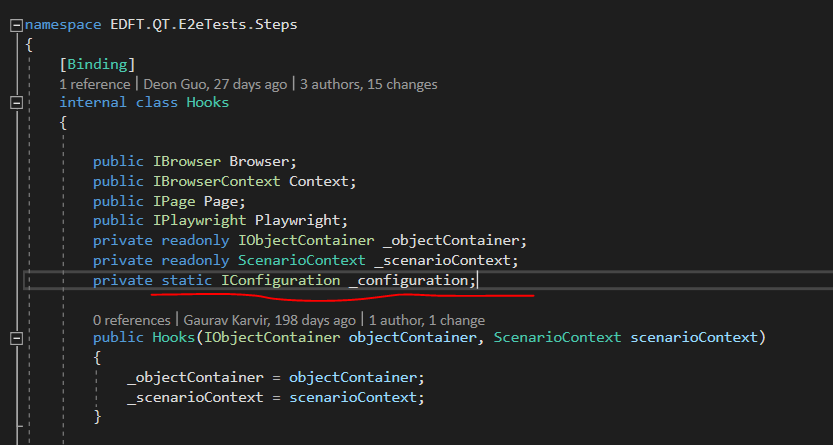
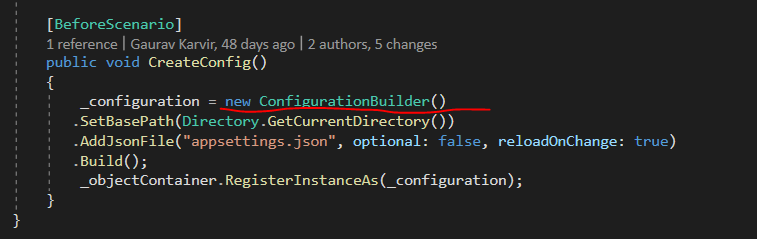
E2E test framework questions:

# 1. Hooks class IConfiguration \_configuration static



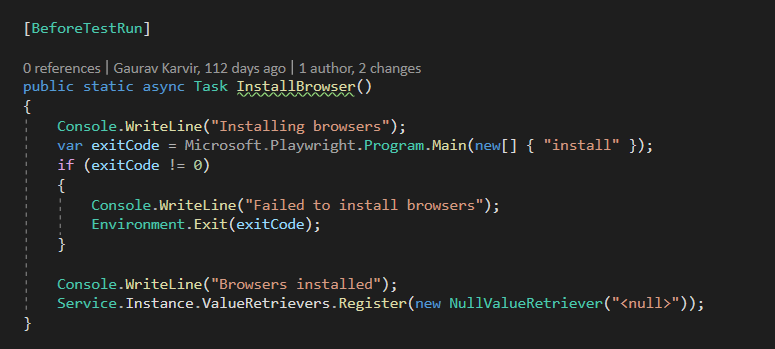
Initiated at BeforeScenario



Does it need to be static? Given that it’s initiated at each scenario. What happens if it’s non-static?

# 2. Hook class static method

This is so that we don’t have to initiate a Hook object to call this method? But as a BeforeTestRun, it runs automatically by specflow – what difference does it make if it’s static or non-static?



Yes it must be static. Specflow requires Before and after testrun to be static – otherwise you get exceptions.

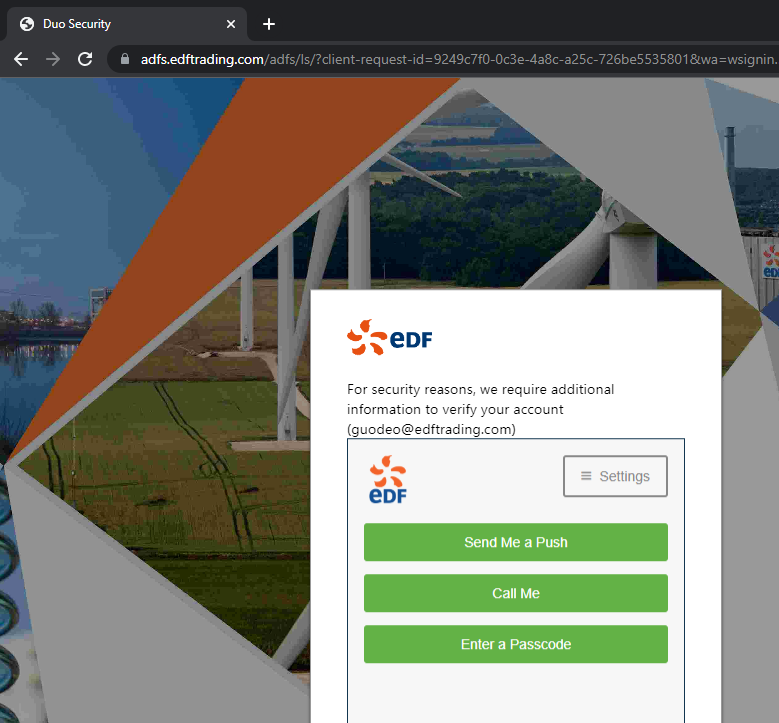
# 3. Should Install browser be in Hooks or BasePage?

We have API tests that don’t require frontend so doesn’t need to install browser. Would BasePage be a better place to perform this?

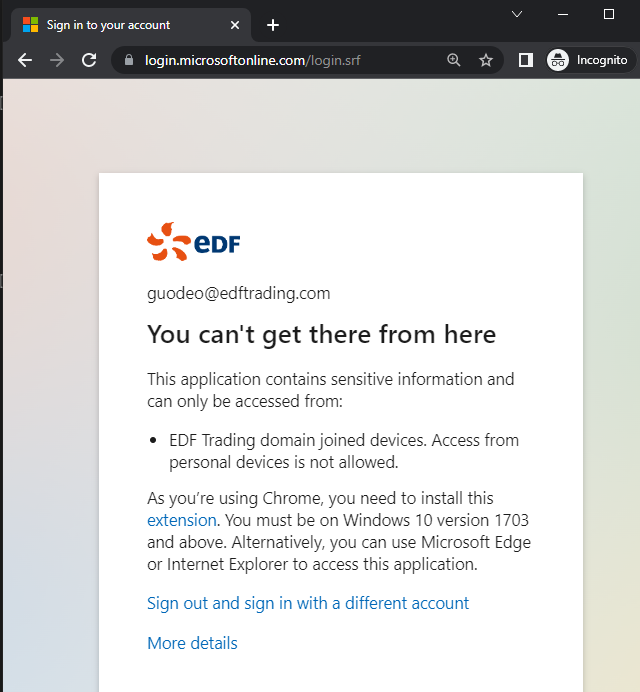
# Authentication with Azure AD

## Problem 1: Launch incognito mode **manually**

Triggers DUO but even after successfully authenticated with DUO, it does not authenticate

