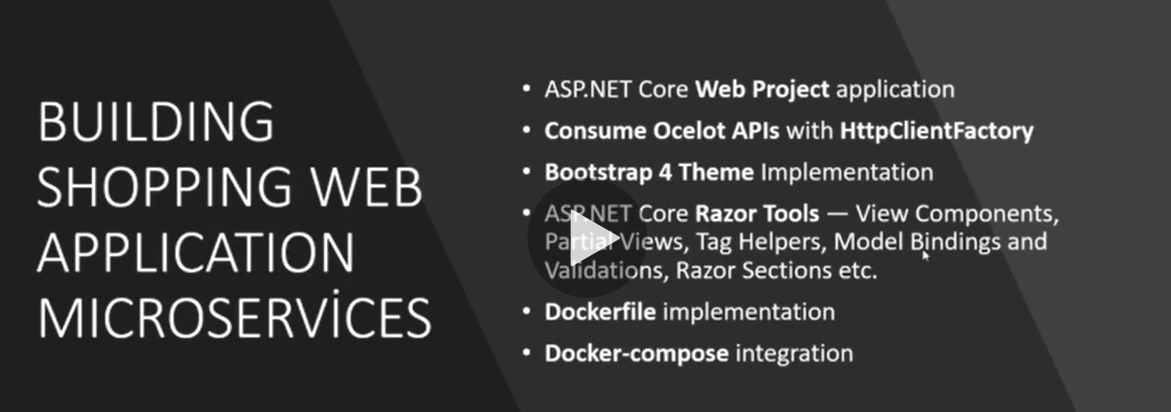
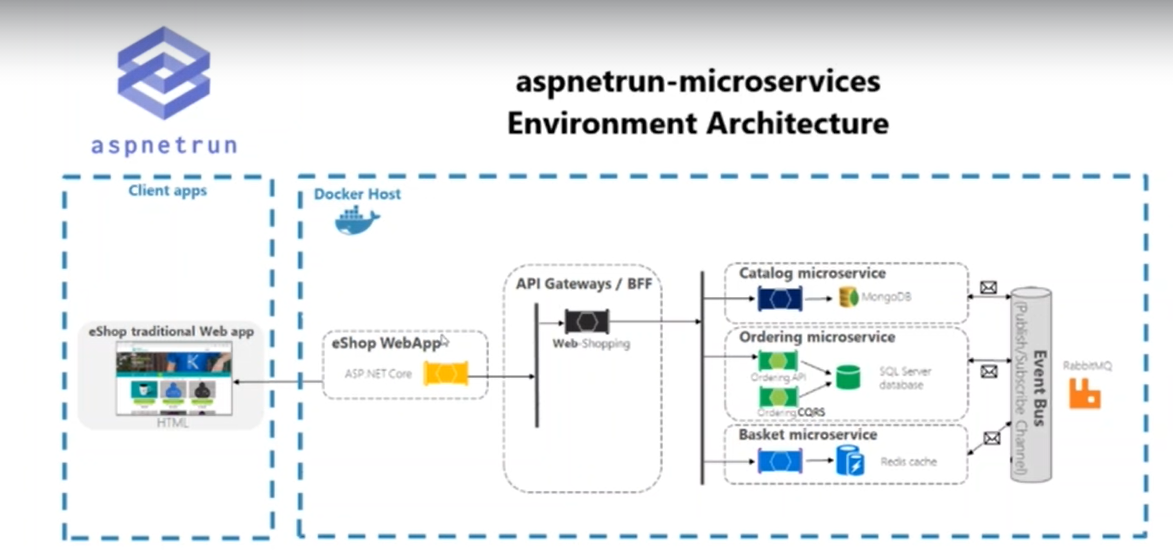
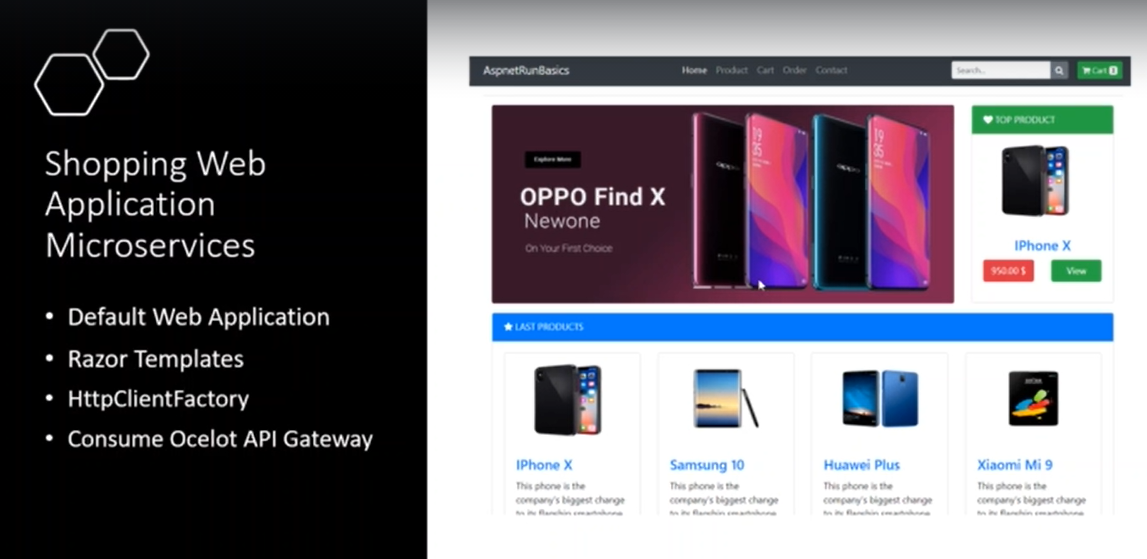
