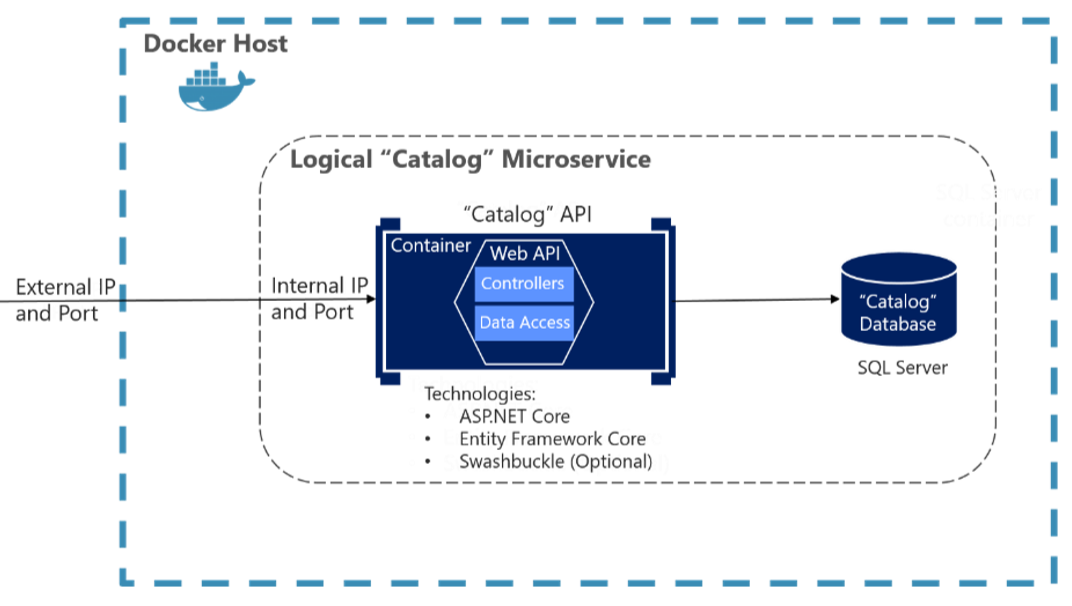
**Sample Microservice Based Application**

# 

# Creating a simple data-driven CRUD microservice

An example of this kind of simple data-drive service is the catalog microservice from the sample application.

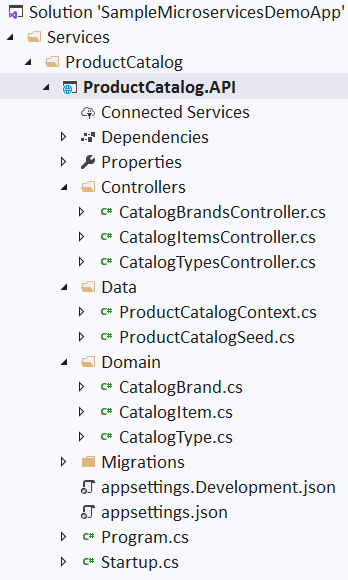
This type of service implements all its functionality in a single ASP.NET Core Web API project that includes classes for its data model, its business logic, and its data access code. It also stores its related data in a database running in SQL Server (as another container for dev/test purposes), but could also be any regular SQL Server host.



**Microservice Code Responsibilities**

* Incoming Requests
  1. HTTP request Messages
  2. RESTful = URL + HTTP Method + Data(json)
* Domain Logic
  1. Business Rules
  2. e.g. Sales Tax / GST Calculation
* Data Access
  1. Queries and Updates
* Integrate
  1. Publishing messages
  2. Third party services

**ProductCatalog Monolithic – Single Layer**



1. Visual Studio 🡪 File 🡪 New Project 🡪 Blank Solution 🡪 Next
2. Project name = MyMicroserviceDemoSolution 🡪 Create
3. Go to Windows Explorer and create d:\DemoSolution\**Services\ProductCatalog** folder
4. Visual Studio 🡪 Solution Explorer 🡪 Right on Solution 🡪 Add Solution Folder 🡪 Services
5. Visual Studio 🡪 Solution Explorer 🡪 Right on Solution 🡪 Add Solution Folder 🡪 **ProductCatalog**
6. Right Click on **Services** folder 🡪 Add New Project 🡪 ASP.NET Core Web Application 🡪 Project Name = **ProductCatalog.API**, Location = d:\DemoSolution\**Services** 🡪 Template = Web API 🡪 Create

