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

Lab 16

This lab walks you through finishing the Cars Controller. Prior to starting this lab, you must have completed Lab 15. In this lab, you need to complete **either** Part 1 or Part 2.

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

Part 1: Update the Cars Controller (EF Core)

Step 1: Update the using Statements an Add the Constructor and Helper Functions

- Update the using statements for the `CarsController` to match the following:

```
using AutoLot.Dal.Repos.Interfaces;
using AutoLot.Models.Entities;
using AutoLot.Services.Logging;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
```

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

```
private readonly ICarRepo _repo;
private readonly IAppLogging<CarsController> _logging;
public CarsController(ICarRepo repo, IAppLogging<CarsController> logging)
{
  _repo = repo;
  _logging = logging;
  //_logging.LogAppError("Test error");
}
```

- Add a helper function that creates a `SelectList` from the Makes:

```
internal SelectList GetMakes(IMakeRepo makeRepo)
  => new SelectList(makeRepo.GetAll(), nameof(Make.Id), nameof(Make.Name));
```

- Add a helper function that gets a single car:

```
internal Car GetOneCar(int? id) => id == null ? null : _repo.Find(id.Value);
```

Step 2: Add the Index and Details action methods

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

```
[Route("/[controller]")]
[Route("/[controller]/[action]")]
public IActionResult Index() => return View(_repo.GetAllIgnoreQueryFilters());
```

- Create 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(_repo.GetAllBy(makeId));
}
```

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

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

Step 3: Add/Update the Create Action Methods

- Update the `HttpGet Create` Action Method:

```
[HttpGet]
public IActionResult Create([FromServices] IMakeRepo makeRepo)
{
  ViewData["MakeId"] = GetMakes(makeRepo);
  return View();
}
```

- Add the `HttpPost Create` action method:

```
[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Create([FromServices] IMakeRepo makeRepo, Car car)
{
  if (ModelState.IsValid)
  {
    _repo.Add(car);
    return RedirectToAction(nameof(Index));
  }
  ViewData["MakeId"] = GetMakes(makeRepo);
  return View(car);
}
```


Step 4: Add/Update the Edit Action Methods

- Update the `HttpGet Edit` Action Method:

```
[HttpGet("{id?}")]
public IActionResult Edit([FromServices] IMakeRepo makeRepo, int? id)
{
  var car = GetOneCar(id);
  if (car == null) { return NotFound();}
  ViewData["MakeId"] = GetMakes(makeRepo);
  return View(car);
}
```

- Add the `HttpPost Edit` action method:

```
[HttpPost("{id}")]
[ValidateAntiForgeryToken]
public IActionResult Edit([FromServices] IMakeRepo makeRepo, int id, Car car)
{
  if (id != car.Id) { return BadRequest();}
  if (ModelState.IsValid)
  {
    _repo.Update(car);
    return RedirectToAction(nameof(Index));
  }
  ViewData["MakeId"] = GetMakes(makeRepo);
  return View(car);
}
```

Step 5: Add/Update the Delete Action Methods

- Update the `HttpGet Delete` Action Method:

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

- Add the `HttpPost Delete` action method:

```
[HttpPost("{id}")]
[ValidateAntiForgeryToken]
public IActionResult Delete(int id, Car car)
{
  _repo.Delete(car);
  return RedirectToAction(nameof(Index));
}
```


Part 2: Update the Cars Controller (API)

Step 1: Update the using Statements an Add the Constructor and Helper Functions

- Update the using statements for the `CarsController` to match the following:

```
using System.Threading.Tasks;
using AutoLot.Models.Entities;
using AutoLot.Services.ApiWrapper;
using AutoLot.Services.Logging;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
```

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

```
private readonly IApiServiceWrapper _serviceWrapper;
private readonly IAppLogging<CarsController> _logging;
public CarsController(IApiServiceWrapper serviceWrapper, IAppLogging<CarsController> logging)
{
  _serviceWrapper = serviceWrapper;
  _logging = logging;
  //_logging.LogAppError("Test error");
}
```

- Add a helper function that creates a `SelectList` from the Makes:

```
internal async Task<SelectList> GetMakesAsync()
  => new SelectList(await _serviceWrapper.GetMakesAsync(), nameof(Make.Id), nameof(Make.Name));
```

- Add a helper function that gets a single car:

```
internal async Task<Car> GetOneCarAsync(int? id)
  => id == null ? null : await _serviceWrapper.GetCarAsync(id.Value);
```

Step 2: Add the Index and Details action methods

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

```
[Route("/[controller]")]
[Route("/[controller]/[action]")]
public async Task<IActionResult> Index()=> View(await _serviceWrapper.GetCarsAsync());
```

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

```
[HttpGet("{makeId}/{makeName}")]
public async Task<IActionResult> ByMake(int makeId, string makeName)
{
  ViewBag.MakeName = makeName;
  return View(await _serviceWrapper.GetCarsByMakeAsync(makeId));
}
```


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

```
[HttpGet("{id?}")]
public async Task<IActionResult> Details(int? id)
{
  if (!id.HasValue) { return BadRequest(); }
  var car = await GetOneCarAsync(id);
  if (car == null) { return NotFound();}
  return View(car);
}
```

Step 3: Add/Update the Create Action Methods

- Update the `HttpGet Create` Action Method:

```
[HttpGet]
public async Task<IActionResult> Create()
{
  ViewData["MakeId"] = await GetMakesAsync();
  return View();
}
```

- Add the `HttpPost Create` action method:

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create(Car car)
{
  if (ModelState.IsValid)
  {
    await _serviceWrapper.AddCarAsync(car);
    return RedirectToAction(nameof(Index));
  }
  ViewData["MakeId"] = await GetMakesAsync();
  return View(car);
}
```

Step 4: Add/Update the Edit Action Methods

- Update the `HttpGet Edit` Action Method:

```
[HttpGet("{id?}")]
public async Task<IActionResult> Edit(int? id)
{
  var car = await GetOneCarAsync(id);
  if (car == null) { return NotFound();}
  ViewData["MakeId"] = await GetMakesAsync();
  return View(car);
}
```


- Add the `HttpPost Edit` action method:

```
[HttpPost("{id}")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id, Car car)
{
  if (id != car.Id) { return BadRequest();}
  if (ModelState.IsValid)
  {
    await _serviceWrapper.UpdateCarAsync(id,car);
    return RedirectToAction(nameof(Index));
  }
  ViewData["MakeId"] = await GetMakesAsync();
  return View(car);
}
```

Step 5: Add/Update the Delete Action Methods

- Update the `HttpGet Delete` Action Method:

```
[HttpGet("{id?}")]
public async Task<IActionResult> Delete(int? id)
{
  var car = await GetOneCarAsync(id);
  if (car == null) { return NotFound();}
  return View(car);
}
```

- Add the `HttpPost Delete` action method:

```
[HttpPost("{id}")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Delete(int id, Car car)
{
  await _serviceWrapper.DeleteCarAsync(id,car);
  return RedirectToAction(nameof(Index));
}
```

Summary

In this lab you 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.