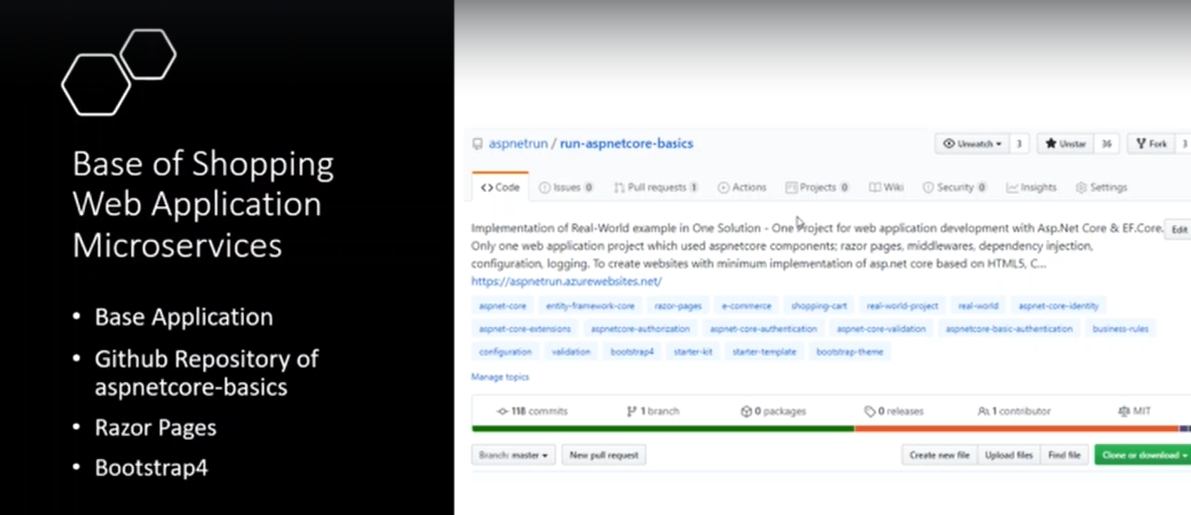
**Section08 Building Shopping Web application**

