.NET 10 App Dev Hands-On Workshop

API Lab 3 –Controllers

This lab creates and configures controllers for the RESTful service. Before starting this lab, you must have completed API Lab 2b.

Part 1: The BaseCrudController

Copilot Agent Mode

```
Setup Prompt: Always use file scoped namespaces. Always combine attributes on a single line when
possible. The project does not use nullable reference types. There is a GlobalUsings.cs file that
includes common usings, don't include using statements in new files if they are already in the
globalusings.cs file. I prefer expression bodied members when possible. Single line if statements
should still use braces. Use ternary operators when appropriate. Use internal over private. All
classes and methods are public unless told otherwise. Don't add a constructor unless instructed to
do so. Use primary constructors when possible and don't declare a class level variable if the
parameter from the constructor can be used. Don't initialize properties unless instructed to do
so. All work is to be done in the Autolot.API project.

Prompt: In the Controllers folder, add a new folder named Base and add a new public abstract class
named BaseCrudController that inherits from ControllerBase.
Make it generic with TEntity, constrained to BaseEntity, new(). Add the ApiController and Route
(api/controller) attributes to the top of the class.
Accept two parameters in the constructor, IAppLogging (appLogging) and IBaseRepo<TEntity>
(baseRepo).
Assign the parameters to two protected readonly properties named AppLoggingInstance and
MainRepoInstance, respectively.
Add the following public virtial action methods:
HttpGet GetAll - (input:none. return type: ActionResult<List<TEntity>>. Logic: return
Ok(MainRepoInstance.GetAllIgnoreQueryFilters.ToList()))
HttpGet("{id}") GetOne -  (input int. return ActionResult<TEntity>. Logic: call
MainRepoInstance.Find(id). If Entity is null, return NoContent(). Else return Ok(entity)
HttpPut("{id}") UpdateOne (input int id, TEntity entity. return ActionResult<TEntity>.
Logic: if id != entity.Id, return BadRequest, if ModelState is not valid, return
ValidationProblem(ModelState).
Else call MainRepoInstance.Update with entity, then return Ok(entity)
HttpPost() AddOne (input TEntity entity. return ActionResult<TEntity>.
Logic: if ModelState is not valid, return ValidationProblem(ModelState) else call
MainRepoInstance.Add with entity,
then return CreatedAtAction(nameof(GetOne),new {id = entity.Id},entity))
HttpDelete("{id}) DeleteOne (input int id. return IActionResult. Logic: If id != entity.Id, return
BadRequest. Else call Delete on MainRepoInstance. return Ok()

Prompt: Add the following global usings to the GlobalUsings.cs file if it does not already exist
(sorted alphabetically. Don't remove any existing global using statements).
global using AutoLot.Api.Controllers.Base;
```

Manual

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 and update the code to the following:

```
namespace AutoLot.Api.Controllers.Base;

[ApiController]
[Route("api/[controller]")]
public abstract class BaseCrudController<TEntity>(
    IAppLogging appLogging, IBaseRepo<TEntity> baseRepo)
  : ControllerBase where TEntity : BaseEntity, new()
{
  protected readonly IBaseRepo<TEntity> MainRepoInstance = baseRepo;
  protected readonly IAppLogging AppLoggingInstance = appLogging;
//rest of the code goes here
}
```

Step 2: Add the Get Methods

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

```
[HttpGet]
public virtual ActionResult<List<TEntity>> GetAll()
  => Ok(MainRepoInstance.GetAllIgnoreQueryFilters().ToList());

[HttpGet("{id}")]
public virtual ActionResult<TEntity> GetOne(int id)
{
  var entity = MainRepoInstance.Find(id);
  return entity == null ? NoContent() : Ok(entity);
}
```

Step 3: Add the Update Method

```
[HttpPut("{id}")]
public virtual ActionResult<TEntity> UpdateOne(int id, TEntity entity)
{
  if (id != entity.Id)
  {
    return BadRequest();
  }
  if (!ModelState.IsValid)
  {
    return ValidationProblem(ModelState);
  }
  MainRepoInstance.Update(entity);
  return Ok(entity);
}
```


Step 4: Add the Add Method

```
[HttpPost]
public virtual ActionResult<TEntity> AddOne(TEntity entity)
{
  if (!ModelState.IsValid)
  {
    return ValidationProblem(ModelState);
  }
  MainRepoInstance.Add(entity);
  return CreatedAtAction(nameof(GetOne), new { id = entity.Id }, entity);
}
```

Step 5: Add the Delete Method

```
[HttpDelete("{id}")]
public virtual IActionResult DeleteOne(int id, TEntity entity)
{
  if (id != entity.Id)
  {
    return BadRequest();
  }
  MainRepoInstance.Delete(entity);
  return Ok();
}
```

Step 6: Update the GlobalUsings

- Add the following to the `GlobalUsings.cs` file:

```
global using AutoLot.Api.Controllers.Base;
```

Part 2: Add the Entity Specific Controllers

Copilot Agent Mode

```
Prompt: Add a new class CarsController to the controllers folder that inherits from
BaseCrudController with TEntity as Car and ICarRepo as BaseRepo<TEntity>.
Don't need the ApiController or Route attributes on this class since they are in the base class.
Add the following method:
HttpGet("bymake/{makeId?}") ByMake (input int? makeId. if makeId.HasValue and is greater than
zero, return Ok(cast MainRepoInstance to ICarRepo and call GetAllBy passing in the makeId.Value).
Else return GetAll())

Prompt: Add a new class for each entity (CarDriver, Driver, make,and Radio) name
<EntityName>sController to the controllers folder that inherit from
BaseCrudController with TEntity as <entityType> and I<EntityType>Repo as BaseRepo<TEntity>.
Don't need the ApiController or Route attributes on this class since they are in the base class.
```


Manual

Step 1: The Cars Controller

- Create a new class named `CarsController.cs` in the Controllers directory. Most of the functionality is handled by the base controller. Update the code to the following:

```
namespace AutoLot.Api.Controllers;
public class CarsController(IAppLogging appLogging, ICarRepo carRepo)
  : BaseCrudController<Car>(appLogging, carRepo)
{
  [HttpGet("bymake/{makeId?}")]
  public ActionResult<List<Car>> ByMake(int? makeId)
    => (makeId is > 0)
        ? Ok(((ICarRepo)MainRepoInstance).GetAllBy(makeId.Value))
        : GetAll();
}
```

Step 2: The Remaining Controllers

- The controllers all follow the same pattern. Create the remaining controllers as shown here:

```
//CarDriversController
namespace AutoLot.Api.Controllers;
public class CarDriversController(IAppLogging appLogging, ICarDriverRepo repo)
  : BaseCrudController<CarDriver>(appLogging, repo);
//DriversController
namespace AutoLot.Api.Controllers;
public class DriversController(IAppLogging appLogging, IDriverRepo repo)
  : BaseCrudController<Driver>(appLogging, repo);
//MakesController
namespace AutoLot.Api.Controllers;
public class MakesController(IAppLogging appLogging, IMakeRepo repo)
  : BaseCrudController<Make>(appLogging, repo);
//RadiosController
namespace AutoLot.Api.Controllers;
public class RadiosController(IAppLogging appLogging, IRadioRepo repo)
  : BaseCrudController<Radio>(appLogging, repo);
```

Summary

This lab created and configured the Controllers for the service.

Next steps

In the next part of this tutorial series, you will add Content Negotiations and Data Shaping to the RESTful service.