**Adding Domain Classes**

1. Add following files and folders from Sample Application
   1. Domain\CatalogType.cs

public class CatalogType

{

public int Id { get; set; }

public string Type { get; set; }

}

* 1. Domain\CatalogBrand.cs

public class CatalogBrand

{

public int Id { get; set; }

public string Brand { get; set; }

}

* 1. Domain\CatalogItem.cs

public class CatalogItem

{

public int Id { get; set; }

public string Name { get; set; }

public string Description { get; set; }

public decimal Price { get; set; }

public string PictureFileName { get; set; }

public int CatalogTypeId { get; set; }

public int CatalogBrandId { get; set; }

public **CatalogBrand** CatalogBrand { get; set; }

public **CatalogType** CatalogType { get; set; }

}

1. Build the application
2. Go to **Solution Explorer**, right click on the Controllers Folder 🡪 **Add** 🡪 **New Scaffolded Item**. 🡪 **API Controller with actions, using Entity Framework** 🡪 Add
3. Select the Model class = "CatalogItem", Data context class: Click +, New data context type: ProductCatalogContext 🡪 Add
4. Examine the code generated.

**Examine the context generated context class: Data/ProductCatalogApi.cs**

using Microsoft.EntityFrameworkCore;

namespace ProductCatalogApi.Models

{

public class ProductCatalogContext : DbContext

{

public ProductCatalogContext (DbContextOptions<ProductCatalogContext> options)

: base(options)

{

}

public DbSet<ProductCatalogApi.Domain.CatalogBrand> CatalogBrands { get; set; }

public DbSet<ProductCatalogApi.Domain.CatalogType> CatalogTypes { get; set; }

public DbSet<ProductCatalogApi.Domain.CatalogItem> CatalogItems { get; set; }

}

}

**Examine the context registered with dependency injection:**

Open **Program.cs and note the following lines of code.**

services.AddDbContext<ProductCatalogContext>(options =>

options.UseSqlServer(Configuration.GetConnectionString("ProductCatalogContext")));

**Examine the appsettings.json file**

"ConnectionStrings": {

"ProductCatalogContext": "Server=.\\sqlexpress;Database=ProductCatalogDb;Trusted\_Connection=True;MultipleActiveResultSets=true"

}

1. **In Controller:** Edit **GetCatalogItem** Method as below

public async Task<ActionResult<IEnumerable<CatalogItem>>> **GetCatalogItem**()

{

return await \_context.CatalogItems.Include("CatalogType").Include("CatalogBrand").ToListAsync();

}

[HttpGet("{id}")]

public async Task<ActionResult<CatalogItem>> **GetCatalogItem**(int id)

{

var catalogItem = await \_context.CatalogItems.Include("CatalogType").Include("CatalogBrand").Where(item => item.Id == id).FirstOrDefaultAsync();

if (catalogItem == null)

{

return NotFound();

}

return catalogItem;

}

1. Goto Package Manager Console

**Add-Migration "Intial Script"**

**Initialize with Seed Data**

1. Data\ProductCatalogSeed.cs

namespace ProductCatalog.API.Domain

{

public class ProductCatalogSeed

{

public static async Task SeedAsync(ProductCatalogContext context)

{

if (!context.CatalogBrands.Any())

{

context.CatalogBrands.AddRange(GetPreconfiguredCatalogBrands());

await context.SaveChangesAsync();

}

if (!context.CatalogTypes.Any())

{

context.CatalogTypes.AddRange(GetPreconfiguredCatalogTypes());

await context.SaveChangesAsync();

}

if (!context.CatalogItems.Any())

{

context.CatalogItems.AddRange(GetPreconfiguredItems());

await context.SaveChangesAsync();

}

}

static IEnumerable<CatalogBrand> GetPreconfiguredCatalogBrands()

{

return new List<CatalogBrand>()

{

new CatalogBrand() { Brand = "Addidas"},

new CatalogBrand() { Brand = "Puma" },

new CatalogBrand() { Brand = "Nike" }

};

}

static IEnumerable<CatalogType> GetPreconfiguredCatalogTypes()

