.NET 10 App Dev Hands-On Workshop

API Lab 4 – Content Negotiation and Data Shaping

This lab covers two topics with RESTful services: Content Negotiation and Data Shaping. Before starting this lab, you must have completed API Lab 3 and EF Core Lab 9. This entire lab works in the `AutoLot.Api` project.

Part 1: Content Negotiation - XML

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 a new class named ContentNegotiationController that
inherits from ControllerBase
and has the ApiController and Route ("api/[controller]") attributes.
Add the following method:
  [HttpGet]
  [Produces("application/json","application/xml","text/csv")]
  public IActionResult Get(IDriverRepo driverRepo) => Ok(driverRepo.GetAll().ToList());

Prompt: In program.cs, add the call to AddXmlSerializerFormatters() after the call to
AddControllersAsServices().
Update the AddControllers() constructor call to add: config.RespectBrowserAcceptHeader = true.
```

Manual

Step 1: Add the ContentNegotiationController

- Add a new controller to the `Controllers` folder:

```
namespace AutoLot.Api.Controllers;
[ApiController]
[Route("api/[controller]")]
public class ContentNegotiationController : ControllerBase
{
  [HttpGet]
  [Produces("application/json","application/xml","text/csv")]
  public IActionResult Get(IDriverRepo driverRepo) => Ok(driverRepo.GetAll().ToList());
}
```

Step 2: Update the Program.cs file to support XML Serialization

- Update the `AddControllers()` method to respect the Accept header and support XML Serialization (changes in bold):

```
builder.Services.AddControllers(config =>
  {
    config.Filters.Add(new CustomExceptionFilterAttribute(builder.Environment));
    config.RespectBrowserAcceptHeader = true;
  })
  .AddControllersAsServices()
  .AddXmlSerializerFormatters()
  //omitted for brevity
```

Step 3: Test Using Bruno or CURL

- Use Bruno or CURL to test the new endpoint. Make sure to add the Accept: application/xml header to the request:

```
CURL -X GET https://localhost:5011/api/ContentNegotiation?v=1.0 -H "accept: application/xml"
```

Part 2: Content Negotiation - CSV

Step 1: Create the CSV Output Formatter

- Add a new folder named `Formatters`, and in that folder, add a new class named `CustomCsvOutputFormatter.cs`. Update the code to the following:

```
using MediaTypeHeaderValue = Microsoft.Net.Http.Headers.MediaTypeHeaderValue;
namespace AutoLot.Api.Formatters;
public class CustomCsvOutputFormatter : TextOutputFormatter
{
  public CustomCsvOutputFormatter()
  {
    SupportedMediaTypes.Add(MediaTypeHeaderValue.Parse("text/csv"));
    SupportedEncodings.Add(Encoding.UTF8);
  }
  protected override bool CanWriteType(Type type) =>
    typeof(IEnumerable<object>).IsAssignableFrom(type) || type is { IsClass: true };
  public override async Task WriteResponseBodyAsync(
    OutputFormatterWriteContext context, Encoding selectedEncoding)
  {
    var response = context.HttpContext.Response;
    var buffer = new StringBuilder();
    var type = context.Object?.GetType();
    if (type == null)
    {
      await response.WriteAsync(string.Empty);
      return;
    }
    var props = type.GetGenericArguments().FirstOrDefault()?.GetProperties()
      ?? type.GetProperties();
    var enumerable = context.Object as IEnumerable<object> ?? [context.Object];
    //header
    buffer.AppendLine(string.Join(",", props.Select(p => p.Name)));
    //rows
    foreach (var item in enumerable)
    {
      var values = props.Select(p => p.GetValue(item, null)?.ToString()?.Replace(",", " ") ?? "");
      buffer.AppendLine(string.Join(",", values));
    }
    await response.WriteAsync(buffer.ToString());
  }
}
```

Step 2: Update the GlobalUsings.cs File

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

```
global using AutoLot.Api.Formatters;
```

Step 3: Update the Program.cs file to support CSV Serialization

- Update the `AddControllers()` method to respect the Accept header and support XML Serialization (changes in bold):

```
builder.Services.AddControllers(
  {
    //omitted for brevity
    config.OutputFormatters.Add(new CustomCsvOutputFormatter());
  })
```

Step 4: Test Using Bruno or CURL

- Use Bruno or CURL to test the new endpoint. Make sure to add the Accept: application/xml header to the request:

```
CURL -X GET https://localhost:5011/api/ContentNegotiation?v=1.0 -H "accept: text/csv"
```

Part 3: Data Shaping

Step 1: Create the ShapedEntity Class

- Create a new folder named `DataShaping`, and in the folder, create a new class named `ShapedEntity.cs`. Update the code to the following:

```
namespace AutoLot.Api.DataShaping;
public class ShapedEntity
{
  public ExpandoObject Entity { get; set; }
  public int Id { get; set; }
}
```

Step 2: Create the DataShaper Interface

- Create a new interface named `IDataShaper.cs` in the `DataShaping` folder. Update the code to the following:

```
namespace AutoLot.Api.DataShaping;
public interface IDataShaper<T>
{
  IEnumerable<ShapedEntity> ShapeData(IEnumerable<T> entities, string fieldsString);
  ShapedEntity ShapeData(T entity, string fieldsString);
  void UpdateData(T entity, Dictionary<string, string> values);
}
```

Step 4: Create the DataShaper Implementation

- Create a new class named `DataShaper.cs`. Update the code to the following:

```
public class DataShaper<T> : IDataShaper<T>
{
  private readonly List<PropertyInfo> _properties;
  private readonly List<PropertyInfo> _complexProperties;
  public DataShaper()
  {
    _properties = typeof(T).GetProperties(BindingFlags.Public | BindingFlags.Instance).ToList();
    _complexProperties = _properties
      .Where(p => p.PropertyType.IsClass && p.PropertyType != typeof(string)).ToList();
    }
  public IEnumerable<ShapedEntity> ShapeData(IEnumerable<T> entities, string fieldsString)
     => entities.Select(e => ShapeData(e, fieldsString));
  internal PropertyInfo FindProperty(List<PropertyInfo> properties, string fieldName)
    => properties.FirstOrDefault(p => p.Name.Equals(fieldName.Trim(),
                                      StringComparison.OrdinalIgnoreCase));
  internal void SetValue<TI>(TI entity, ExpandoObject shapedObject, PropertyInfo prop)
  {
    ((IDictionary<string, object>)shapedObject).Add(prop.Name, prop.GetValue(entity));
  }
  public void UpdateData(T entity, Dictionary<string, string> values)
  {
    //This only works with strings
    foreach (var entry in values)
    {
      var property = FindProperty(_properties, entry.Key);
      if (property != null)
      {
        property.SetValue(entity, entry.Value);
        continue;
      }
      foreach (var complexProperty in _complexProperties)
      {
        var innerProperties = complexProperty.PropertyType
          .GetProperties(BindingFlags.Public | BindingFlags.Instance).ToList();
        var innerProperty = FindProperty(innerProperties, entry.Key);
        if (innerProperty != null)
        {
          var innerValue = complexProperty.GetValue(entity);
          if (innerValue == null)
          {
            innerValue = Activator.CreateInstance(complexProperty.PropertyType);
            complexProperty.SetValue(entity, innerValue);
          }
          innerProperty.SetValue(innerValue, entry.Value);
          continue;
        }
      }
    }
  }
  public ShapedEntity ShapeData(T entity, string fieldsString)
  {
    var shapedObject = new ExpandoObject();
    var idProp = FindProperty(_properties, nameof(BaseEntity.Id));
    var entityId = (int?)idProp!.GetValue(entity) ?? 0;
    if (string.IsNullOrWhiteSpace(fieldsString))
    {
      foreach (var prop in _properties.Where(x=>!x.Name.Equals(idProp!.Name)))
      {
        if (prop.PropertyType.IsClass && prop.PropertyType != typeof(string))
        {
          var innerProperties = prop.PropertyType
             .GetProperties(BindingFlags.Public | BindingFlags.Instance).ToList();
          var innerValue = prop.GetValue(entity);
          foreach (var innerProp in innerProperties)
          {
            SetValue(innerValue, shapedObject, innerProp);
          }
          continue;
        }
        SetValue(entity, shapedObject, prop);
      }
    }
    else
    {
      var fields = fieldsString.Split(',', StringSplitOptions.RemoveEmptyEntries);
      foreach (var field in fields)
      {
        var prop = FindProperty(_properties, field.Trim());
        if (prop != null)
        {
          SetValue(entity, shapedObject, prop);
        }
        foreach (var complexProp in _complexProperties)
        {
          var innerProperties = complexProp.PropertyType
               .GetProperties(BindingFlags.Public | BindingFlags.Instance).ToList();
          var innerProp = FindProperty(innerProperties, field.Trim());
          if (innerProp == null)
          {
            continue;
          }
          var innerValue = complexProp.GetValue(entity);
          SetValue(innerValue, shapedObject, innerProp);
        }
      }
    }
    return new ShapedEntity { Entity = shapedObject, Id = entityId };
  }
}
```

Step 5: Update the GlobalUsings.cs File

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

```
global using AutoLot.Api.DataShaping;
```

Step 6: Update the Program.cs File

- Add the interface and implementation to the DI container:

```
builder.Services.AddScoped(typeof(IDataShaper<>), typeof(DataShaper<>));
```

Step 7: Add the DataShapingController

- Add a new controller name `DataShapingController.cs` to the `Controllers` folder, and update the code to the following:

```
namespace AutoLot.Api.Controllers;
[ApiController]
[Route("api/[controller]")]
public class DataShapingController(
  IDriverRepo driverRepo,IDataShaper<Driver> dataShaper) : ControllerBase
{
  [HttpGet]
  [Produces("application/json")]
  public IActionResult GetFromQuery([FromQuery] string fields) =>
    Ok(dataShaper.ShapeData(driverRepo.GetAll(), fields));
  [HttpPost("{id}")]
  public IActionResult UpdateDriverFromValues(int id, [FromQuery] string values)
  {
    var convertedValues = JsonSerializer.Deserialize<Dictionary<string, string>>(values);
    var driver = driverRepo.Find(id);
    dataShaper.UpdateData(driver, convertedValues);
    driverRepo.Update(driver);
    return Ok(driver);
  }
}
```

Step 8: Test Using Bruno or CURL

- Use Bruno or CURL to test the new endpoint.:

```
CURL -X GET
https://localhost:5011/api/DataShaping?v=1.0&fields=personinformation.firstname,personinformation.
lastname,timestamp
```

Summary

This lab added Content Negotiations and Data Shaping to the RESTful service.

Next steps

In the next part of this tutorial series, you will add versioning and OpenAPI documentation.