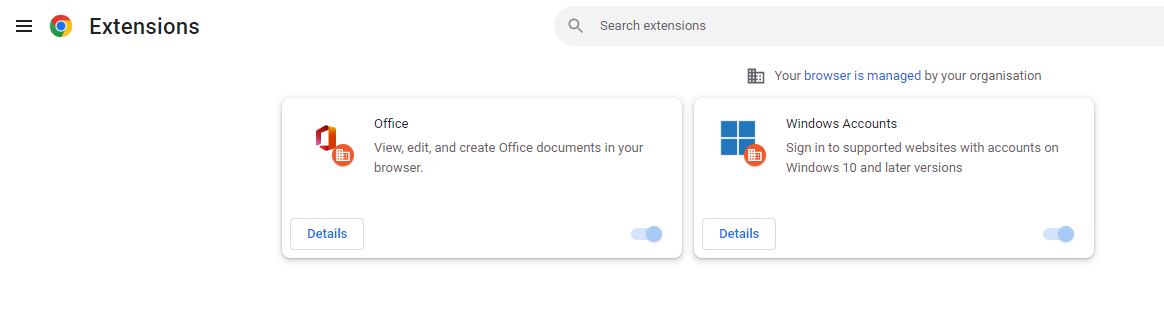


After auth with DUO:

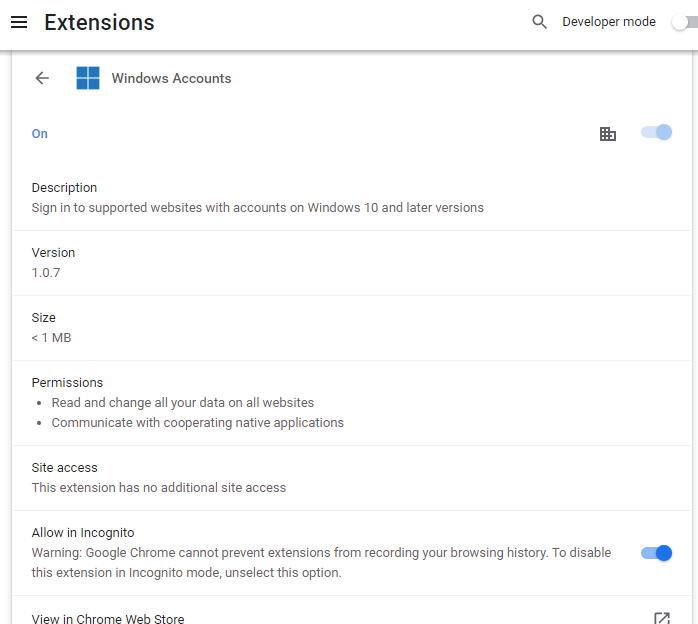


Solution:

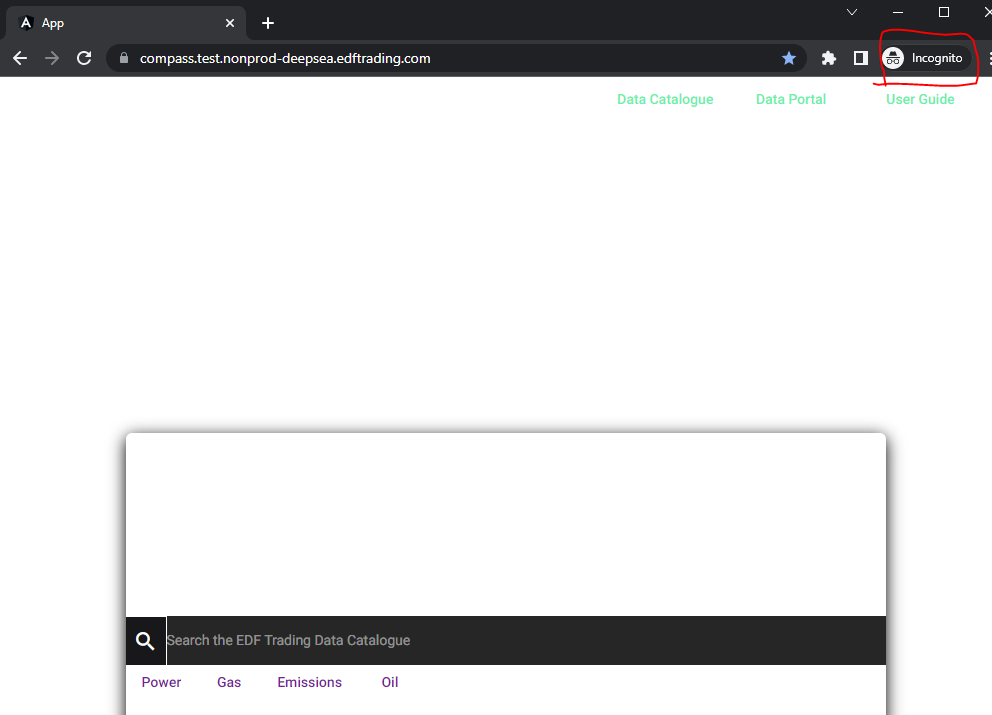
Install Windows Account extension



And allow in incognito mode:

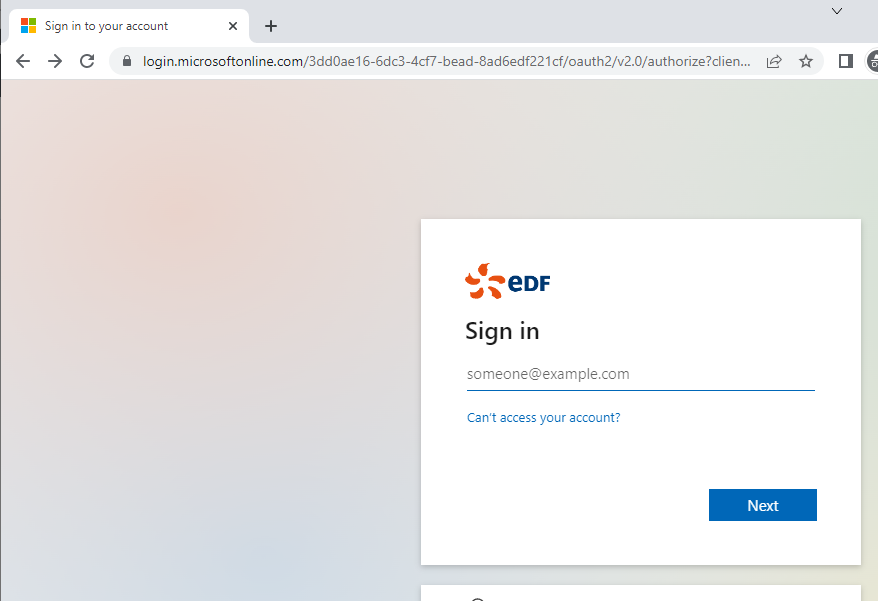


After this, manually launch incognito browser and go to <https://compass.test.nonprod-deepsea.edftrading.com/> 🡪 authenticated automatically



## Problem 2: playwright launched incognito-like browser still can’t SSO like manually launched incognito

Taken to MS Oauth page directly



### Solution:

Chrome always insists on authentication – it does not

Use persistent context

BrowserTypeLaunchPersistentContextOptions typeLaunchOptions = new BrowserTypeLaunchPersistentContextOptions