{

return new List<CatalogType>()

{

new CatalogType() { Type = "Running"},

new CatalogType() { Type = "Basketball" },

new CatalogType() { Type = "Tennis" },

};

}

static IEnumerable<CatalogItem> GetPreconfiguredItems()

{

return new List<CatalogItem>()

{

new CatalogItem() {PictureFileName="demo.jpg",CatalogTypeId=2,CatalogBrandId=3, Description = "Shoes for next century", Name = "World Star", Price = 199.5M },

new CatalogItem() {PictureFileName="demo.jpg",CatalogTypeId=1,CatalogBrandId=2, Description = "Will make you world champions", Name = "White Line", Price= 88.50M },

new CatalogItem() {PictureFileName="demo.jpg",CatalogTypeId=2,CatalogBrandId=3, Description = "You have already won gold medal", Name = "Prism White Shoes", Price = 129},

new CatalogItem() {PictureFileName="demo.jpg",CatalogTypeId=2,CatalogBrandId=2, Description = "Olympic runner", Name = "Foundation Hitech", Price = 12 },

new CatalogItem() {PictureFileName="demo.jpg",CatalogTypeId=2,CatalogBrandId=1, Description = "Roslyn Red Sheet", Name = "Roslyn White", Price = 188.5M },

new CatalogItem() {PictureFileName="demo.jpg",CatalogTypeId=2,CatalogBrandId=2, Description = "Lala Land", Name = "Blue Star", Price = 112},

new CatalogItem() {PictureFileName="demo.jpg",CatalogTypeId=2,CatalogBrandId=1, Description = "High in the sky", Name = "Roslyn Green", Price = 212},

new CatalogItem() {PictureFileName="demo.jpg",CatalogTypeId=1,CatalogBrandId=1, Description = "Light as carbon", Name = "Deep Purple", Price = 238.5M },

new CatalogItem() {PictureFileName="demo.jpg",CatalogTypeId=1,CatalogBrandId=2, Description = "High Jumper", Name = "Addidas<White> Running", Price = 129 },

new CatalogItem() {PictureFileName="demo.jpg",CatalogTypeId=2,CatalogBrandId=3, Description = "Dunker", Name = "Elequent", Price = 12},

new CatalogItem() {PictureFileName="demo.jpg",CatalogTypeId=1,CatalogBrandId=2, Description = "All round", Name = "Inredeible", Price = 248.5M },

new CatalogItem() {PictureFileName="demo.jpg",CatalogTypeId=2,CatalogBrandId=1, Description = "Pricesless", Name = "London Sky", Price = 412 },

new CatalogItem() {PictureFileName="demo.jpg",CatalogTypeId=3,CatalogBrandId=3, Description = "Tennis Star", Name = "Elequent", Price = 123 },

new CatalogItem() {PictureFileName="demo.jpg",CatalogTypeId=3,CatalogBrandId=2, Description = "Wimbeldon", Name = "London Star", Price = 218.5M},

new CatalogItem() {PictureFileName="demo.jpg",CatalogTypeId=3,CatalogBrandId=1, Description = "Rolan Garros", Name = "Paris Blues", Price = 312 }

};

}

}

}

1. Edit the Code in Main as below

var app = builder.Build();

using (var scope = app.Services.CreateScope())

{

var services = scope.ServiceProvider;

var context = services.GetRequiredService<ProductCatalogContext>();

context.Database.Migrate();

ProductCatalogSeed.SeedAsync(context).Wait();

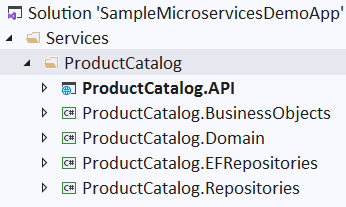
}

1. Navigate to [http://localhost:1234/**swagger/v1**/swagger.json](http://localhost:1234/swagger/v1/swagger.json), open the document and view in VS.NET
2. Navigate to <http://localhost:1234/swagger> and you will find Swagger UI has been enabled on the API.
3. Using the Swagger UI, we can test the API created.

