //Location: http://localhost:8477/api/ShoppingCartRecord/1 (201)
  CreatedAtRouteResult createdAtRouteResult = CreatedAtRoute("GetShoppingCart",
    new {controller = "ShoppingCart", customerId = customerId },
    null);
  return createdAtRouteResult;
    }
```

5) Create a method named `UpdateShoppingCartRecord` that returns an HTTP 201 (created). Update the route to take a token for the `recordId` and route name of `UpdateCartRecord`, then add the attributes for response codes:

```
[HttpPut("{recordId}", Name = "UpdateCartRecord")]
[ProducesResponseType(201)]
[ProducesResponseType(400)]
[ProducesResponseType(500)]
public ActionResult UpdateShoppingCartRecord(int recordId, ShoppingCartRecord item)
{
  if (item == null || item.Id != recordId || !ModelState.IsValid)
  {
    return BadRequest();
  }
  item.DateCreated = DateTime.Now;
  _repo.Context.CustomerId = item.CustomerId;
  _repo.Update(item);
  //Location: http://localhost:8477/api/ShoppingCartRecord/0 (201)
  return CreatedAtRoute("GetShoppingCartRecord",
     new {controller = "ShoppingCartRecord", recordId = item.Id},
     null);
}
```

6) Create a method named `DeleteCartRecord` that returns an HTTP 201 (created). Update the route to take a token for the `recordId` and route name of `UpdateCartRecord`, then add the attributes for response codes:

```
[HttpDelete("{recordId}", Name = "DeleteCartRecord")]
[ProducesResponseType(201)]
[ProducesResponseType(400)]
[ProducesResponseType(500)]
public IActionResult DeleteCartRecord(int recordId, ShoppingCartRecord item)
```

```
{
  if (recordId != item.Id)
  {
    return NotFound();
  }
  _repo.Context.CustomerId = item.CustomerId;
  _repo.Delete(item);
  return NoContent();
}
```

Summary

This lab created and configured the Controllers for the service.

Next steps

In the next part of this tutorial series, you will add Docker support and a Docker-Compose orchestration.