{

Headless = false,

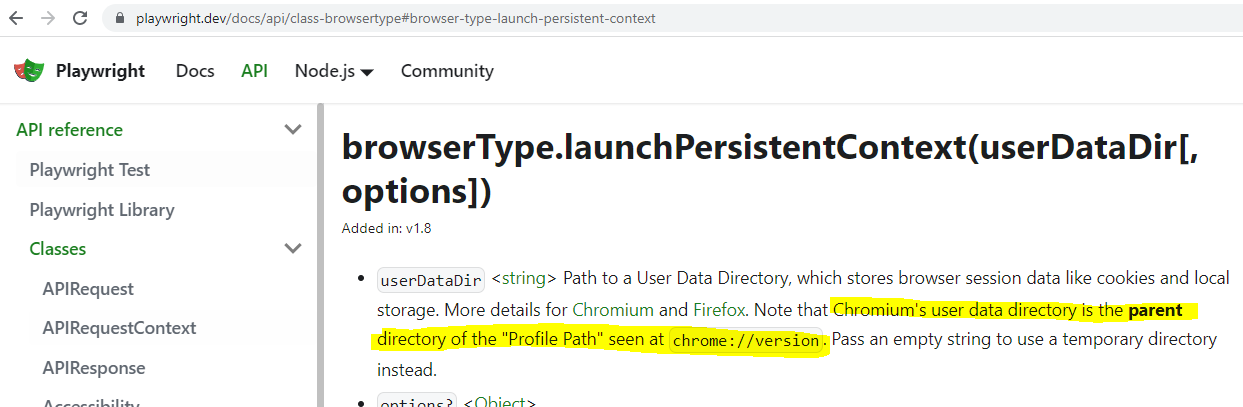
IgnoreHTTPSErrors = true,

Channel = "msedge"

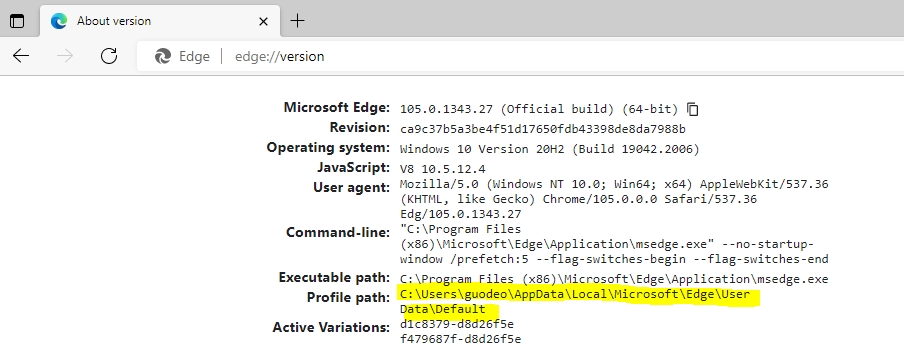
};

Context = await Playwright.Chromium.LaunchPersistentContextAsync(@"C:\Users\guodeo\AppData\Local\Microsoft\Edge\", typeLaunchOptions);

### Note:

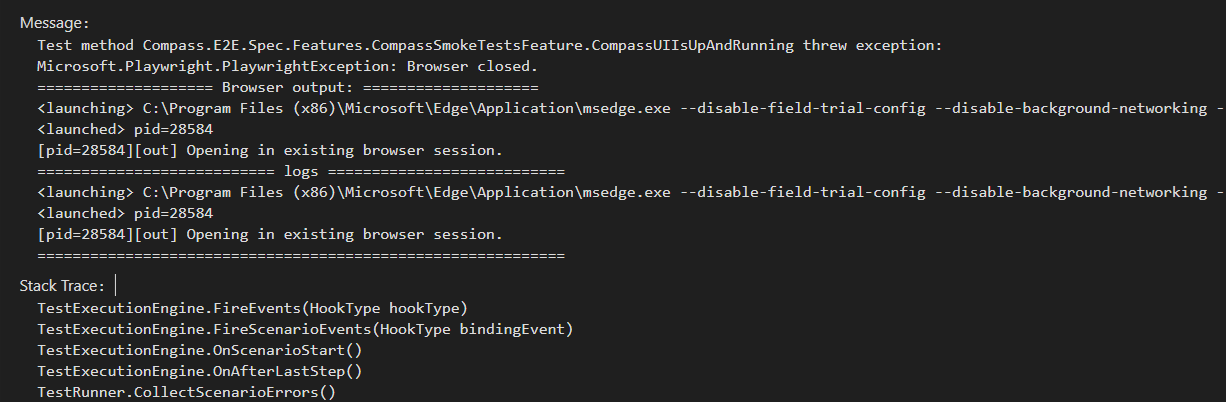
The correct folder path Note that Chromium's user data directory is the **parent (or grandparent)** directory of the "Profile Path" seen at chrome://version

For Edge, type: edge://version to find out the profile path



Based on Playwright’s note, the user data directory should be “C:\Users\guodeo\AppData\Local\Microsoft\Edge\User Data\”

However, with this path I got exception:



In the end, it worked with path: C:\Users\guodeo\AppData\Local\Microsoft\Edge\

## Problem 3: Async mechanism

*[Given(@"the user has obtained a valid access token")]*

*public async* ***Task*** *GivenTheUserHasObtainedAValidAccessToken()*

Specflow generated step definition code uses void instead of Task

*[Given(@"the user has obtained a valid access token")]*

*public async void GivenTheUserHasObtainedAValidAccessToken()*

This breaks async flow – make sure to use Task return type when using async methods!

Alternatively, use this helper to wrap async into sync method: <https://anduin.aiursoft.com/post/2019/12/12/how-to-run-async-method-in-c-synchronous-method>

add the AsyncHelper  class to your project. then use -> e.g. var result = AsyncHelper.RunSync(MyAsyncMethod)

# Json

You don't really need to worry about JContainer in most cases. It is there to help organize and structure [LINQ-to-JSON](https://www.newtonsoft.com/json/help/html/LINQtoJSON.htm) into well-factored code.

The [JToken](https://www.newtonsoft.com/json/help/html/T_Newtonsoft_Json_Linq_JToken.htm) hierarchy looks like this:

JToken - abstract base class

JContainer - abstract base class of JTokens that can contain other JTokens

JArray - represents a JSON array (contains an ordered list of JTokens)

JObject - represents a JSON object (contains a collection of JProperties)

JProperty - represents a JSON property (a name/JToken pair inside a JObject)

JValue - represents a primitive JSON value (string, number, boolean, null)

So you see, a [JObject](https://www.newtonsoft.com/json/help/html/T_Newtonsoft_Json_Linq_JObject.htm) is a [JContainer](https://www.newtonsoft.com/json/help/html/T_Newtonsoft_Json_Linq_JContainer.htm), which is a [JToken](https://www.newtonsoft.com/json/help/html/T_Newtonsoft_Json_Linq_JToken.htm).