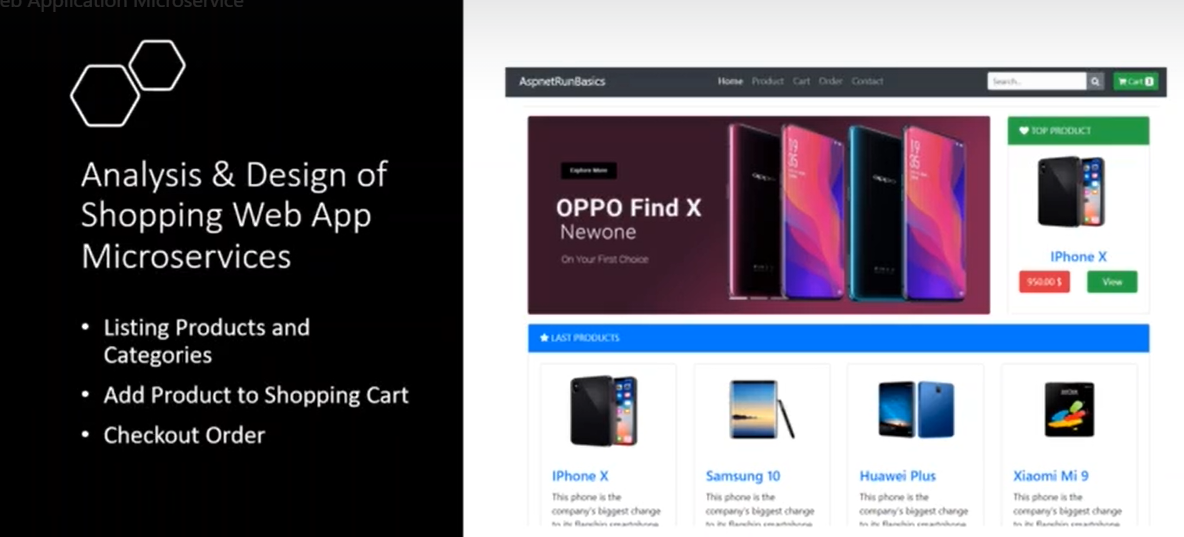
**Notes:-**

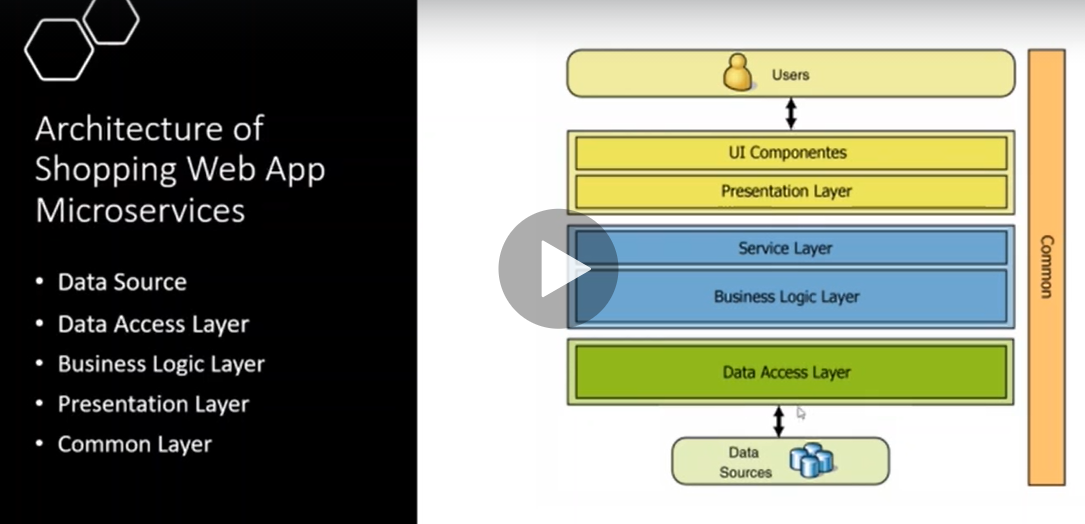




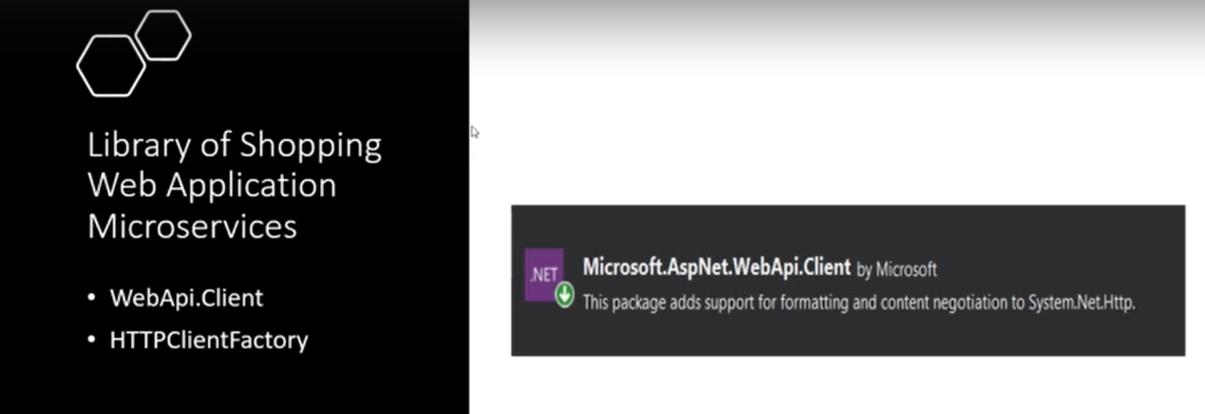








**We will use Web API client to use service factory to send http client to the API Gateway**



**Stesp:-**

**1-create src > WebApp > create new asp.net core web application**

**2-on Nuget manager > install the library Microsoft.AspNet.WebApi.Client to use the httpClientfactory to allow send messages**

