

Lab2_PRN231

Building Book Management Application Using OData

Introduction

Imagine you're a librarian of an university, your leader has asked you to develop an application for book management especially support the querying data in many options.

The book information includes id, ISBN, title, author, price, press information and address information.

The press information includes press id, press name, the press information also has category information such as book, ebook, magazine.

The address information will have city and country.

The application has to support adding, viewing, modifying, and removing books - a standardized usage action verbs better known as Create, Read, Update, Delete (**CRUD**).

OData is an open protocol for operating data over HTTP. It also follows REST architecture. Currently OData is widely used for **exposing data**, so this is the suitable method to implement Book Management application.

➔ This lab explores creating an application using OData to create service, and ASP.NET Core Web Application with Model-View-Controller.

→ An **In-Memory Database** will be created to persist the book data that will be used for reading and managing book data by **Entity Framework Core**.

Lab Objectives

In this lab, you will:

- Use the Visual Studio.NET to create OData Service using ASP.NET Core Web Web API Project.
- Develop Web application using **MVC** Pattern.
- Use **Entity Framework Core** to create a In-Memory database
- Develop Entity classes, DbContext class, DataSource class to perform CRUD actions using Entity Framework Core
- Run the project and test the services using Postman.
- Run the project and test the application actions.

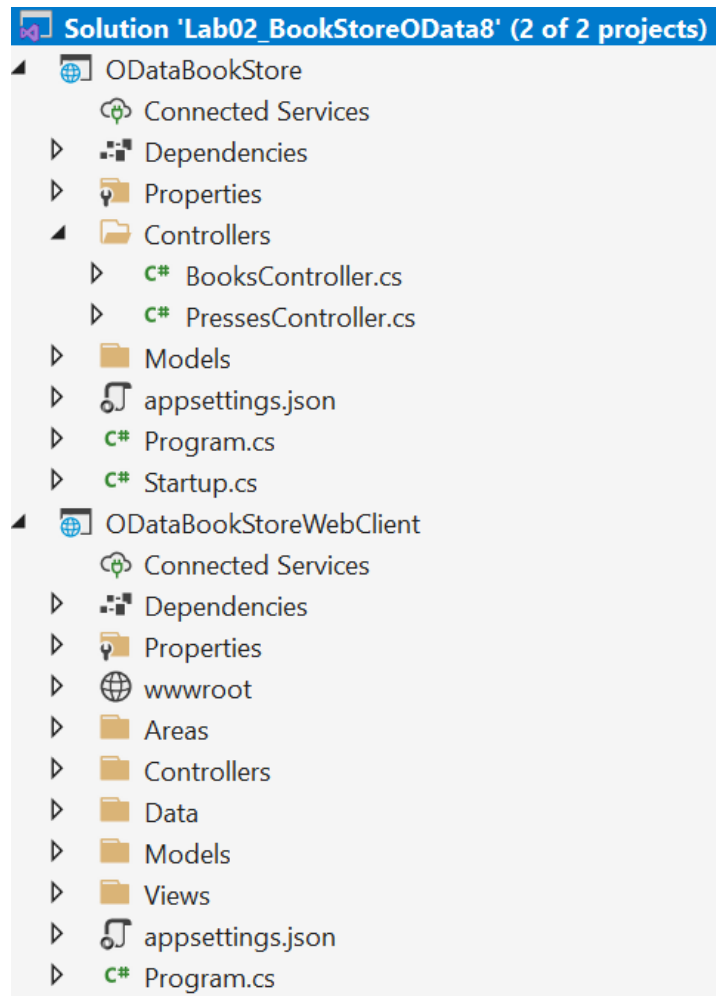
Guidelines

Activity 01: Create a Blank Solution

Step 01. Create a Solution named **No_Fullname_Lab02**.

Step 05. Create ASP.NET Core Web Web API Project for OData Service (named **No_Fullname_OdataBookStore**).