Here's the basic rule of thumb:

* If you know you have an object (denoted by curly braces { and } in JSON), use [JObject](https://www.newtonsoft.com/json/help/html/T_Newtonsoft_Json_Linq_JObject.htm)
* If you know you have an array or list (denoted by square brackets [ and ]), use [JArray](https://www.newtonsoft.com/json/help/html/T_Newtonsoft_Json_Linq_JArray.htm)
* If you know you have a primitive value, use [JValue](https://www.newtonsoft.com/json/help/html/T_Newtonsoft_Json_Linq_JValue.htm)
* If you don't know what kind of token you have, or want to be able to handle any of the above in a general way, use [JToken](https://www.newtonsoft.com/json/help/html/T_Newtonsoft_Json_Linq_JToken.htm). You can then check its Type property to determine what kind of token it is and cast it appropriately.

# MongoDB & Json

* Get DB: IMongoDatabase db = MongoClient(connectionString).GetDatabase(dbName)
* Get collection in BsonDocument format so that it can then be converted to Json easily: db.GetCollection<BsonDocument>(collecitonName)
* Use filter to get result in list: collection.Find(filter).ToLIst()
* Results.single() returns one and only object in results – will fail if results have than one
* String jsonString = BsonDocument.toJson()
* JObject j = JObject.Parse(jsonString)

[Given(@"(.\*) exists in the database")]

public void GivenExistsInTheDatabase(string ts)

{

string connectionString = \_configuration["CompassDbConnection"];

string dbName = \_configuration["CompassDbName"];

string collectionName = \_configuration["CompassDbCollection"];

//todo: refactor this to a helper class

IMongoDatabase db = new MongoClient(connectionString).GetDatabase(dbName);

var collection = db.GetCollection<BsonDocument>(collectionName);

var filter = Builders<BsonDocument>.Filter.Eq("catalogueData.id", ts);

var results = collection.Find(filter).ToList();

//todo: insert the ts record into the database

//currently: assume the record is in db (e.g.: seeded) and this if condition should always be false

if(results.Count == 0)

{

Assert.Fail("TS does not exist in the database");

}

//by default, BsonDocument.ToJson() uses Mongo driver's extended Json format which is not standard Json, e.g.: contains line1: "\_id" : ObjectId("6306410b8091c1401d201467"), which is invalid JSON

//as a result, it fails to parse into a JObject. This can be solved by telling MongoDb.Bson to use strict standard Json format with the JsonWriterSettings

var jsonWriterSettings = new JsonWriterSettings { OutputMode = JsonOutputMode.Strict };

string result = results.Single().ToJson(jsonWriterSettings);

JObject resultDb = JObject.Parse(result);

\_scenarioContext["ResultDb"] = resultDb;

//save data in scenarioContext

\_scenarioContext["ts"] = ts;

\_outputHelper.WriteLine(\_scenarioContext["ResultDb"].ToString());

}

# Use ConfidentialClientApplication to get Access Token

[When(@"the user sends a request to the endpoint for the ts")]

public async Task WhenTheUserSendsARequestToTheEndpointForTheTs()

{

//todo: refactor to common helper

//obtain access token

IConfidentialClientApplication app = ConfidentialClientApplicationBuilder

.Create(\_configuration["ClientId"])

.WithClientSecret(\_configuration["ClientSecret"])

.WithAuthority(\_configuration["Authority"])

.Build();

app.AddInMemoryTokenCache();

// With client credentials flows the scopes is ALWAYS of the shape "resource/.default", as the

// application permissions need to be set statically (in the portal or by PowerShell), and then granted by

// a tenant administrator

string[] scopes = new string[] { \_configuration["Scope"] };

AuthenticationResult result = null;

try

{

result = await app.AcquireTokenForClient(scopes).ExecuteAsync();

//add access token to context

\_scenarioContext["AccessToken"] = result.AccessToken;

\_outputHelper.WriteLine("Token acquired: ");

}catch(MsalServiceException e) when (e.Message.Contains("AADSTS70011"))

{

\_outputHelper.WriteLine("invalid scope");

}

//send request with the access token and process the response

if(result != null)

{

var httpClient = new HttpClient();

var apiCaller = new ProtectedApiCallHelper(httpClient);

string url = \_configuration["CompassApiBaseUrl"] + "/details?ts=" + \_scenarioContext["ts"];

await apiCaller.CallWebApiAndProcessResultAsync(url, result.AccessToken, Display, \_scenarioContext);

}

}

# HttpClient and Json

public class ProtectedApiCallHelper

{

protected HttpClient \_httpClient { get; set; }

public ProtectedApiCallHelper(HttpClient httpClient)

{

\_httpClient = httpClient;

}

public async Task CallWebApiAndProcessResultAsync(string webApiUrl, string accessToken, Action<JObject> processResult, ScenarioContext scenarioContext)

{

if (!string.IsNullOrEmpty(accessToken))

{

var defaultRequestHeaders = \_httpClient.DefaultRequestHeaders;

//add media type json if not present

if(defaultRequestHeaders.Accept == null || !defaultRequestHeaders.Accept.Any(m => m.MediaType == "application/json"))

{

\_httpClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

}

//add auth header bearer

defaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", accessToken);

//send request

HttpResponseMessage response = await \_httpClient.GetAsync(webApiUrl);

if (response.IsSuccessStatusCode)

{

//response.Content.ReadAsString() sometimes returns json string with too many escape \ in it, as a result it became invalid JSON

//deserialize is needed in this case

var jsonBytes = await response.Content.ReadAsByteArrayAsync();

var json = Newtonsoft.Json.JsonConvert.DeserializeObject(System.Text.Encoding.UTF8.GetString(jsonBytes));

if (json != null)

{

JObject result = JObject.Parse((string)json);

scenarioContext["ResultApi"] = result;

processResult(result);

}

else

{

Assert.Fail("null JSON object in http response");

}

}

else

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine($"API call failed: {response.StatusCode}");

string content = await response.Content.ReadAsStringAsync();

Console.WriteLine($"Response Content: {content}");

}

Console.ResetColor();

}

}

}

# Query MongoDb and transform to JObject

The helper method that takes in query as function parameter

public List<JObject>? GetQueryData(string connectionString, string dbName, string collectionName, Func<IMongoCollection<BsonDocument>, List<BsonDocument>> runQuery)

{

IMongoDatabase db = new MongoClient(connectionString).GetDatabase(dbName);

var collection = db.GetCollection<BsonDocument>(collectionName);

var results = runQuery(collection);

if (results.Count == 0)

{

return null;

}

else

{

var jsonWriterSettings = new JsonWriterSettings { OutputMode = JsonOutputMode.Strict };

List<JObject> jsonResults = new();

results.ForEach(r => {

string jsonString = r.ToJson(jsonWriterSettings);

jsonResults.Add(JObject.Parse(jsonString));

});

return jsonResults;

}

}

How to use the helper method: define the query func and call the helper method

public List<JObject>? GetLastRunBatches()

{

List<JObject>? lastRunBatches = GetQueryData(\_configuration["CompassDbConnection"], \_configuration["CompassDbName"], "Audit", QueryLastBatches);

if(lastRunBatches == null)

{

Assert.Fail($"Unable to find batches with the run");

}

//\_outputHelper.WriteLine(lastRunBatches[0].ToString());

return lastRunBatches;

}

private List<BsonDocument> QueryLastBatches(IMongoCollection<BsonDocument> collection)

{

string lastRunId = GetLastRunId();