**ProductCatalog Monolithic – Multi Layer**

**Splitting into Multiple Projects**

1. Add the **Standard Class Library** Projects as shown below



1. **Following references must be added to each project**

**ProductCatalog.Repositories** should refer to

* + ProductCatalog.Domain

**ProductCatalog.EFRepositories** should refer to

* + ProductCatalog.Repositories
    - ProductCatalog.Domain

**ProductCatalog.BusinessObjects** should refer to

* + ProductCatalog.Repositories
    - ProductCatalog.Domain

**ProductCatalog.API** should refer to

* + ProductCatalog.BusinessObjects
    - ProductCatalog.Domain
  + ProductCatalog.Repositories
    - ProductCatalog.Domain
  + ProductCatalog.EFRepositories
    - ProductCatalog.Repositories
      1. ProductCatalog.Domain

1. Move the following from **ProductCatalog.API** project to **ProductCatalog.Domain** Project and change the namespace accordingly
   * CatalogBrand.cs
   * CatalogType.cs
   * CatalogItem.cs
   * ProductCatalogContext.cs
   * ProductCatalogSeed.cs
2. Delete the Migration folder from the API project and also Drop the database using SQL Server Object Explorer
3. To the project **ProductCatalog.Repositories** add interface **ICatalogItemRepository**

public interface ICatalogItemRepository

{

Task<IEnumerable<CatalogItem>> GetCatalogItems();

Task<CatalogItem> GetCatalogItemDetails(int id);

Task<CatalogItem> Add(CatalogItem item);

Task Update(CatalogItem item);

Task Delete(int id);

}

1. To the project **ProductCatalog.EFRepositories** add interface **CatalogItemRepository**

public class CatalogItemRepository : ICatalogItemRepository

{

ProductCatalogContext \_context;

public CatalogItemRepository(ProductCatalogContext context)

{

\_context = context;

}

public async Task<CatalogItem> Add(CatalogItem item)

{

\_context.CatalogItems.Add(item);

await \_context.SaveChangesAsync();

return item;

}

public async Task Delete(int id)

{

var catalogItem = await \_context.CatalogItems.FindAsync(id);

if (catalogItem == null)

{

throw new ApplicationException("Not Found");

}

\_context.CatalogItems.Remove(catalogItem);

await \_context.SaveChangesAsync();

}

public async Task<CatalogItem> GetCatalogItemDetails(int id)

{

var catalogItem = await \_context.CatalogItems.Include("CatalogType").Include("CatalogBrand").FirstAsync(item => item.Id == id);

if (catalogItem == null)

{

throw new ApplicationException("Not Found");

}

return catalogItem;

}

public async Task<IEnumerable<CatalogItem>> GetCatalogItems()

{

return await \_context.CatalogItems.Include("CatalogType").Include("CatalogBrand").ToListAsync();

}

public async Task Update(CatalogItem item)

{

\_context.Entry(item).State = EntityState.Modified;

await \_context.SaveChangesAsync();

}

}

1. To the project **ProductCatalog.BusinessObjects** add interface **ICatalogItemBO and CatalogItemBO**

public interface ICatalogItemBO

{

Task<IEnumerable<CatalogItem>> GetCatalogItems();

Task<CatalogItem> GetCatalogItemDetails(int id);

Task<CatalogItem> Add(CatalogItem item);

Task Update(CatalogItem item);

Task Delete(int id);

}

public class CatalogItemBO : ICatalogItemBO

{

ICatalogItemRepository \_repository;

public CatalogItemBO(ICatalogItemRepository repository)

{

\_repository = repository;

}

public async Task<CatalogItem> Add(CatalogItem item)

{

await \_repository.Add(item);

return item;

}

public async Task Delete(int id)

{

await \_repository.Delete(id);

}

public async Task<CatalogItem> GetCatalogItemDetails(int id)

{

return await \_repository.GetCatalogItemDetails(id);

}

public async Task<IEnumerable<CatalogItem>> GetCatalogItems()

{

return await \_repository.GetCatalogItems();

}

public async Task Update(CatalogItem item)

{

await \_repository.Update(item);

}

}

1. To the project **ProductCatalog.BusinessObjects** add interface **ICatalogItemBO and CatalogItemBO**
2. Add the following to **ProductCatalog.API\Startup.cs -> ConfigureService Method**

services.AddTransient<ICatalogItemBO, CatalogItemBO>();

services.AddTransient<ICatalogItemRepository, CatalogItemRepository>();

