Build an ASP.NET Core Service and App with .NET (Core) 5.0 Two-Day Hands-On Lab

Lab 10

This lab creates and configures the controllers for the RESTful service. Prior to starting this lab, you must have completed Lab 9. This entire lab works on the `AutoLot.Api` project.

Part 1: The BaseController

Step 1: Initial File and Constructor Code

- Create a new folder named `Base` under the `Controllers` folder. Add a new class file named `BaseCrudController.cs` to the folder. In that file, add the following namespaces:

```
using System;
using System.Collections.Generic;
using AutoLot.Dal.Exceptions;
using AutoLot.Models.Entities.Base;
using AutoLot.Dal.Repos.Base;
using AutoLot.Services.Logging;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
```

- Make the class public and abstract and generic, accepting a `BaseEntity` type and a Controller that inherits from `BaseCrudController`:

```
namespace AutoLot.Api.Controllers.Base
{
  public abstract class BaseCrudController<T, TController> : ControllerBase
    where T : BaseEntity, new()
    where TController : BaseCrudController<T, TController>
  {
  }
}
```

- Add the `ApiController` attribute to the class to opt-in to the API benefits:

```
[ApiController]
public abstract class BaseCrudController<T, TController> : ControllerBase
  where T : BaseEntity, new()
  where TController : BaseCrudController<T, TController>
{
}
```

- Inject in an instance of `IRepo<T>` and `IAppLogging<TController>` and assign them to protected `readonly` variables:

```
protected readonly IRepo<T> MainRepo;
protected readonly IAppLogging<TController> Logger;

protected BaseCrudController(IRepo<T> repo, IAppLogging<TController> logger)
{
  MainRepo = repo;
  Logger = logger;
}
```

Step 2: Add the Get Methods

- There are two base methods to get records – `GetAll` and `GetOne`:

```
[HttpGet]
public ActionResult<IEnumerable<T>> GetAll()
{
  return Ok(MainRepo.GetAllIgnoreQueryFilters());
}

[HttpGet("{id}")]
public ActionResult<T> GetOne(int id)
{
  var entity = MainRepo.Find(id);
  if (entity == null)
  {
    return NoContent();
  }
  return Ok(entity);
}
```

Step 3: Add the Add Method

```
[HttpPost]
public ActionResult<T> AddOne(T entity)
{
  try
  {
    MainRepo.Add(entity);
  }
  catch (Exception ex)
  {
    return BadRequest(ex);
  }
  return CreatedAtAction(nameof(GetOne), new {id = entity.Id}, entity);
}
```

Step 4: Add the Update Method

```
[HttpPut("{id}")]
public IActionResult UpdateOne(int id,[FromBody]T entity)
{
  if (id != entity.Id) { return BadRequest();}
  try
  {
    MainRepo.Update(entity);
  }
  catch (CustomException ex)
  {
    //This shows an example with the custom exception
    //Should handle more gracefully
    return BadRequest(ex);
  }
  catch (Exception ex)
  {
    //Should handle more gracefully
    return BadRequest(ex);
  }
  return Ok(entity);
}
```

Step 5: Add the Delete Method

```
[HttpDelete("{id}")]
public ActionResult<T> DeleteOne(int id, T entity)
{
  if (id != entity.Id) { return BadRequest();}
  try
  {
    MainRepo.Delete(entity);
  }
  catch (Exception ex)
  {
    //Should handle more gracefully
    return new BadRequestObjectResult(ex.GetBaseException()?.Message);
  }

  return Ok();
    }
```

All files copyright Phil Japikse (http://www.skimedic.com/blog)

Part 3: Add the Entity Specific Controllers

Step 1: Cars Controller

- Create a new class named `CarsController.cs` in the Controllers directory. Update the using statements to the following:

```
using System.Collections.Generic;
using AutoLot.Api.Controllers.Base;
using Microsoft.AspNetCore.Mvc;
using AutoLot.Models.Entities;
using AutoLot.Dal.Repos.Interfaces;
using AutoLot.Services.Logging;
using Microsoft.AspNetCore.Http;
```

- Make the class public, inherit from `BaseCrudController` passing in the generic types, and add the controller level `Route` attribute:

```
namespace AutoLot.Api.Controllers
{
  [Route("api/[controller]")]
  public class CarsController : BaseCrudController<Car, CarsController>
  {
  }
}
```

- Add a constructor that takes an instance of `ICarRepo` and the strongly typed logger:

```
public CarsController(ICarRepo carRepo, IAppLogging<CarsController> logger)
  : base(carRepo, logger) { }
```

- Add the `GetCarsByMake` method:

```
[HttpGet("bymake/{id?}")]
public ActionResult<IEnumerable<Car>> GetCarsByMake(int? id)
{
  if (id.HasValue && id.Value>0)
  {
    return Ok(((ICarRepo)MainRepo).GetAllBy(id.Value));
  }
  return Ok(MainRepo.GetAllIgnoreQueryFilters());
}
```

Step 2: CreditRisks Controller

- Create a new class named `CreditRisksController.cs` in the Controllers directory. Update the using statements to the following:

```
using AutoLot.Api.Controllers.Base;
using AutoLot.Models.Entities;
using AutoLot.Dal.Repos.Interfaces;
using Microsoft.AspNetCore.Mvc;
using AutoLot.Services.Logging;
```

- Make the class public, inherit from `BaseCrudController` passing in the generic types, and add the controller level `Route` attribute:

```
namespace AutoLot.Api.Controllers
{
  [Route("api/[controller]")]
  public class CreditRisksController : BaseCrudController<CreditRisk, CreditRisksController>
  {
  }
}
```

- Add a constructor that takes an instance of `ICarRepo` and the strongly typed logger:

```
public CreditRisksController(
  ICreditRiskRepo creditRiskRepo, IAppLogging<CreditRisksController> logger)
  : base(creditRiskRepo, logger) { }
```

Step 3: Customers Controller

- Create a new class named `CustomersController.cs` in the Controllers directory. Update the using statements to the following:

```
using AutoLot.Api.Controllers.Base;
using AutoLot.Models.Entities;
using AutoLot.Dal.Repos.Interfaces;
using Microsoft.AspNetCore.Mvc;
using AutoLot.Services.Logging;
```

- Make the class public, inherit from `BaseCrudController` passing in the generic types, and add the controller level `Route` attribute:

```
namespace AutoLot.Api.Controllers
{
  [Route("api/[controller]")]
  public class CustomersController : BaseCrudController<Customer, CustomersController>
  {
  }
}
```

- Add a constructor that takes an instance of `ICarRepo` and the strongly typed logger:

```
public CustomersController(ICustomerRepo customerRepo, IAppLogging<CustomersController> logger)
  : base(customerRepo, logger) { }
```

All files copyright Phil Japikse (http://www.skimedic.com/blog)

Step 4: Makes Controller

- Create a new class named `MakesController.cs` in the Controllers directory. Update the using statements to the following:

```
using AutoLot.Api.Controllers.Base;
using AutoLot.Models.Entities;
using Microsoft.AspNetCore.Mvc;
using AutoLot.Dal.Repos.Interfaces;
using AutoLot.Services.Logging;
```

- Make the class public, inherit from `BaseCrudController` passing in the generic types, and add the controller level `Route` attribute:

```
namespace AutoLot.Api.Controllers
{
  [Route("api/[controller]")]
  public class MakesController : BaseCrudController<Make, MakesController>
  {
  }
}
```

- Add a constructor that takes an instance of `ICarRepo` and the strongly typed logger:

```
public MakesController(IMakeRepo makeRepo, IAppLogging<MakesController> logger)
  : base(makeRepo, logger) { }
```

Step 5: Orders Controller

- Create a new class named `OrdersController.cs` in the Controllers directory. Update the using statements to the following:

```
using AutoLot.Api.Controllers.Base;
using AutoLot.Dal.Repos.Interfaces;
using AutoLot.Models.Entities;
using AutoLot.Services.Logging;
using Microsoft.AspNetCore.Mvc;
```

- Make the class public, inherit from `BaseCrudController` passing in the generic types, and add the controller level `Route` attribute:

```
namespace AutoLot.Api.Controllers
{
  [Route("api/[controller]")]
  public class OrdersController : BaseCrudController<Order, OrdersController>
  {
  }
}
```

- Add a constructor that takes an instance of `ICarRepo` and the strongly typed logger:

```
public OrdersController(IOrderRepo orderRepo, IAppLogging<OrdersController> logger)
  : base(orderRepo, logger) { }
```

Summary

This lab created and configured the Controllers for the service.

Next steps

In the next part of this tutorial series, you will augment the basic Swagger support in the services app.

All files copyright Phil Japikse (http://www.skimedic.com/blog)