FilterDefinition<BsonDocument> filter = Builders<BsonDocument>.Filter.Eq("auditRunId", lastRunId);

return collection.Find(filter).ToList();

}

## Query MongoDB

private List<BsonDocument> QueryLastBatches(IMongoCollection<BsonDocument> collection)

{

string lastRunId = GetLastRunId();

FilterDefinition<BsonDocument> filter = Builders<BsonDocument>.Filter.Eq("auditRunId", lastRunId);

return collection.Find(filter).ToList();

}

public string GetLastRunId()

{

List<JObject>? lastRunList = GetQueryData(\_configuration["CompassDbConnection"], \_configuration["CompassDbName"], "Audit", QueryAuditLastRun);

JProperty runIdProperty = lastRunList.Single().Property("runId");

if (runIdProperty is null)

{

Assert.Fail("Unable to find last run id");

}

return runIdProperty.Value.ToString();

}

//query documents with runId field, sort by ‘end’ field and get the 1st one only

private List<BsonDocument> QueryAuditLastRun(IMongoCollection<BsonDocument> collection)

{

FilterDefinition<BsonDocument> filter = Builders<BsonDocument>.Filter.Exists("runId");

SortDefinition<BsonDocument> sort = Builders<BsonDocument>.Sort.Descending("end");

return collection.Find(filter).Sort(sort).Limit(1).ToList();

}

# Newtonsoft Json.net

* JToken.DeepEquals(JToken, JToken)
* Enumerate child JToken objects inside of a JToken object: Foreach(JToken in JToken){}
* JObject extends IDictionary<string, JToken>, i.e.: JObject.Key gets the node name and JObject.Value gets the value (type JToken) of the node
* JToken[“key”] retrieves the child JToken with the key: “key”

[Then(@"the response should have the same data as the record in the database")]

public void ThenTheResponseShouldHaveTheSameDataAsTheRecordInTheDatabase()

{

JObject resultDb = (JObject) \_scenarioContext["ResultDb"];

JObject resultApi = (JObject) \_scenarioContext["ResultApi"];

if (resultDb == null || resultApi == null)

{

Assert.Fail("DB query result null or Api response result null");

}

else

{

//todo: JObject from Mongo is not exactly the same as the JObject from API response because:

//id - different - should ignore this

//Assert.IsTrue(JToken.DeepEquals((JToken)\_scenarioContext["ResultDb"], (JToken)\_scenarioContext["ResultApi"]));

foreach(var tokenDb in (JObject)\_scenarioContext["ResultDb"]) {

//\_outputHelper.WriteLine(pair.Key + " \*\*\* " + pair.Value);

if(tokenDb.Key != "\_id") //\_id node is expected to be different - ignore

{

\_outputHelper.WriteLine($"Checking Json token: {tokenDb.Key}");

Assert.IsTrue(JToken.DeepEquals(tokenDb.Value, resultApi[tokenDb.Key]));

}

}

}

}

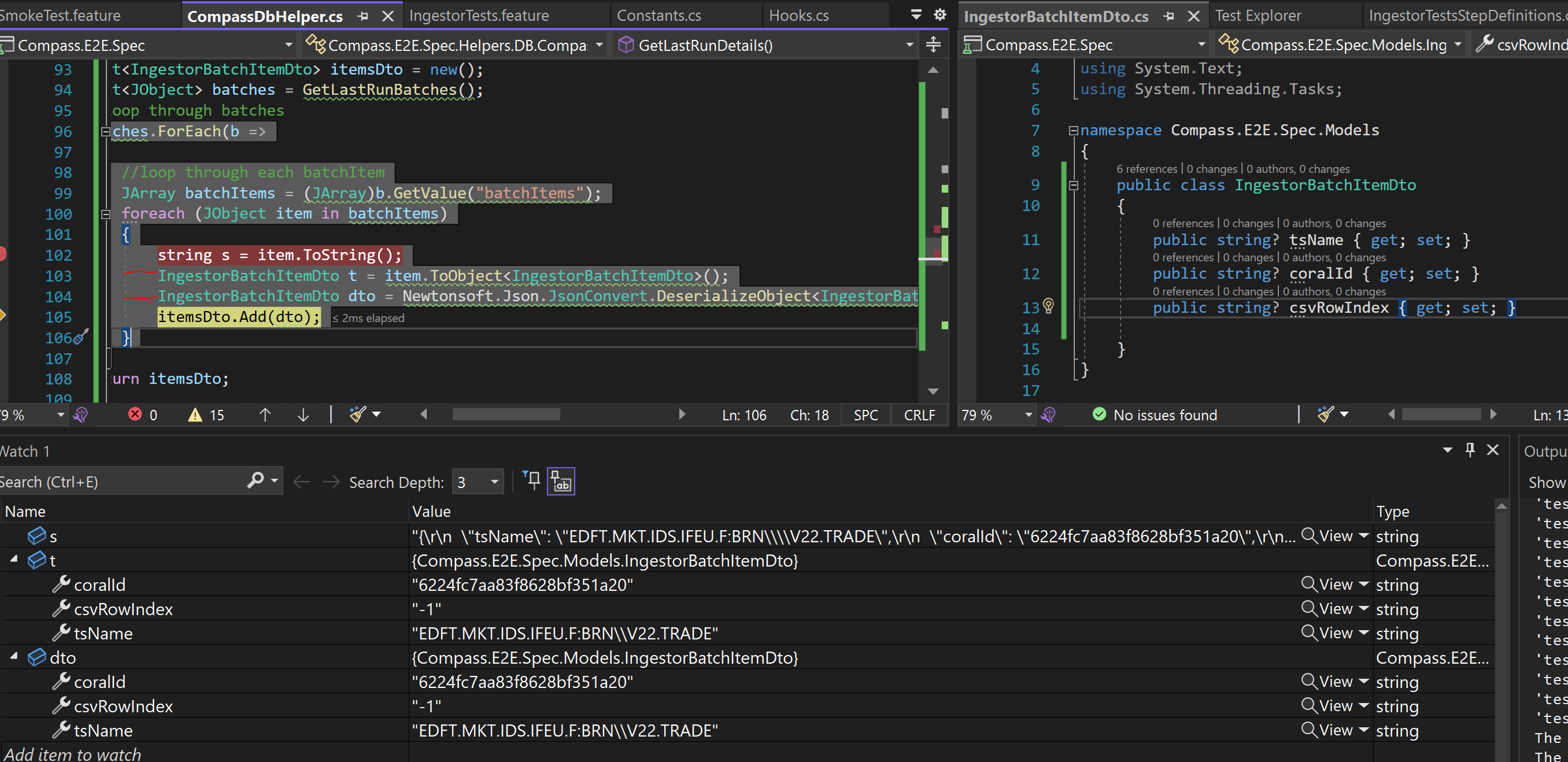
## Converting JObject to object class

2 ways:

1. directly: JObject.ToObject<DtoClass>()

2. Deserialize from string:

JsonConvert.DeserializeObject<DtoClass>(jsonString)



## From Object to JSON

public class Computer

{

public string Cpu { get; set; }

public int Memory { get; set; }

public IList<string> Drives { get; set; }

}

JObject o = (JObject)JToken.FromObject(computer);