1. Edit **ProductCatalog.API\Controllers\CatalogItemController.cs** as below

[Route("[controller]")]

[ApiController]

public class CatalogItemsController : ControllerBase

{

ICatalogItemBO \_boCatalogItem;

public CatalogItemsController(ICatalogItemBO boCatalogItem)

{

// \_context = context;

\_boCatalogItem = boCatalogItem;

}

[HttpGet]

public async Task<ActionResult<IEnumerable<CatalogItem>>> GetCatalogItem()

{

var items = await \_boCatalogItem.GetCatalogItems(); ;

return Ok(items);

}

[HttpGet("{id}")]

public async Task<ActionResult<CatalogItem>> GetCatalogItem(int id)

{

var catalogItem = await \_boCatalogItem.GetCatalogItemDetails(id);

if (catalogItem == null)

{

return NotFound();

}

return catalogItem;

}

[HttpPut("{id}")]

public async Task<IActionResult> PutCatalogItem(int id, CatalogItem catalogItem)

{

if (id != catalogItem.Id)

{

return BadRequest();

}

try

{

await \_boCatalogItem.Update(catalogItem);

}

catch (ApplicationException ex)

{

if (ex.Message == "Not Found")

{

return NotFound();

}

else

{

throw;

}

}

return NoContent();

}

[HttpPost]

[ProducesResponseType(StatusCodes.Status201Created)]

public async Task<ActionResult<CatalogItem>> PostCatalogItem(CatalogItem catalogItem)

{

await \_boCatalogItem.Add(catalogItem);

return CreatedAtAction("GetCatalogItem", new { id = catalogItem.Id }, catalogItem);

}

[HttpDelete("{id}")]

public async Task DeleteCatalogItem(int id)

{

await \_boCatalogItem.Delete(id);

}

}

1. Exclude from project **CatalogTypeController.cs and CatalogBrandController.cs** files as they are not yet updated
2. Go to Menu Tools 🡪 NuGet Package Manager 🡪 **NuGet Package Manager Console**
3. In package Manage Window 🡪 Change Defaut project to **ProductCatalog.Domain** and execute following command

Add-Migration "Initial Database"

1. Run the Application and Test the o/p in Swagger UI

**~~Writing Generic Repository~~**

1. ~~Add the following to~~ **~~ProductCatalog.Repository\IGenericRepository.cs~~**

~~public interface IGenericRepository<T> where T:class~~

~~{~~

~~Task<IEnumerable<T>> GetAll();~~

~~Task<T> GetById(int id);~~

~~Task<T> Add(T item);~~

~~Task Update(T item);~~

~~Task Delete(int id);~~

~~}~~

1. ~~Add the following to~~ **~~ProductCatalog.EFRepository~~**~~\~~**~~GenericRepository.cs~~**

~~using Microsoft.EntityFrameworkCore;~~

~~using ProductCatalog.Repositories;~~

~~using System;~~

~~using System.Collections.Generic;~~

~~using System.Text;~~

~~using System.Threading.Tasks;~~

~~namespace ProductCatalog.EFRepositories~~

~~{~~

~~public class GenericRepository<T> : IGenericRepository<T> where T : class~~

~~{~~

~~private readonly DbContext \_context;~~

~~DbSet<T> dbSet;~~

~~public GenericRepository(DbContext context)~~

~~{~~

~~\_context = context;~~

~~dbSet = \_context.Set<T>();~~

~~}~~

~~public async virtual Task<T> Add(T item)~~

~~{~~

~~dbSet.Add(item);~~

~~await \_context.SaveChangesAsync();~~

~~return item;~~

~~}~~

~~public async virtual Task Delete(int id)~~

~~{~~

~~T entity = await dbSet.FindAsync(id);~~

~~dbSet.Remove(entity);~~

~~await \_context.SaveChangesAsync();~~

~~}~~

~~public async virtual Task<IEnumerable<T>> GetAll()~~

~~{~~

~~return await dbSet.ToListAsync<T>();~~

~~}~~

~~public async virtual Task<T> GetById(int id)~~

~~{~~

~~return await dbSet.FindAsync(id);~~

~~}~~

~~public async virtual Task Update(T item)~~

~~{~~

~~\_context.Entry<T>(item).State = EntityState.Modified;~~

~~await \_context.SaveChangesAsync();~~

~~}~~

~~}~~

~~}~~

1. ~~Add the following Interfaces to~~ **~~ProductCatalog~~**~~.~~**~~Repository~~** ~~Project~~

**~~ICatalogTypeRepository.cs~~**

~~public interface ICatalogTypeRepository : IGenericRepository<CatalogType>~~

~~{ }~~

**~~ICatalogBrandRepository.cs~~**

~~public interface ICatalogBrandRepository : IGenericRepository<CatalogBrand>~~

~~{ }~~

**~~ICatalogItemRepository.cs~~**

~~public interface ICatalogItemRepository : IGenericRepository<CatalogItem>~~

~~{ }~~

1. ~~Add the following to~~ **~~ProductCatalog.EFRepository~~**~~\~~**~~CatalogItemRepository.cs~~**

~~public class CatalogItemRepository : GenericRepository<CatalogItem>, ICatalogItemRepository~~

~~{~~

~~private readonly ProductCatalogContext \_context;~~

~~public CatalogItemRepository(ProductCatalogContext context) : base (context)~~

~~{~~

~~\_context = context;~~

~~}~~

~~public async~~**~~override~~**~~Task<IEnumerable<CatalogItem>> GetAll()~~

~~{~~

~~return await \_context.CatalogItems.Include("CatalogType").Include("CatalogBrand").ToListAsync();~~

~~}~~

~~public async~~**~~override~~**~~Task<CatalogItem> GetById(int id)~~

~~{~~

~~return await \_context.CatalogItems.Include("CatalogType").Include("CatalogBrand").Where(item1 => item1.Id == id).FirstOrDefaultAsync<CatalogItem>();~~

~~}~~

~~}~~

1. ~~Add the following to~~ **~~ProductCatalog.EFRepository~~**~~\~~**~~CatalogTypeRepository.cs~~**

~~public class CatalogTypeRepository : GenericRepository<CatalogType>, ICatalogItemRepository~~

~~{~~

~~public CatalogTypeRepository(ProductCatalogContext context) :base(context)~~

~~{ }~~

~~}~~

1. ~~Add the following to~~ **~~ProductCatalog.EFRepository~~**~~\~~**~~CatalogBrandRepository.cs~~**

~~public class CatalogBrandRepository : GenericRepository<CatalogBrand>, ICatalogItemRepository~~

~~{~~

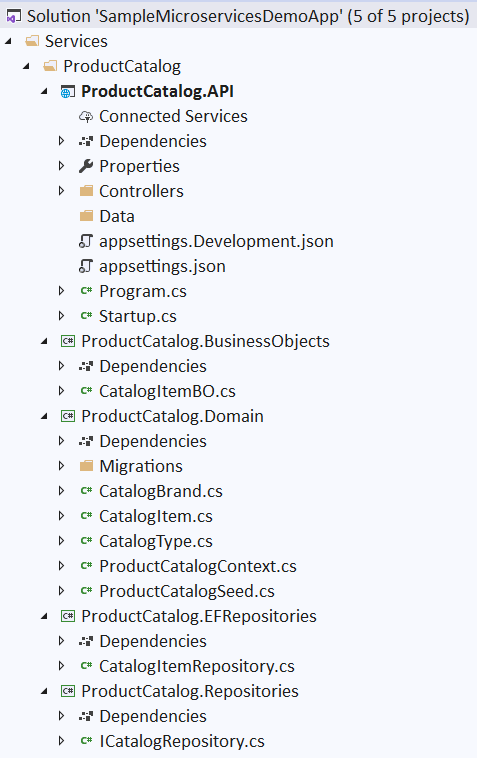
~~public CatalogBrandRepository(ProductCatalogContext context) :base(context)~~

~~{ }~~

~~}~~

1. ~~Edit BO classes and resolve errors by replacing the method names from GenericRepository~~

