

Build an ASP.NET Core MVC App with EF Core

One-Day Hands-On Lab

Lab 12

This lab walks you through creating the `BaseCrudController` and finishing the `CarsController`. Prior to starting this lab, you must have completed Lab 11.

NOTE: The views will be created in the next lab.

Part 1: Create the `BaseCrudController`

Step 1: Update the global using statements

- Add the following global using statements to the `GlobalUsings.cs` file:

```
global using AutoLot.Dal.Repos.Base;
global using AutoLot.Models.Entities.Base;
global using AutoLot.Mvc.Controllers.Base;
global using Microsoft.AspNetCore.Mvc.Rendering;
```

Step 2: Create the `BaseCrudController` class, constructor, and helper methods

- Create a new folder named `Base` in the `Controllers` folder, and in this folder create a new class named `BaseCrudController`. Make the class `public abstract` and inherit `Controller`. Since this will serve many downstream controllers, make it generic, taking in an entity and controller type. Finally, add the default route to the controller:

```
namespace AutoLot.Mvc.Controllers.Base;
```

```
[Route("[controller]/[action]")]
public abstract class BaseCrudController : Controller where TEntity:
BaseEntity, new()
{
    //implementation goes here
}
```

- Add a constructor that takes an instance of `IAppLogging<TController>` and `IBaseRepo` and assigns them to private variables:

```
protected readonly IAppLogging<TController> AppLoggingInstance;
protected readonly IBaseRepo<TEntity> BaseRepoInstance;
```

```
protected BaseCrudController(IAppLogging<TController> appLogging, IBaseRepo<TEntity> baseRepo)
{
    AppLoggingInstance = appLogging;
    BaseRepoInstance = baseRepo;
}
```

- Add an abstract function that returns a SelectList of look up values (like Makes):

```
protected abstract SelectList GetLookupValues();
```

- Add a helper function that gets a single entity:

```
protected TEntity GetOneEntity(int? id) => id == null ? null : BaseRepoInstance.Find(id.Value);
```

Step 3: Add the Index and Details action methods

- Create the Index action method, set the routing, and return all entities:

```
[Route("/[controller]")]
[Route("/[controller]/[action]")]
[HttpGet]
public virtual IActionResult Index() => View(BaseRepoInstance.GetAllIgnoreQueryFilters());
```

- Create the Details action method, set the routing, and return a single entity:

```
[HttpGet("{id?}")]
public virtual IActionResult Details(int? id)
{
    if (!id.HasValue)
    {
        return BadRequest();
    }
    var entity = GetOneEntity(id);
    if (entity == null)
    {
        return NotFound();
    }
    return View(entity);
}
```

Step 3: Add the Create Action Methods

- Update the HttpGet Create Action Method:

```
[HttpGet]
public virtual IActionResult Create()
{
    ViewData["LookupValues"] = GetLookupValues();
    return View();
}
```

- Add the HttpPost Create action method:

```
[HttpPost]
[ValidateAntiForgeryToken]
public virtual IActionResult Create(TEntity entity)
{
    if (ModelState.IsValid)
    {
        BaseRepoInstance.Add(entity);
        return RedirectToAction(nameof(Index));
    }
    ViewData["LookupValues"] = GetLookupValues();
}
```

```
return View(entity);
}
```

Step 4: Add/Update the Edit Action Methods

- Update the `HttpGet` Edit Action Method:

```
[HttpGet("{id?}")]
public virtual IActionResult Edit(int? id)
{
    var entity = GetOneEntity(id);
    if (entity == null)
    {
        return NotFound();
    }
    ViewData["LookupValues"] = GetLookupValues();
    return View(entity);
}
```

- Add the `HttpPost` Edit action method:

```
[HttpPost("{id}")]
[ValidateAntiForgeryToken]
public virtual IActionResult Edit(int id, TEntity entity)
{
    if (id != entity.Id)
    {
        return BadRequest();
    }
    if (ModelState.IsValid)
    {
        BaseRepoInstance.Update(entity);
        return RedirectToAction(nameof(Index));
    }
    ViewData["LookupValues"] = GetLookupValues();
    return View(entity);
}
```

Step 5: Add/Update the Delete Action Methods

- Update the `HttpGet` Delete Action Method:

```
[HttpGet("{id?}")]
public virtual IActionResult Delete(int? id)
{
    var entity = GetOneEntity(id);
    if (entity == null)
    {
        return NotFound();
    }
    return View(entity);
}
```

- Add the `HttpPost` Delete action method:

```
[HttpPost("{id}")]
```

```
[ValidateAntiForgeryToken]
public virtual IActionResult Delete(int id, TEntity entity)
{
    BaseRepoInstance.Delete(entity);
    return RedirectToAction(nameof(Index));
}
```

Part 2: Update the Cars Controller

Step 1: Update the class to use the BaseCrudController and Implement the SelectList Helper Function

- ❑ Remove the route (it comes from the base class) and inherit from BaseCrudController. Next, delete all of the action methods except for the ByMake method:

```
namespace AutoLot.Mvc.Controllers;

public class CarsController : BaseCrudController<Car,CarsController>
{
    [HttpGet("{makeId}/{makeName}")]
    public IActionResult ByMake(int makeId, string makeName)
    {
        return View();
    }
}
```

- ❑ Add a constructor that takes an instance of IAppLogging<T>, ICarRepo, and IMakeRepo, passing the first two to the base class and assign the IMakeRepo to a private variable:

```
private readonly IMakeRepo _makeRepo;
public CarsController(
    IAppLogging<CarsController> logging,
    ICarRepo repo,
    IMakeRepo makeRepo)
    :base(logging,repo)
{
    _makeRepo = makeRepo;
}
```

- ❑ Override the abstract function to get the SelectList from the Makes:

```
protected override SelectList GetLookupValues()
=> new SelectList(_makeRepo.GetAll(), nameof(Make.Id), nameof(Make.Name));
```

Step 2: Update the ByMake action method

- ❑ Update the ByMake action method, set the routing, and return all cars for a certain make:

```
[HttpGet("{makeId}/{makeName}")]
public IActionResult ByMake(int makeId, string makeName)
{
    ViewBag.MakeName = makeName;
    return View(((ICarRepo)BaseRepoInstance).GetAllBy(makeId));
}
```

Summary

In this lab you created the BaseCrudController and finished the Cars Controller. The application will not properly run until after completing the next lab, which updates and/or adds the views.

Next steps

In the next part of this tutorial series, you will create the Views for the application.