Console.WriteLine(o.ToString());

// {

// "Cpu": "Intel",

// "Memory": 32,

// "Drives": [

// "DVD",

// "SSD"

// ]

// }

JArray a = (JArray)JToken.FromObject(computer.Drives);

Console.WriteLine(a.ToString());

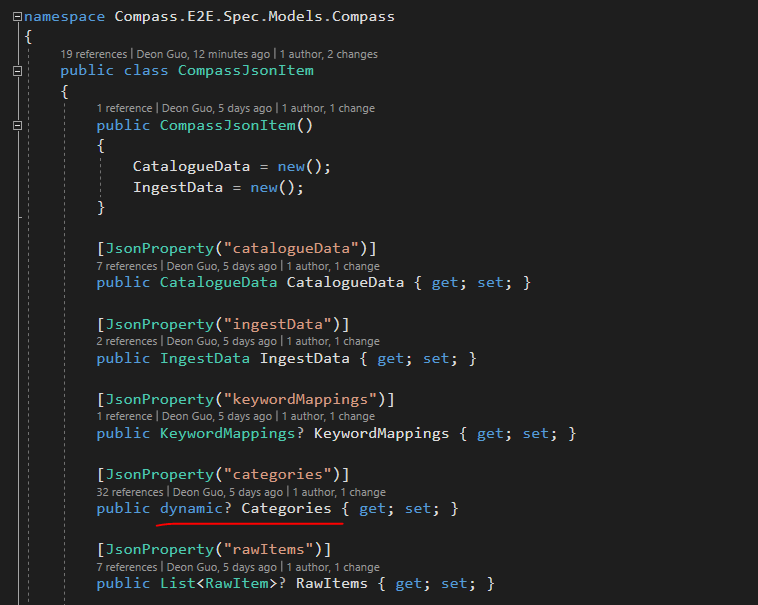
// [

// "DVD",

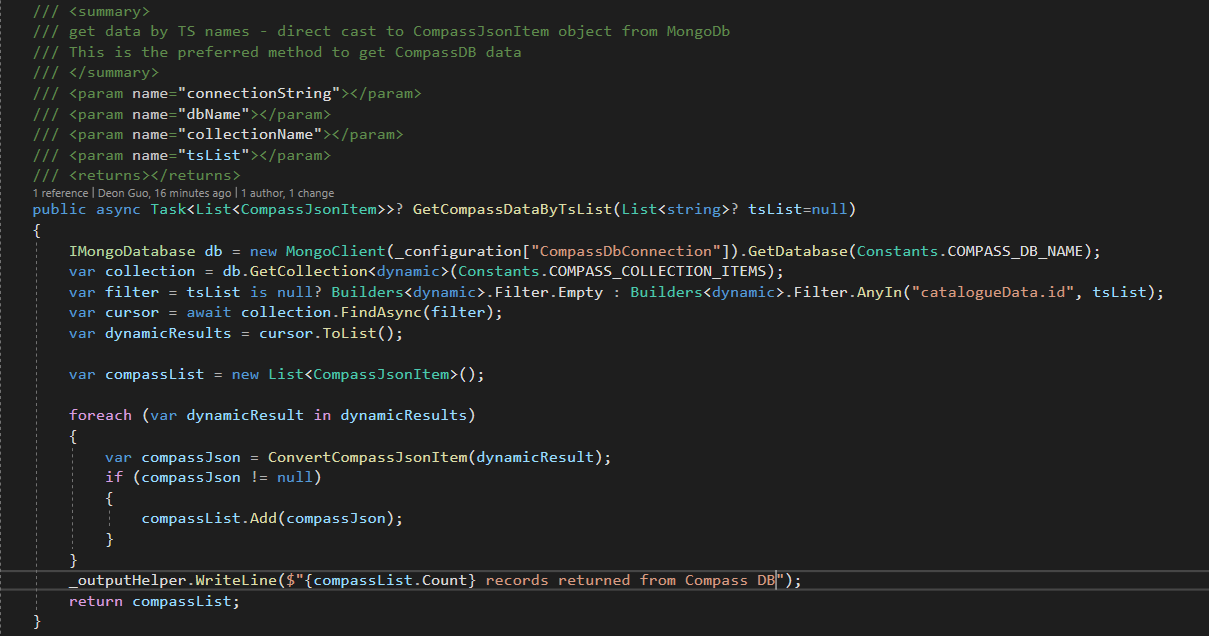
// "SSD"

// ]

## Use Dynamic type between MongoDB, Dto classes and comparison

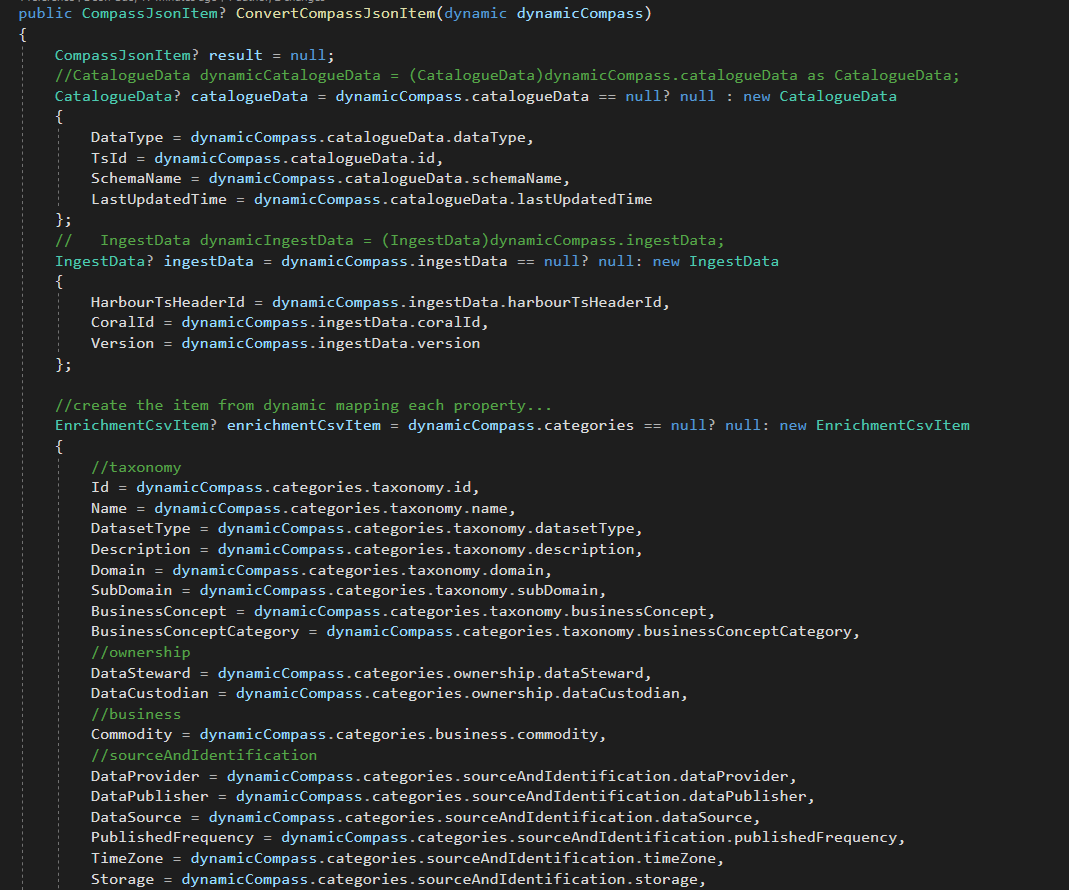
Consider this Dto class: 

