LinksPlatform's Platform.Data.Doublets.Gql Class Library

1.1 ./csharp/Platform.Data.Doublets.Gql.Tests/ClientTests.cs

```
using GraphQL.Client.Http;
using GraphQL.Client.Serializer.Newtonsoft;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Hosting;
using Platform.Data.Doublets.Gql.Client;
using Platform.Data.Doublets.Gql.Server;
using Platform.Data.Doublets.Memory;
using Platform.Data.Doublets.Memory.United.Generic;
using Platform.Memory;
using Serilog;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Reflection;
using System.Threading;
using Xunit;
using TLinkAddress = System.UInt64;

namespace Platform.Data.Doublets.Gql.Tests;

public class ClientTests : IDisposable
{
    private readonly LinksConstants<TLinkAddress> _constants;
    private readonly LinksGqlAdapter _linksGqlAdapter;
    private readonly Process _serverProcess;
    private readonly Uri _endPoint;
    public string TempFilePath = new IO.TemporaryFile();

    public ClientTests()
    {
        _constants = new LinksConstants<TLinkAddress>(true);
        _serverProcess = TestExtensions.RunServer(TempFilePath);
        _endPoint = TestExtensions.GetEndPointFromServerProcess(_serverProcess);
        var graphQlClient = new GraphQLHttpClient(_endPoint, new NewtonsoftJsonSerializer());
        _linksGqlAdapter = new LinksGqlAdapter(graphQlClient, _constants);
    }

    public void Dispose()
    {
        _serverProcess.Kill(true);
    }



    private void TestCud()
    {
        TLinkAddress linksAmount = 5;
        // Create
        for (TLinkAddress i = 1; i <= linksAmount; i++)
        {
            TLinkAddress one = 1;
            // Create
            var createdLink = _linksGqlAdapter.CreateAndUpdate(one, i);
            // Count
            Assert.Equal(i, createdLink);
            Assert.Equal(i, _linksGqlAdapter.Count());
            var allLinks = new List<Link<TLinkAddress>>();
            _linksGqlAdapter.Each(link =>
            {
                allLinks.Add(new Link<TLinkAddress>(link));
                return _constants.Continue;
            });
            Assert.Equal(i, _linksGqlAdapter.Count());
        }
    }

    [Fact]
    public void CudTest()
    {
        TestCud();
    }

    [Fact]
    public void EachTest()
    {
        TestCud();
        var count = _linksGqlAdapter.Count();
        TLinkAddress eachIterations = 0;
        _linksGqlAdapter.Each(link =>
        {
            Assert.Equal(++eachIterations, _linksGqlAdapter.GetTarget(link));
            return _linksGqlAdapter.Constants.Continue;
        }, new Link<TLinkAddress>(_constants.Any, _constants.Any, _constants.Any));
        Assert.Equal(count, eachIterations);
    }
}
```

1.2 ./csharp/Platform.Data.Doublets.Gql.Tests/DeepGenericLinksTests.cs

```
using GraphQL.Client.Http;
using GraphQL.Client.Serializer.Newtonsoft;
using Platform.Data.Doublets.Decorators;
using Platform.Data.Doublets.Gql.Client;
using Platform.Data.Doublets.Tests;
using Platform.IO;
using Platform.Memory;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Net.Http;
using Xunit;
using TLinkAddress = System.UInt64;

namespace Platform.Data.Doublets.Gql.Tests;

public class DeepGenericLinksTests
{
    public readonly string TempFilePath;
    public readonly Uri EndPoint = new Uri("");

    public DeepGenericLinksTests()
    {
        TempFilePath = new TemporaryFile();
    }

    [Fact (Skip = "Temp skip", Timeout = 60000)]
    public void CRUDTest()
    {
        Using(links =>
        {
            var allLinks = links.All();
            foreach (var linkToDelete in allLinks)
            {
                var id = linkToDelete![0];
                if (links.Exists(id))
                {
                    links.Delete(id);
                }
            }
            links.TestCRUDOperations();
        });
    }

    [Fact (Skip = "Temp skip", Timeout = 60000)]
    public void RawNumbersCRUDTest()
    {
        Using(links =>
        {
            var allLinks = links.All();
            foreach (var linkToDelete in allLinks)
            {
                var id = linkToDelete![0];
                if (links.Exists(id))
                {
                    links.Delete(id);
                }
            }
            links.TestRawNumbersCRUDOperations();
        });
    }

    [Fact (Skip = "Temp skip", Timeout = 60000)]
    public void MultipleRandomCreationsAndDeletionsTest()
    {
        Using(links =>
        {
            var allLinks = links.All();
            foreach (var linkToDelete in allLinks)
            {
                var id = linkToDelete![0];
                if (links.Exists(id))
                {
                    links.Delete(id);
                }
            }
            links.TestMultipleRandomCreationsAndDeletions(10);
        });
    }

    [Fact (Skip = "Temp skip", Timeout = 60000)]
    public void MultipleRandomCreationsAndDeletionsWithDecoratorsTest()
    {
        Using(links =>
        {
            var allLinks = links.All();
            foreach (var linkToDelete in allLinks)
            {
                var id = linkToDelete![0];
                if (links.Exists(id))
                {
                    links.Delete(id);
                }
            }
            links.DecorateWithAutomaticUniquenessAndUsagesResolution().TestMultipleRandomCreatio
            ↪  nsAndDeletions(10);
        });
    }

    private void Using(Action<ILinks<TLinkAddress>> action)
    {
        var graphqlClient = new GraphQLHttpClient(EndPoint, new NewtonsoftJsonSerializer());
        var linksConstants = new LinksConstants<TLinkAddress>(true);
        var token = "";
        var linksGqlStorage = new DeepGqlAdapter(graphqlClient, linksConstants, token);
        using var logFile = File.Open("linksLogger.txt", FileMode.Create, FileAccess.Write);
        LoggingDecorator<TLinkAddress> decoratedLinksStorage = new(linksGqlStorage, logFile);
        action(decoratedLinksStorage);
    }
}
```