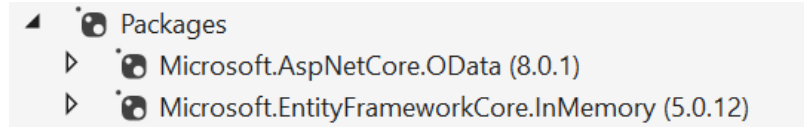
Step 06. Create ASP.NET Core Web Application (Model-View-Controller) Project (named **No_Fullname_OdataBookStoreWebClient**)



Activity 02: Creating an OData Service

Step 01. Create a skeleton of the ASP.NET Core OData service using ASP.NET Core Web API.

Step 02. Install the following packages from NuGet:



dotnet package add Microsoft.AspNetCore.OData --version 8.0.1

dotnet package add Microsoft.EntityFrameworkCore.InMemory --version 5.0.12 (for simplicity, using the version with the In-Memory data source, in the real application should use SqlServer)

Step 03. Add the model classes for building the Entity Data Model (EDM)
This steps will create 1 enum data named Category and 3 classes named Address, Press and Book.

```
public class Address
{
    3 references
    public string City { get; set; }
    3 references
    public string Street { get; set; }
}

public enum Category
{
    Book,
    Magazine,
    EBook
}

public class Press
{
    5 references
    public int Id { get; set; }
    5 references
    public string Name { get; set; }
    3 references
    public Category Category { get; set; }
}
```

```
public class Book
{
    5 references
    public int Id { get; set; }
    3 references
    public string ISBN { get; set; }
    3 references
    public string Title { get; set; }
    3 references
    public string Author { get; set; }
    3 references
    public decimal Price { get; set; }
    4 references
    public Address Location { get; set; }
    5 references
    public Press Press { get; set; }
}
```

Step 04. Build the Entity Data Model

OData uses the *Entity Data Model (EDM)* to describe the structure of data. To build the EDM add a method in *Program.cs* class.

```
0 references
public Startup(IConfiguration configuration)...
```

```
1 reference
public IConfiguration Configuration { get; }
```

```
// This method gets called by the runtime. Use this method to add services to
0 references
public void ConfigureServices(IServiceCollection services)...
```

```
// This method gets called by the runtime. Use this method to configure the HT
0 references
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)...
```

```
1 reference
private static IEdmModel GetEdmModel()
{
    ODataConventionModelBuilder builder = new ODataConventionModelBuilder();
    builder.EntitySet<Book>("Books");
    builder.EntitySet<Press>("Presses");
    return builder.GetEdmModel();
}
```

Step 05. Register the OData Services through Dependency Injection

Register the OData Services in *ConfigureServices()* of *Startup.cs*

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<BookStoreContext>(opt => opt.UseInMemoryDatabase("BookLists"));
    services.AddControllers();

    services.AddControllers().AddOData(option => option.Select().Filter()
        .Count().OrderBy().Expand().SetMaxTop(100)
        .AddRouteComponents("odata", GetEdmModel()));
}
```

Register the OData Endpoint in *Configure()* method of *Startup.cs*

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseODataBatching();

        app.UseRouting();

        // Test middleware
        app.Use(next => context =>
        {
            var endpoint = context.GetEndpoint();
            if (endpoint == null)
            {
                return next(context);
            }

            IEnumerable<string> templates;
            IODataRoutingMetadata metadata =
                endpoint.Metadata.GetMetadata<IODataRoutingMetadata>();
            if (metadata != null)
            {
                templates = metadata.Template.GetTemplates();
            }

            return next(context);
        });

        app.UseAuthorization();

        app.UseEndpoints(endpoints =>
        {
            endpoints.MapODataRoute("odata", GetEdmModel());
        });
    }
}
```

Note: Can use the below option *Configure()* method

`app.UseEndpoints(endpoints =>`

```
{
    endpoints.MapODataRoute("odata", "odata", GetEdmModel());
});
```

Step 06. Create the Data Source

Create the data context class (this class extends DbContext class)

```
public class BookStoreContext : DbContext
{
    0 references
    public BookStoreContext(DbContextOptions<BookStoreContext> options)
        : base(options)
    {
    }

    9 references
    public DbSet<Book> Books { get; set; }
    3 references
    public DbSet<Press> Presses { get; set; }

    0 references
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Book>().OwnsOne(c => c.Location);
    }
}
```

Add service with In-Memory Database to ConfigureServices() method of Startup.cs

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<BookStoreContext>(opt => opt.UseInMemoryDatabase("BookLists"));
    services.AddControllers();

    services.AddControllers().AddOData(option => option.Select().Filter()
        .Count().OrderBy().Expand().SetMaxTop(100)
        .AddRouteComponents("odata", GetEdmModel()));
}
```

Add sample data


```
public static class DataSource
{
    7 references
    private static IList<Book> listBooks { get; set; }
    2 references
    public static IList<Book> GetBooks()
    {
        if (listBooks != null)
        {
            return listBooks;
        }
        listBooks = new List<Book>();
        Book book = new Book
        {
            Id = 1,
            ISBN = "978-0-321-87758-1",
            Title = "Essential C#5.0",
            Author = "Mark Michaelis",
            Price = 59.99m,
            Location = new Address {
                City = "HCM City",
                Street = "D2, Thu Duc District" },
            Press = new Press
            {
                Id = 1,
                Name = "Addison-Wesley",
                Category = Category.Book
            }
        };
        listBooks.Add(book);
        book = new Book...;
        listBooks.Add(book);
        book = new Book...;
        listBooks.Add(book);
        return listBooks;
    }
}
```

Step 07. Add Controllers

Create 2 Controllers named *BooksController*, *PressesController* extends *ApiController*.

```
public class BooksController : ODataController
{
    private BookStoreContext db;

    0 references
    public BooksController(BookStoreContext context) {...}

    [EnableQuery(PageSize = 1)]
    0 references
    public IActionResult Get() {...}

    [EnableQuery]
    0 references
    public IActionResult Get(int key, string version) {...}

    [EnableQuery]
    0 references
    public IActionResult Post([FromBody]Book book) {...}

    [EnableQuery]
    0 references
    public IActionResult Delete([FromBody]int key) {...}
}
```

The details of constructor and functions in BooksController.

```
public BooksController(BookStoreContext context)
{
    db = context;
    db.ChangeTracker.QueryTrackingBehavior = QueryTrackingBehavior.NoTracking;
    if (context.Books.Count() == 0)
    {
        foreach (var b in DataSource.GetBooks())
        {
            context.Books.Add(b);
            context.Presses.Add(b.Press);
        }
        context.SaveChanges();
    }
}
```

```
[EnableQuery(Pagesize = 1)]
```

0 references

```
public IActionResult Get()
{
    return Ok(db.Books);
}
```

```
[EnableQuery]
```

0 references

```
public IActionResult Get(int key, string version)
{
    return Ok(db.Books.FirstOrDefault(c => c.Id == key));
}
```

```
[EnableQuery]
```

0 references

```
public IActionResult Post([FromBody]Book book)
{
    db.Books.Add(book);
    db.SaveChanges();
    return Created(book);
}
```

```
[EnableQuery]
```

0 references

```
public IActionResult Delete([FromBody]int key)
{
    Book b = db.Books.FirstOrDefault(c => c.Id == key);
    if (b == null)
    {
        return NotFound();
    }

    db.Books.Remove(b);
    db.SaveChanges();
    return Ok();
}
```

The details of functions in PressesController.

```
public class PressesController : ODataController
{
    private BookStoreContext db;

    0 references
    public PressesController(BookStoreContext context)
    {
        db = context;
        if (context.Books.Count() == 0)
        {
            foreach (var b in DataSource.GetBooks())
            {
                context.Books.Add(b);
                context.Presses.Add(b.Press);
            }
            context.SaveChanges();
        }
    }

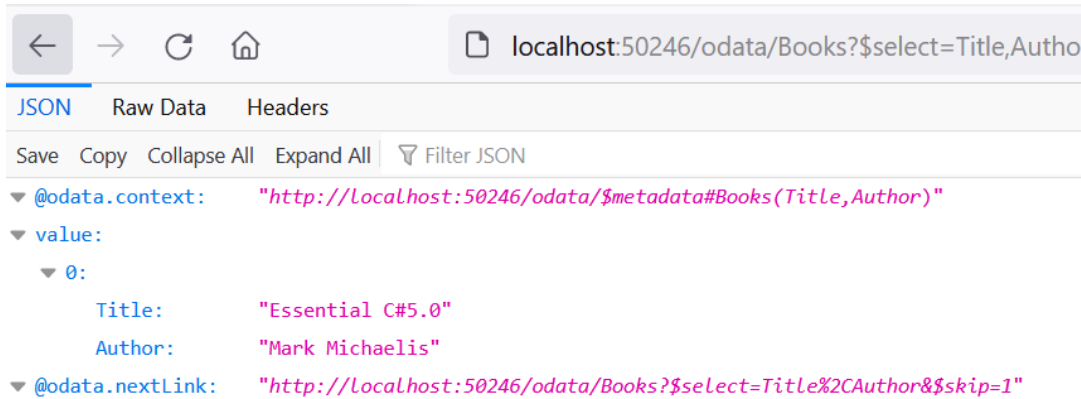
    [EnableQuery]
    0 references
    public IActionResult Get()
    {
        return Ok(db.Presses);
    }
}
```

Activity 03: Testing OData Service

Step 01. Test \$select option

The \$select system query option allows clients to request a specific set of properties for each entity or complex type. The set of properties will be comma-separated while requesting.

[http://localhost:50246/odata/Books?\\$select=Title,Author](http://localhost:50246/odata/Books?$select=Title,Author)



Step 02. Test \$filter option

The \$filter filters data based on a boolean condition. The following are conditional operators that have to be used in URLs.

- *eq* - equals to.
- *ne* - not equals to
- *gt* - greater than
- *ge* - greater than or equal
- *lt* - less than
- *le* - less than or equal

[http://localhost:50246/odata/Books?\\$filter=Title eq 'Enterprise Games'](http://localhost:50246/odata/Books?$filter=Title eq 'Enterprise Games')