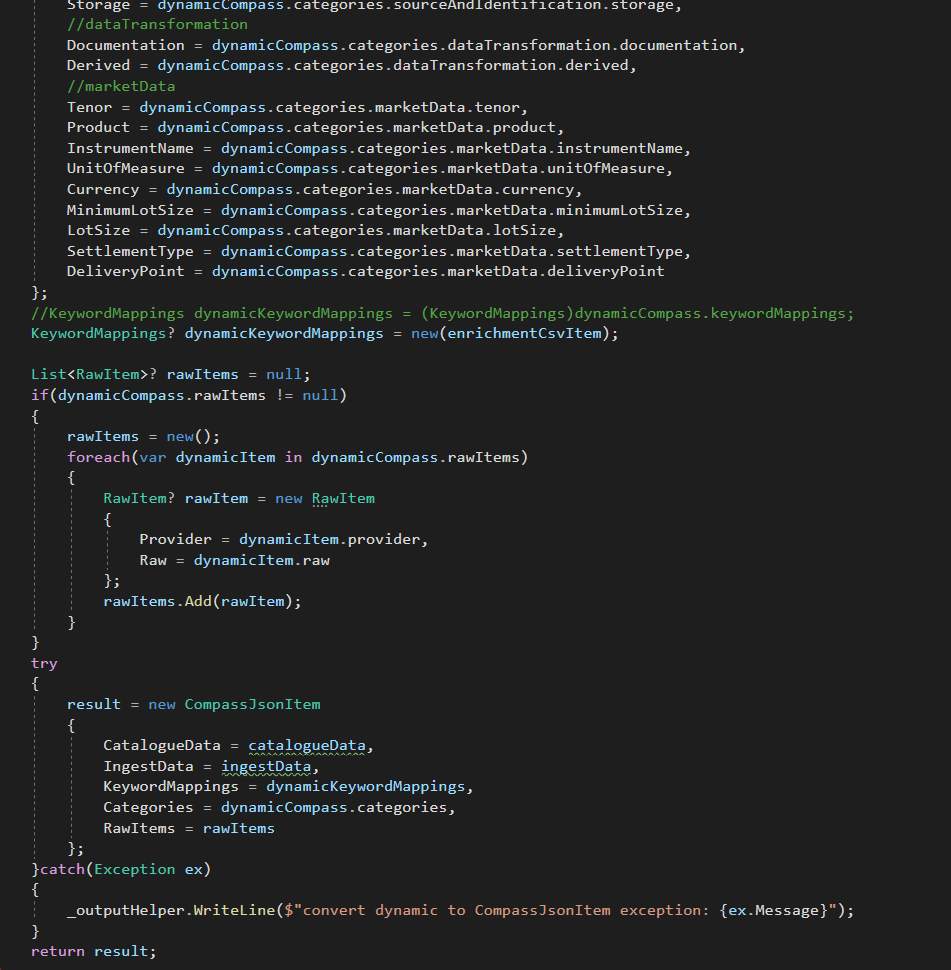
## Get dynamic objects from MongoDB



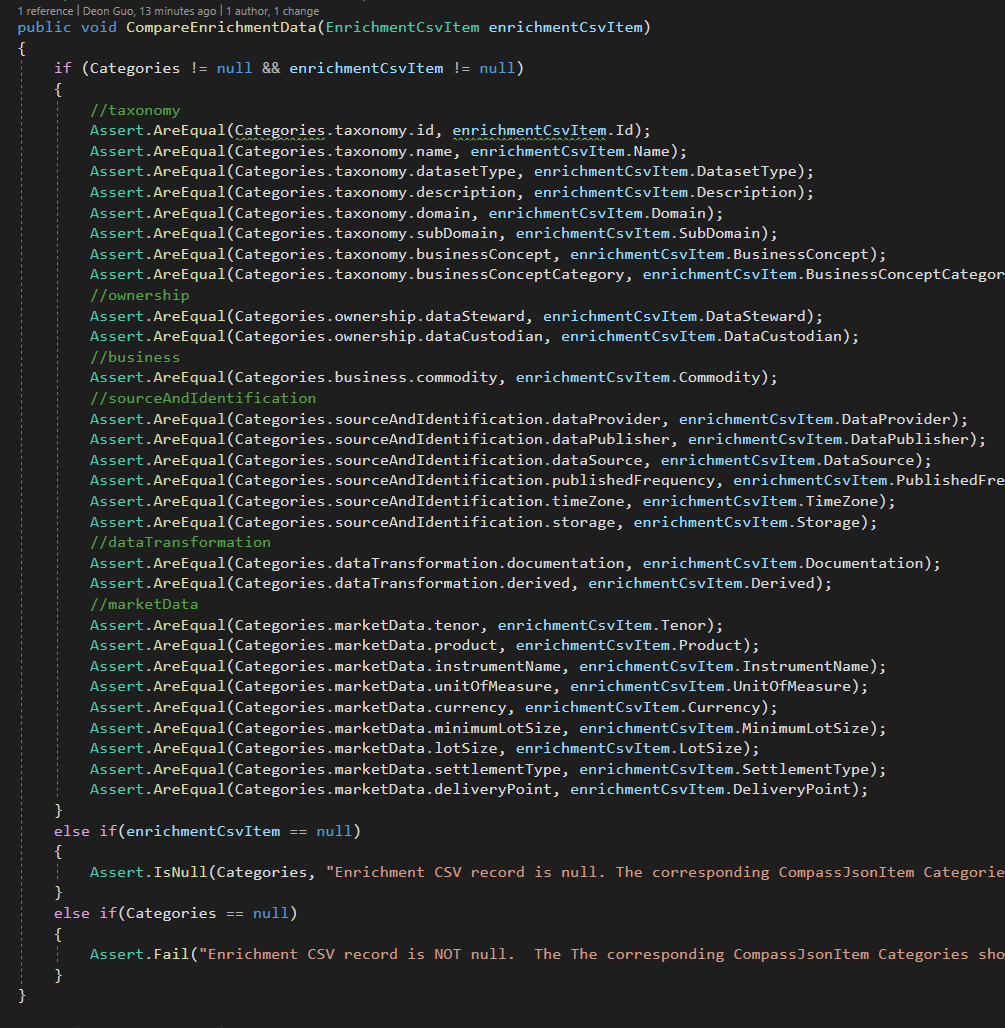
## Convert from dynamic to Dto class

Need to painstakingly specify each property mapping….