~~Final Layout of Projects in Solution Explorer~~

~~~~

**Web Client**

1. Right Click on Solution 🡪 Add 🡪 New Solution Folder, Name=MyWebApp
2. Right Click on WebApp 🡪 Add 🡪 New Project 🡪 ASP.NET Core Web Application 🡪 Next
3. Project Name = WebMVC 🡪 Location=D:\DemoSolution\MyWebApp 🡪 Create
4. Select Web Application (Model-View-Controller) 🡪 Create
5. Add following **Models** (Same as in ProductCatalogAPI Microservice)

public class CatalogType

{

public int Id { get; set; }

public string Type { get; set; }

}

public class CatalogBrand

{

public int Id { get; set; }

public string Brand { get; set; }

}

public class CatalogItem

{

public string Id { get; set; }

public string Name { get; set; }

public string Description { get; set; }

public decimal Price { get; set; }

public string PictureUri { get; set; }

public int CatalogBrandId { get; set; }

public CatalogBrand CatalogBrand { get; set; }

public int CatalogTypeId { get; set; }

public CatalogType CatalogType { get; set; }

}

1. Edit appsettings.json

{

. . .

"**CatalogAPIUrl**": "https://localhost:44341"

}

1. Add ICatalogService.cs

public interface ICatalogService

{

Task<IEnumerable<CatalogItem>> GetCatalogItems(int? brand, int? type);

Task<CatalogItem> GetItemDetails(int id);

Task<IEnumerable<SelectListItem>> GetBrands();

Task<IEnumerable<SelectListItem>> GetTypes();

}

1. Add Services/CatalogService.cs

public class CatalogService : ICatalogService

{

private readonly string \_remoteServiceBaseUrl;

public CatalogService(IConfiguration config)

{

\_remoteServiceBaseUrl = config["CatalogAPIUrl"];

}

public async Task<IEnumerable<CatalogItem>> **GetCatalogItems**(int? brand, int? type)

{

var client = new HttpClient();

var result = await client.GetAsync(\_remoteServiceBaseUrl + "/CatalogItems/");

var dataString = await result.Content.ReadAsStringAsync();

return JsonConvert.DeserializeObject<IEnumerable<CatalogItem>>(dataString);

}

public async Task<CatalogItem> **GetItemDetails**(int id)

{

HttpClient client = new HttpClient();

string strjson = await client.GetStringAsync(\_remoteServiceBaseUrl + "/CatalogItems/" + id);

CatalogItem items = JsonConvert.DeserializeObject<CatalogItem>(strjson);

return items;

}

public async Task<IEnumerable<SelectListItem>> **GetBrands**()

{

var client = new HttpClient();

var result = await client.GetAsync(\_remoteServiceBaseUrl + "/api/CatalogBrand/");

var dataString = await result.Content.ReadAsStringAsync();

var catalogBrands = JsonConvert.DeserializeObject<IEnumerable<CatalogItem>>(dataString);

return new SelectList(catalogBrands, "Id", "Brand");

}

public async Task<IEnumerable<SelectListItem>> **GetTypes**()

{

var client = new HttpClient();

var result = await client.GetAsync(\_remoteServiceBaseUrl + "/api/CatalogType/");

var dataString = await result.Content.ReadAsStringAsync();

var catalogTypes = JsonConvert.DeserializeObject<IEnumerable<CatalogItem>>(dataString);

return new SelectList(catalogTypes, "Id", "Type");

}

}

1. Add the following to Startup.ConfigureService so that is can be injected.

services.AddTransient<ICatalogService, CatalogService>();

1. Right Click on Controllers Folder 🡪 Add 🡪 Controller… 🡪 MVC Controller – Empty 🡪 Add 🡪 Controller name = **CatalogController**
2. Edit CatalogController.cs

public class CatalogController : Controller

{

// GET: Catalog

ICatalogService \_catalogService;

public CatalogController(**ICatalogService** catalogService)

{

\_catalogService = catalogService;

}

public async Task<ActionResult> **Index**()

{

IEnumerable<CatalogItem> items = await \_catalogService.GetCatalogItems(null, null);

return View(items);

}

// GET: Catalog/Details/5

public async Task<ActionResult> **Details**(int id)

{

CatalogItem item = await \_catalogService.GetItemDetails(id);

return View(item);

}

}

1. Right Click on **Index** method 🡪 Add View… 🡪 View name=Index, Template=**List**, Model class= CatalogItem (WebMvc.Models) 🡪 Add
2. Right Click on **Details** method 🡪 Add View… 🡪 View name=Index, Template=**Details**, Model class= CatalogItem (WebMvc.Models) 🡪 Add