**3-on the appsettings.json we set the following configuration as below**

**{//we define ApiSetting for api gateway for the Catalog,Basket,Order API'S**

**"ApiSettings": {**

**"BaseAddress": "http://localhost:7000",**

**"CatalogPath": "/Catalog",**

**"BasketPath": "/Basket",**

**"OrderPath": "/Order"},**

**"Logging": {**

**"LogLevel": {**

**"Default": "Information",**

**"Microsoft": "Warning",**

**"Microsoft.Hosting.Lifetime": "Information"}},**

**"AllowedHosts": "\*"}**

**4-we create folder settings > create interface called IApiSettings and class called ApiSettings**

**//this class represent the ApiSettings that consume the appsettings.json**

**public interface IApiSettings{**

**string BaseAddress { get; set; }**

**string CatalogPath { get; set; }**

**string BasketPath { get; set; }**

**string OrderPath { get; set; }}**

**public class ApiSettings : IApiSettings{**

**public string BaseAddress { get; set; }**

**public string CatalogPath { get; set; }**

**public string BasketPath { get; set; }**

**public string OrderPath { get; set; }}**

**5-we will unify the httpClient in central place as below**

**A-create folder called Infrastructure > create three classes**

**\*ApiBuilder.cs (to provide the prepare of generate URL and add params)**

**//to generate UriBuilder that container the full url and username and password and so on.**

**public class ApiBuilder{**

**private readonly string \_fullUrl;**

**private UriBuilder \_builder;**

**public ApiBuilder(string url){\_fullUrl = url;**

**\_builder = new UriBuilder(url);}**

**public Uri GetUri() => \_builder.Uri;**

**public ApiBuilder Scheme(string scheme){**

**\_builder.Scheme = scheme;**

**return this;}**

**public ApiBuilder Host(string host){**

**\_builder.Host = host;**

**return this;}**

**public ApiBuilder Port(int port){**

**\_builder.Port = port;**

**return this;}**

**//it will add path to the Path in the UriBuilder**

**public ApiBuilder AddToPath(string path){**

**IncludePath(path);**

**return this;}**

**//it will set path to the Api Builder**

**public ApiBuilder SetPath(string path){**

**\_builder.Path = path;**

**return this;}**

**//it will prepare the added params to the UriBuilder**

**public void IncludePath(string path){**

**if (string.IsNullOrEmpty(\_builder.Path)**

**|| \_builder.Path == "/"){**

**\_builder.Path = path;}**

**else{**

**var newPath = $"{\_builder.Path}/{path}";**

**\_builder.Path = newPath.Replace("//", "/");}}**

**public ApiBuilder Fragment(string fragment){**

**\_builder.Fragment = fragment;**

**return this;}**

**public ApiBuilder SetSubdomain(string subdomain){**

**\_builder.Host = string.Concat(subdomain, ".", new Uri(\_fullUrl).Host);**

**return this;}**

**public bool HasSubdomain(){**

**return \_builder.Uri.HostNameType == UriHostNameType.Dns**

**&& \_builder.Uri.Host.Split('.').Length > 2;}**

**public ApiBuilder AddQueryString(string name, string value){**

**var qsNv = HttpUtility.ParseQueryString(\_builder.Query);**

**qsNv[name] = string.IsNullOrEmpty(qsNv[name])**

**? value**

**: string.Concat(qsNv[name], ",", value);**

**\_builder.Query = qsNv.ToString();**

**return this;}**

**public ApiBuilder QueryString(string queryString){**

**if (!string.IsNullOrEmpty(queryString)){**

**\_builder.Query = queryString;}**

**return this;}**

**public ApiBuilder UserName(string username){**

**\_builder.UserName = username;**

**return this;}**

**public ApiBuilder Password(string password){**

**\_builder.Password = password;**

**return this;}**

**public string GetLeftPart(){**

**return \_builder.Uri.GetLeftPart(UriPartial.Path);}}**

**\*HttpRequestBuilder:**

**//to generate HttpRequest and depend on the APiBuilder to prepare request**

**public class HttpRequestBuilder{**