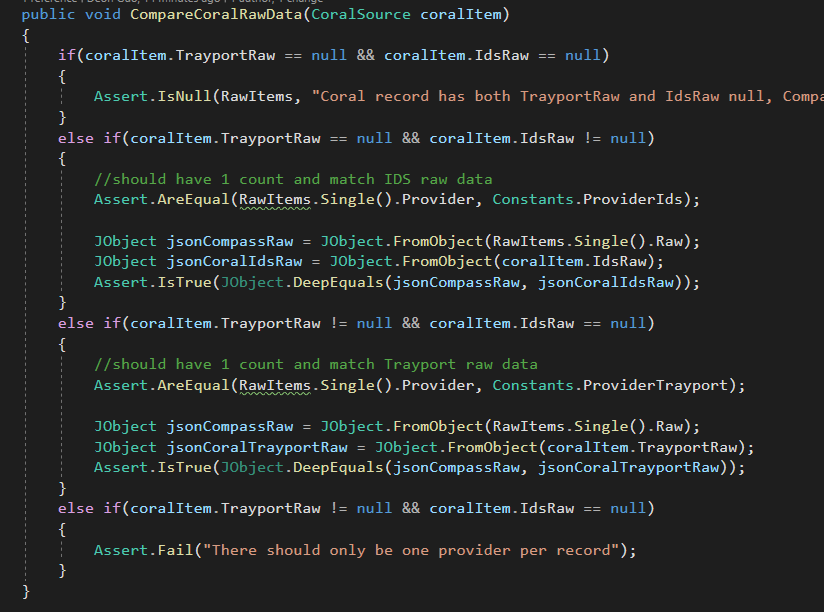




### Compare dynamic object



### Compare dynamic JSONs



# Dependency Injection

1. in Hooks (executes before assembly or scenario)

Declare the IObjectContainer, ScenarioContext and pass them into the constructor as parameters, assign the values from param to the class variables.

The [BeforeScenario] sets the \_configuration and adds it to the \_objectContainer – do we really need this?

private readonly IObjectContainer \_objectContainer;

private readonly ScenarioContext \_scenarioContext;

private static IConfiguration \_configuration;

public Hooks(IObjectContainer objectContainer, ScenarioContext scenarioContext)

{

\_objectContainer = objectContainer;

\_scenarioContext = scenarioContext;

}

[BeforeScenario]

public void CreateConfig()

{

\_configuration = new ConfigurationBuilder()

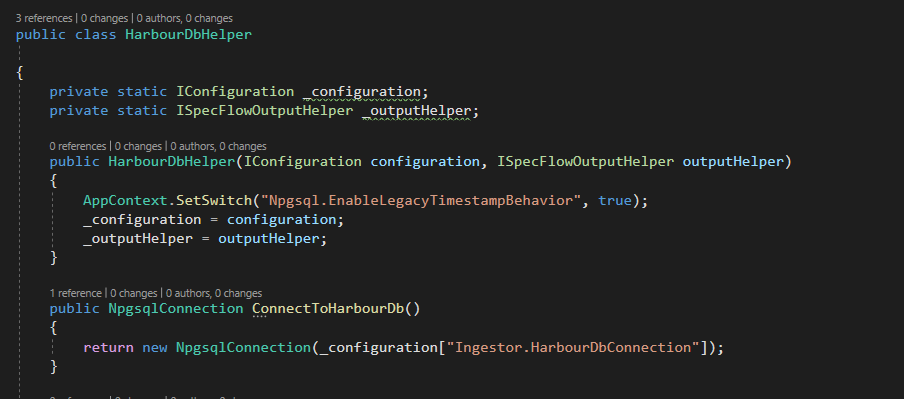
.SetBasePath(Directory.GetCurrentDirectory())

.AddJsonFile("appsettings.json", optional: false, reloadOnChange: true)

.Build();

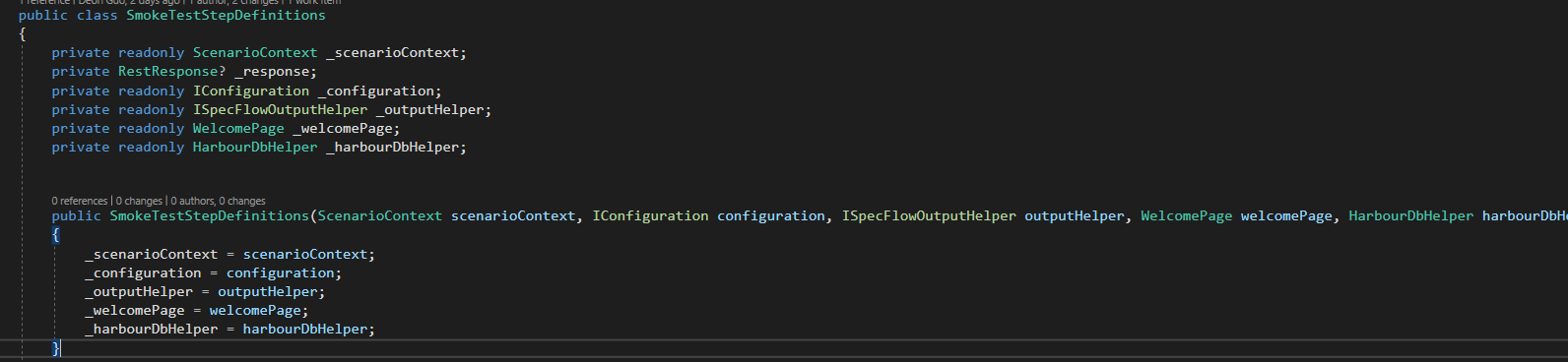
\_objectContainer.RegisterInstanceAs(\_configuration);

}

In the helper class, declare \_config and assign the value from parameter – this should inject \_config into this class variable \_configuration

However, this requires an instance of HarbourDbHelper to be instantiated.

So in the step definition, we inject the harbourDbHelper into the \_harbourDbHelper class variable. The harbourDbHelper parameter is instantiated during the constructor call of SmokeTestStepDefinitions(…)



However, if we made all methods in HarbourDbHelper class static, i.e.: no harbourDbHelper object instantiated in the step definition class, then the \_configuration static class field does not get injected because the constructor was never called.

# CSV

## CSV Gotchas

There are a couple of CSV gotchas that have to be brought up before we dive deeper. Hopefully they should go ahead and explain why rolling your own is sometimes more pain than it’s worth.

* A CSV may or may not have a header row. If there is a header row, then the order of the columns is not important since you can detect what is actually in each column. If there is no header row, then you rely on the order of the columns being the same. Any CSV parser should be able to both read columns based on a “header” value, and by index.
* Any field may be contained in quotes. However fields that contain a line-break, comma, or quotation marks must be contained in quotes.
* To re-emphasize the above, **line breaks within a field are allowed within a CSV as long as they are wrapped in quotation marks**, this is what trips most people up who are simply reading line by line like it’s a regular text file.
* Quote marks within a field are notated by doing double quote marks (As opposed to say an escape character like a back slash).
* Each row should have the same amount of columns, however in the RFC this is labelled as a “should” and not a “must”.
* While yes, the C in CSV stands for comma, ideally a CSV parser can also handle things like TSV (That is using tabs instead of commas).

# Harbour & PostgresHelper

## Todo: use DateTime for the CompletedTimestamp field as follow:

