

Base Controller Lab

Perform these labs on your own computer using Visual Studio 2022 to ensure you understand the lessons presented in the corresponding videos and lectures.

Lab 1: Create Base Controller Class

Right mouse-click on the project folder and add a new folder named **BaseClasses**.

Right mouse-click on the BaseClasses folder and add a new class named **ControllerBaseAPI**.

Replace the contents in the file with the following code.

```
using Microsoft.AspNetCore.Mvc;

namespace AdvWorksAPI.BaseClasses;

public class ControllerBaseAPI : ControllerBase
{
    protected readonly ILogger _Logger;

    public ControllerBaseAPI(ILogger logger)
    {
        _Logger = logger;
        InfoMessage = string.Empty;
        ErrorLogMessage = string.Empty;
    }

    public string InfoMessage { get; set; }
    public string ErrorLogMessage { get; set; }
}
```

Open the **CustomerController.cs** file and add a new using statement.

```
using AdvWorksAPI.BaseClasses;
```

Change the class declaration to inherit from the **ControllerBaseAPI** class

```
public class CustomerController : ControllerBaseAPI
```

Remove the **private readonly _Logger** variable

Remove the line that sets the **_Logger** variable within the constructor.

Modify the constructor to pass the **logger** parameter to the base class.

```
public CustomerController(IRepository<Customer> repo,
ILogger<CustomerController> logger) : base(logger)
{
    _Repo = repo;
}
```

Open the **LogTestController.cs** file and add a new using statement.

```
using AdvWorksAPI.BaseClasses;
```

Change the class declaration to inherit from the **ControllerBaseAPI** class

```
public class LogTestController : ControllerBaseAPI
```

Remove the **private readonly _Logger** variable

Remove the line that sets the **_Logger** variable within the constructor.

Modify the constructor to pass the **logger** parameter to the base class.

Your class declaration should now look like the following:

```
public class LogTestController : ControllerBaseAPI
{
    public LogTestController(ILogger<LogTestController>
logger) : base(logger)
    {
    }

    // REST OF THE CODE HERE
}
```

Compile

Compile the application and make sure everything builds.

Lab 2: Add Exception Handling Method

Open the **ControllerBaseAPI.cs** file.

Add the two overloads named `HandleException<T>()` shown below to log an exception and return a status code of 500 with a generic message appropriate for the caller of this API.

```
/// <summary>
/// Call this method to return a '500 Internal Server
Error' and log an exception.
/// </summary>
/// <typeparam name="T">The type to return</typeparam>
/// <param name="ex">An Exception object</param>
/// <param name="infoMsg">The info message to display to
the user</param>
/// <param name="errorMsg">The error message to
log</param>
/// <returns>A Status Code of 500</returns>
protected ActionResult<T> HandleException<T>(Exception
ex, string infoMsg, string errorMsg)
{
    // Set properties from parameters passed in
    InfoMessage = infoMsg;
    ErrorLogMessage = errorMsg;

    return HandleException<T>(ex);
}

/// <summary>
/// Call this method to return a '500 Internal Server
Error' and log an exception.
/// Prior to calling this method...
///     Fill in the InfoMessage property with the value
to display to the caller.
///     Fill in the ErrorLogMessage property with the
value to place into the log file.
/// </summary>
/// <typeparam name="T">The type to return</typeparam>
/// <param name="ex">An Exception object</param>
/// <returns>A Status Code of 500</returns>
protected ActionResult<T> HandleException<T>(Exception
ex)
{
    ActionResult<T> ret;

    // Create status code with generic message
    ret =
StatusCode(StatusCodes.Status500InternalServerError,
InfoMessage);

    // Add Message, Source, and Stack Trace
    ErrorLogMessage += $"{Environment.NewLine}Message:
{ex.Message}";
}
```

```
        ErrorLogMessage += $"{Environment.NewLine}Source:
        {ex.Source}";
        ErrorLogMessage += $"{Environment.NewLine}Stack Trace:
        {ex.StackTrace}";

        // Log the exception
        _Logger.LogError(ex, "{ErrorLogMessage}",
        ErrorLogMessage);

        return ret;
    }
```

Open the **CustomerController.cs** file.

Modify the Get() method with the changes shown in **bold** below.

```
[HttpGet]
[ProducesResponseType (StatusCodes.Status200OK) ]
[ProducesResponseType (StatusCodes.Status404NotFound) ]
[ProducesResponseType (StatusCodes.Status500InternalServerError)]
public ActionResult<IEnumerable<Customer>> Get()
{
    ActionResult<IEnumerable<Customer>> ret;
    List<Customer> list;
    InfoMessage = "No Customers are available.";

    try {
        // Intentionally Cause an Exception
        throw new ApplicationException("ERROR!");

        // Get all data
        list = _Repo.Get();

        if (list != null && list.Count > 0) {
            ret = StatusCode(StatusCodes.Status200OK, list);
        }
        else {
            ret = StatusCode(StatusCodes.Status404NotFound,
InfoMessage);
        }
    }
    catch (Exception ex) {
        InfoMessage = "Error in Customer API. Please Contact the System Administrator.";

        ErrorLogMessage = "Error in CustomerController.Get()";

        ret = HandleException<IEnumerable<Customer>>(ex);
    }

    return ret;
}
```

Try it Out

Delete all files in the **Logs** folder.

Run the application and click on the **GET /api/Customer** button.

Ensure all exceptions appear as they should.

Lab 3: Serialize an Object

Open the **ControllerBaseAPI.cs** file and add a new property.

```
public string EntityAsJson { get; set; }
```

Modify the constructor to set the **EntityAsJson** property to an empty string.

```
public ControllerBaseAPI(ILogger logger)
{
    _Logger = logger;
    InfoMessage = string.Empty;
    ErrorLogMessage = string.Empty;
    EntityAsJson = string.Empty;
}
```

Add a method named **SerializeEntity()**.

```
/// <summary>
/// Serialize an object into a JSON string
/// </summary>
/// <typeparam name="T">The type to
serialize</typeparam>
/// <param name="entity">An instance of the type</param>
/// <returns>A JSON string</returns>
protected string SerializeEntity<T>(T entity)
{
    try {
        // Attempt to serialize entity
        EntityAsJson = JsonSerializer.Serialize(entity);
    }
    catch {
        // Ignore the error
    }

    return EntityAsJson;
}
```

Open the **LogTestController.cs** file.

Locate the **LogCustomer()** method and make the change shown in **bold** below.

```
private void LogCustomer()
{
    // Log an Object
    Customer entity = new()
    {
        CustomerID = 999,
        FirstName = "Bruce",
        LastName = "Jones",
        Title = "Mr.",
        CompanyName = "Beach Computer Consulting",
        EmailAddress =
"Jones.Bruce@beachcomputerconsulting.com",
        Phone = "(714) 555-5555",
        ModifiedDate = DateTime.Now
    };

    string json = base.SerializeEntity<Customer>(entity);

    _Logger.LogInformation("Customer = {json}", json);
}
```

Try it Out

Delete all files in the **Logs** folder.

Run the application and click on the **GET /api/LogTest/LogObject** button.

Stop the application and check the **InfoLog-nnnn.txt** file to see the Customer is still serialized.