**private readonly HttpRequestMessage \_request;**

**private string \_baseAddress;**

**private readonly ApiBuilder \_apiBuilder;**

**public HttpRequestBuilder(string uri) : this(new ApiBuilder(uri)) { }**

**public HttpRequestBuilder(ApiBuilder apiBuilder){**

**\_request = new HttpRequestMessage();**

**\_apiBuilder = apiBuilder;**

**\_baseAddress = \_apiBuilder.GetLeftPart();}**

**public HttpRequestBuilder AddToPath(string path){**

**\_apiBuilder.AddToPath(path);**

**\_request.RequestUri = \_apiBuilder.GetUri();**

**return this;}**

**public HttpRequestBuilder SetPath(string path){**

**\_apiBuilder.SetPath(path);**

**\_request.RequestUri = \_apiBuilder.GetUri();**

**return this;}**

**public HttpRequestBuilder HttpMethod(HttpMethod httpMethod){**

**\_request.Method = httpMethod;**

**return this;}**

**public HttpRequestBuilder Headers(Action<HttpRequestHeaders> funcOfHeaders){**

**funcOfHeaders(\_request.Headers);**

**return this;}**

**public HttpRequestBuilder Headers(NameValueCollection headers){**

**\_request.Headers.Clear();**

**foreach (var item in headers.AllKeys){\_request.Headers.Add(item, headers[item]);}**

**return this;}**

**public HttpRequestBuilder Content(HttpContent content){**

**\_request.Content = content;**

**return this;}**

**public HttpRequestBuilder RequestUri(Uri uri){**

**\_request.RequestUri = new ApiBuilder(uri.ToString())**

**.GetUri();**

**return this;}**

**public HttpRequestBuilder RequestUri(string uri){return RequestUri(new Uri(uri));}**

**public HttpRequestBuilder BaseAddress(string address){**

**\_baseAddress = address;**

**return this;}**

**public HttpRequestBuilder Subdomain(string subdomain){**

**\_apiBuilder.SetSubdomain(subdomain);**

**\_request.RequestUri = \_apiBuilder.GetUri();**

**return this;}**

**public HttpRequestBuilder AddQueryString(string name, string value){**

**\_apiBuilder.AddQueryString(name, value);**

**\_request.RequestUri = \_apiBuilder.GetUri();**

**return this;}**

**public HttpRequestBuilder SetQueryString(string qs){**

**\_apiBuilder.QueryString(qs);**

**\_request.RequestUri = \_apiBuilder.GetUri();**

**return this;}**

**public HttpRequestMessage GetHttpMessage(){return \_request;}**

**public ApiBuilder GetApiBuilder(){return new ApiBuilder(\_request.RequestUri.ToString());}}**

**\*BaseHttpClientWithFactory : used to send request**

**//this class unify all the http client factory t send request all types**

**public abstract class BaseHttpClientWithFactory{**

**private readonly IHttpClientFactory \_factory;**

**public Uri BaseAddress { get; set; }**

**public string BasePath { get; set; }**

**public BaseHttpClientWithFactory(IHttpClientFactory factory) => \_factory = factory;**

**private HttpClient GetHttpClient(){return \_factory.CreateClient();}**

**public virtual async Task<T> SendRequest<T>(HttpRequestMessage request) where T : class{**

**var client = GetHttpClient();**

**var response = await client.SendAsync(request);**

**T result = null;**

**response.EnsureSuccessStatusCode();**

**if (response.IsSuccessStatusCode){**

**result = await response.Content.ReadAsAsync<T>(GetFormatters());}**

**return result;}**

**protected virtual IEnumerable<MediaTypeFormatter> GetFormatters(){**

**// Make default the JSON**

**return new List<MediaTypeFormatter> { new JsonMediaTypeFormatter() };}}**

**B-Interfaces > we create the three interfaces that represent the operation of each Model**

**public interface IBasketApi{**

**Task<BasketModel> GetBasket(string userName);**

**Task<BasketModel> UpdateBasket(BasketModel model);**

**Task CheckoutBasket(BasketCheckoutModel model);}**

**public interface ICatalogApi{**

**Task<IEnumerable<CatalogModel>> GetCatalog();**

**Task<IEnumerable<CatalogModel>> GetCatalogByCategory(string category);**

**Task<CatalogModel> GetCatalog(string id);**

**Task<CatalogModel> CreateCatalog(CatalogModel model);}**

**public interface IOrderApi{**

**Task<IEnumerable<OrderResponseModel>> GetOrdersByUserName(string userName);}**

**\*On the parent folder we will create three classes that inherit from the interfaces**

**public class OrderApi : BaseHttpClientWithFactory, IOrderApi{**

**private readonly IApiSettings \_settings;**

**public OrderApi(IHttpClientFactory factory, IApiSettings settings)**

**: base(factory){\_settings = settings;}**

**public async Task<IEnumerable<OrderResponseModel>> GetOrdersByUserName(string userName){**

**//generate GET Request http://localhost:7000/Order/username={userName} with the body serialize**

**var message = new HttpRequestBuilder(\_settings.BaseAddress)**

**.SetPath(\_settings.OrderPath)**

**.AddQueryString("username", userName)**

**.HttpMethod(HttpMethod.Get)**

**.GetHttpMessage();**

**return await SendRequest<IEnumerable<OrderResponseModel>>(message);}}**

**public class CatalogApi : BaseHttpClientWithFactory, ICatalogApi{**

**private readonly IApiSettings \_settings;**

**public CatalogApi(IHttpClientFactory factory, IApiSettings settings)**

**: base(factory){\_settings = settings;}**

**public async Task<IEnumerable<CatalogModel>> GetCatalog(){**

**//generate Get Request http://localhost:7000/Catalog/**

**var message = new HttpRequestBuilder(\_settings.BaseAddress)**

**.SetPath(\_settings.CatalogPath)**

**.HttpMethod(HttpMethod.Get)**

**.GetHttpMessage();**

**return await SendRequest<IEnumerable<CatalogModel>>(message);}**

**public async Task<CatalogModel> GetCatalog(string id){**

**//generate Get Request http://localhost:7000/Catalog/{id}**

**var message = new HttpRequestBuilder(\_settings.BaseAddress)**

**.SetPath(\_settings.CatalogPath)**

**.AddToPath(id)**

**.HttpMethod(HttpMethod.Get)**

**.GetHttpMessage();**

**return await SendRequest<CatalogModel>(message);}**

**public async Task<IEnumerable<CatalogModel>> GetCatalogByCategory(string category){**

**//generate Get Request http://localhost:7000/Catalog/GetProductByCategory/{category}**

**var message = new HttpRequestBuilder(\_settings.BaseAddress)**

**.SetPath(\_settings.CatalogPath)**

**.AddToPath("GetProductByCategory")**

**.AddToPath(category)**

**.HttpMethod(HttpMethod.Get)**

**.GetHttpMessage();**

**return await SendRequest<IEnumerable<CatalogModel>>(message);}**

**public async Task<CatalogModel> CreateCatalog(CatalogModel catalogModel){**

**//generate POST Request http://localhost:7000/Catalog with the body serialized**

**var message = new HttpRequestBuilder(\_settings.BaseAddress)**

**.SetPath(\_settings.CatalogPath)**

**.HttpMethod(HttpMethod.Post)**

**.GetHttpMessage();**

**var json = JsonConvert.SerializeObject(catalogModel);**

**message.Content = new StringContent(json, Encoding.UTF8, "application/json");**

**return await SendRequest<CatalogModel>(message);}}**

**public class BasketApi : BaseHttpClientWithFactory, IBasketApi{**

**//it will consume the IApiSettings to use it to send the required property to generate Http Client**

**//we register the factory to generate factory instance**

**private readonly IApiSettings \_settings;**

**public BasketApi(IHttpClientFactory factory, IApiSettings settings)**

**: base(factory){\_settings = settings;}**

**public async Task<BasketModel> GetBasket(string userName){**

**//this part it will prepare the http request by using HttpRequestBuilder that using ApiBuilder to prepare**

**//base address and BasketPath that defined in the settings injected and set the username**

**//generate GET Request http://localhost:7000/Basket/username={userName} with the body serialized**

**var message = new HttpRequestBuilder(\_settings.BaseAddress)**

**.SetPath(\_settings.BasketPath)**

**.AddQueryString("username", userName)**

**.HttpMethod(HttpMethod.Get)**

**.GetHttpMessage();**

**return await SendRequest<BasketModel>(message);}**

**public async Task<BasketModel> UpdateBasket(BasketModel model){**

**//generate POST Request http://localhost:7000/Basket with the body serialized**

**var message = new HttpRequestBuilder(\_settings.BaseAddress)**

**.SetPath(\_settings.BasketPath)**

**.HttpMethod(HttpMethod.Post)**

**.GetHttpMessage();**

**var json = JsonConvert.SerializeObject(model);**

**message.Content = new StringContent(json, Encoding.UTF8, "application/json");**

**return await SendRequest<BasketModel>(message);}**

**public async Task CheckoutBasket(BasketCheckoutModel model){**

**//generate POST Request http://localhost:7000/Basket/Checkout with the body serialized**

**var message = new HttpRequestBuilder(\_settings.BaseAddress)**

**.SetPath(\_settings.BasketPath)**

**.AddToPath("Checkout")**

**.HttpMethod(HttpMethod.Post)**

**.GetHttpMessage();**

**var json = JsonConvert.SerializeObject(model);**

**message.Content = new StringContent(json, Encoding.UTF8, "application/json");**

**await SendRequest<BasketCheckoutModel>(message);}}**

**6-on the startup.cs we will configure using the HttpClient and apply DI for the API classes**

**public class Startup{**

**public Startup(IConfiguration configuration){Configuration = configuration;}**

**public IConfiguration Configuration { get; }**

**// This method gets called by the runtime. Use this method to add services to the container.**

**public void ConfigureServices(IServiceCollection services){**

**#region Configuration Dependencies**

**//it will consume the appsettings.json**

**services.Configure<ApiSettings>(Configuration.GetSection(nameof(ApiSettings)));**

**//it will generate one time for first request for any http client request**

**services.AddSingleton<IApiSettings>(sp => sp.GetRequiredService<IOptions<ApiSettings>>().Value);**

**#endregion**

**#region Project Dependencies**

**// add for httpClient factory**

**services.AddHttpClient();**

**// add api dependecy**

**services.AddTransient<ICatalogApi, CatalogApi>();**

**services.AddTransient<IBasketApi, BasketApi>();**

**services.AddTransient<IOrderApi, OrderApi>();**

**#endregion**

**services.AddRazorPages();}**

**// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.**

**public void Configure(IApplicationBuilder app, IWebHostEnvironment env){**

**if (env.IsDevelopment()){app.UseDeveloperExceptionPage();}**

**else{**

**app.UseExceptionHandler("/Error");**

**// The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.**

**app.UseHsts();}**

**app.UseDefaultFiles(new DefaultFilesOptions{**

**DefaultFileNames = new List<string> { "product.html" }});**

**app.UseStaticFiles();**

**app.UseHttpsRedirection();**

**app.UseRouting();**

**app.UseAuthorization();**

**app.UseEndpoints(endpoints =>{endpoints.MapRazorPages();});}}**

**7-on the Docker-compose.yml we configurre**

**//it will define create the image aspnetrunbasics**

**aspnetrunbasics:**

**image: ${DOCKER\_REGISTRY-}aspnetrunbasics**

**build:**

**context: .**

**dockerfile: src/WebApp/AspnetRunBasics/Dockerfile**

**//it will create container aspnetrunbasics that depend on the //catalog.api,basket.api,ordering.api,apigateway**

**aspnetrunbasics:**

**container\_name: aspnetrunbasics**

**environment:**

**- ASPNETCORE\_ENVIRONMENT=Development**

**- "ApiSettings:BaseAddress=apigateway"**

**depends\_on:**

**- catalog.api**

**- basket.api**

**- ordering.api**

**- apigateway**

**volumes:**

**- ${APPDATA}/Microsoft/UserSecrets:/root/.microsoft/usersecrets:ro**

**- ${APPDATA}/ASP.NET/Https:/root/.aspnet/https:ro**

**ports:**

**- "5003:80"**