← → ↻ 🏠 localhost:50246/odata/Books?\$filter=Title eq 'Enterprise Games'

JSON Raw Data Headers

Save Copy Collapse All Expand All Filter JSON

@odata.context: "http://localhost:50246/odata/\$metadata#Books"

▼ value:

▼ 0:

Id: 2

ISBN: "063-6-920-02371-5"

Title: "Enterprise Games"

Author: "Michael Hugos"

Price: 49.99

▼ Location:

City: "Bellevue"

Street: "Main St."

Step 03. Test \$orderby option

The \$orderby sorts the data using 'asc' and 'desc' keywords. We can do sorting on multiple properties using comma separation.

http://localhost:50246/odata/Presses?\$orderby=Name desc

← → ↻ 🏠 localhost:50246/odata/Presses?\$orderby=Name desc

JSON Raw Data Headers

Save Copy Collapse All Expand All Filter JSON

@odata.context: "http://localhost:50246/odata/\$metadata#Presses"

▼ value:

▼ 0:

Id: 2

Name: "O'Reilly"

Category: "EBook"

▼ 1:

Id: 3

Name: "Apress"

Category: "EBook"

▼ 2:

Id: 1

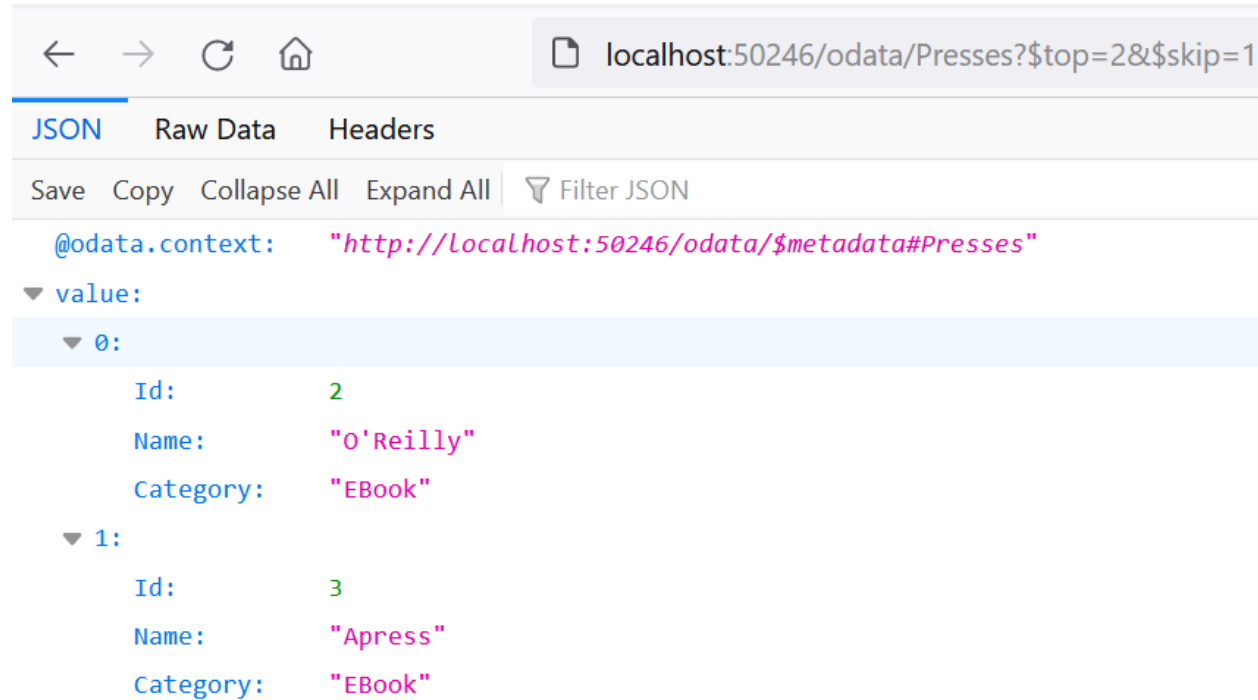
Name: "Addison-Wesley"

Category: "Book"

Step 04. Test \$top and \$skip options

The \$top fetches specified the count of top records in the collection. So to work this operator, we must specify an extension method like 'SetMaxTo(specify_max_number)'.

[http://localhost:50246/odata/Presses?\\$top=2&\\$skip=1](http://localhost:50246/odata/Presses?$top=2&$skip=1)



```

@odata.context: "http://localhost:50246/odata/$metadata#Presses"
value:
  0:
    Id: 2
    Name: "O'Reilly"
    Category: "EBook"
  1:
    Id: 3
    Name: "Apress"
    Category: "EBook"
  
```

Activity 04: ASP.NET Core Web Application with Model-View-Controller Project to get data from OData Service

Step 01. Create ASP.NET Core Web App (Model-View-Controller) named ODataBookStoreWebClient

Step 02. Create Controller to connect to OData Service

- Use HttpClient, C# HttpClient creates HTTP requests.
- The GetAsync method sends a GET request to the specified Uri as an asynchronous operation. The await operator suspends the evaluation of the enclosing async method until the asynchronous operation completes. When the asynchronous operation completes, the await operator returns the result of the operation, if any.

```
public class BookController : Controller
{
    private readonly HttpClient client = null;
    private string ProductApiUrl = "";
    0 references
    public BookController ()
    {
        client = new HttpClient();
        var contentType = new MediaTypeWithQualityHeaderValue("application/json");
        client.DefaultRequestHeaders.Accept.Add(contentType);
        ProductApiUrl = "http://localhost:50246/odata/Books";
    }

    public async Task<IActionResult> Index()
    {
        HttpResponseMessage response = await client.GetAsync(ProductApiUrl);
        string strData = await response.Content.ReadAsStringAsync();

        dynamic temp = JObject.Parse(strData);
        var lst = temp.value;
        List<Book> items = ((JArray)temp.value).Select(x => new Book
        {
            Id = (int)x["Id"],
            Author = (string)x["Author"],
            ISBN = (string)x["ISBN"],
            Title = (string)x["Title"],
            Price = (decimal)x["Price"],
        }).ToList();

        return View(items);
    }

    public ActionResult Details(int id)
    0 references
    public ActionResult Create()
    [HttpPost]
    [ValidateAntiForgeryToken]
    0 references
    public ActionResult Create(IFormCollection collection)
    0 references
    public ActionResult Edit(int id)
    [HttpPost]
    [ValidateAntiForgeryToken]
    0 references
    public ActionResult Edit(int id, IFormCollection collection)
    0 references
    public ActionResult Delete(int id)
    [HttpPost]
    [ValidateAntiForgeryToken]
    0 references
    public ActionResult Delete(int id, IFormCollection collection)
}
```



```
public class PressController : Controller
{
    private readonly HttpClient client = null;
    private string ProductApiUrl = "";

    0 references
    public PressController()
    {
        client = new HttpClient();
        var contentType = new MediaTypeWithQualityHeaderValue("application/json");
        client.DefaultRequestHeaders.Accept.Add(contentType);
        ProductApiUrl = "http://localhost:50246/odata/Presses";
    }

    // GET: BookController
    0 references
    public async Task<IActionResult> Index()
    {
        HttpResponseMessage response = await client.GetAsync(ProductApiUrl);
        string strData = await response.Content.ReadAsStringAsync();

        dynamic temp = JObject.Parse(strData);
        var lst = temp.value;
        List<Press> items = ((JArray)temp.value).Select(x => new Press
        {
            Id = (int)x["Id"],
            Name = (string)x["Name"],
        }).ToList();

        return View(items);
    }
}
```

Step 03. Create View

@model IEnumerable<BookStore0Data8.Models.Book>

@{

ViewData["Title"] = "Index";

}

<h1>Book List</h1>

<p>

<a asp-action="Create">Create New

@ViewBag.StringJSON

</p>

<table class="table">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.Id)

</th>

<th>

@Html.DisplayNameFor(model => model.Title)

</th>

<th>

@Html.DisplayNameFor(model => model.ISBN)

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.Id)

</td>

<td>

@Html.DisplayFor(modelItem => item.Title)

</td>

<td>

@Html.DisplayFor(modelItem => item.ISBN)

</td>

<td>

@Html.ActionLink("Edit", "Edit", new { /* id=item.PrimaryKey */ }) |

@Html.ActionLink("Details", "Details", new { /* id=item.PrimaryKey */ }) |

@Html.ActionLink("Delete", "Delete", new { /* id=item.PrimaryKey */ })

</td>





</tr>

}

</tbody>

</table>

Step 04. Test the function of Web Client

 <https://localhost:44353/Book/Index>

ODataBookStoreWebClient Home Privacy Hello admin@fbookstore.com! Logout

Book List

[Create New](#)

Id	Title	ISBN	
1	Essential C#5.0	978-0-321-87758-1	Edit Details Delete
2	Enterprise Games	063-6-920-02371-5	Edit Details Delete
3	Unity Game Programming	063-6-920-02371-9	Edit Details Delete

© 2022 - ODataBookStoreWebClient - [Privacy](#)

Activity 05: Build and run Project. Test all CRUD actions

Note: Choose the option for multiple startup projects.

Solution 'Lab02_BookStoreOData8' Property Pages

Configuration: N/A Platform: N/A Configuration Manager...

Common Properties
Startup Project
Project Dependencies
Code Analysis Settings
Debug Source Files
Configuration Properties

☐ Current selection

☐ Single startup project
ODataBookStore

☒ Multiple startup projects:

Project	Action
ODataBookStore	Start
ODataBookStoreWebClient	Start