1.3 ./csharp/Platform.Data.Doublets.Gql.Tests/GenericLinksTests.cs

```
using GraphQL.Client.Http;
using GraphQL.Client.Serializer.Newtonsoft;
using Platform.Data.Doublets.Decorators;
using Platform.Data.Doublets.Gql.Client;
using Platform.Data.Doublets.Tests;
using Platform.IO;
using Platform.Memory;
using System;
using System.Diagnostics;
using System.IO;
using Xunit;

namespace Platform.Data.Doublets.Gql.Tests;

public class GenericLinksTests : IDisposable
{
    public readonly string TempFilePath;
    public readonly Uri EndPoint;
    public readonly Process ServerProcess;

    public GenericLinksTests()
    {
        TempFilePath = new TemporaryFile();
        ServerProcess = TestExtensions.RunServer(TempFilePath);
        EndPoint = TestExtensions.GetEndPointFromServerProcess(ServerProcess);
    }

    public void Dispose()
    {
        ServerProcess.Kill(true);
    }

    [Fact (Skip = "Temp skip", Timeout = 60000)]
    public void CRUDTest()
    {
        Using(links => links.TestCRUDOperations());
    }

    [Fact (Skip = "Temp skip", Timeout = 60000)]
    public void RawNumbersCRUDTest()
    {
        Using(links => links.TestRawNumbersCRUDOperations());
    }

    [Fact (Skip = "Temp skip", Timeout = 60000)]
    public void MultipleRandomCreationsAndDeletionsTest()
    {
        Using(links => links.TestMultipleRandomCreationsAndDeletions(7));
    }

    [Fact (Skip = "Temp skip", Timeout = 60000)]
    public void MultipleRandomCreationsAndDeletionsWithDecoratorsTest()
    {
        Using(links =>
        {
            var allLinks = links.All();
            foreach (var linkToDelete in allLinks)
            {
                var id = linkToDelete![0];
                if (links.Exists(id))
                {
                    links.Delete(id);
                }
            }
            links.DecorateWithAutomaticUniquenessAndUsagesResolution().TestMultipleRandomCreatio⌟
             ↪  nsAndDeletions(10);
        });
    }
    private void Using(Action<ILinks<ulong>> action)
    {
        var graphqlClient = new GraphQLHttpClient(EndPoint, new NewtonsoftJsonSerializer());
        var linksConstants = new LinksConstants<ulong>(true);
        var linksGqlStorage = new LinksGqlAdapter(graphqlClient, linksConstants);
        using var logFile = File.Open("linksLogger.txt", FileMode.Create, FileAccess.Write);
        LoggingDecorator<ulong> decoratedLinksStorage = new(linksGqlStorage, logFile);
        action(decoratedLinksStorage);
    }
}
```

1.4 ./csharp/Platform.Data.Doublets.Gql.Tests/MutationTests.cs

```
using GraphQL;
using GraphQL.SystemTextJson;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using Platform.Data.Doublets.Gql.Client;
using Platform.Data.Doublets.Gql.Schema;
using Platform.Data.Doublets.Memory;
using Platform.Data.Doublets.Memory.United.Generic;
using Platform.IO;
using Platform.Memory;
using System;
using System.Collections.Generic;
using System.Linq;
using Xunit;
using TLinkAddress = System.UInt64;

namespace Platform.Data.Doublets.Gql.Tests
{
    public class MutationTests
    {
        public static EqualityComparer<TLinkAddress> EqualityComparer =
         ↪  EqualityComparer<TLinkAddress>.Default;
        public static ILinks<ulong> CreateLinks() => CreateLinks<ulong>(new TemporaryFile());

        public static ILinks<TLinkAddress> CreateLinks<TLinkAddress>(string dataDBFilename)
        {
            var linksConstants = new LinksConstants<TLinkAddress>(true);
            return new UnitedMemoryLinks<TLinkAddress>(new
             ↪  FileMappedResizableDirectMemory(dataDBFilename),
             ↪  UnitedMemoryLinks<TLinkAddress>.DefaultLinksSizeStep, linksConstants,
             ↪  IndexTreeType.Default);
        }

        [Fact]
        public void InsertLinksOne()
        {
            var links = CreateLinks();
            LinksSchema linksSchema = new(links, new DefaultServiceProvider());
            var jsonTask = linksSchema.ExecuteAsync(_ => { _.Query = @"
        mutation {
          insert_links_one(object: {from_id: 1, to_id: 1}) {
            id
          from_id
          to_id
          }
        }
        "; });
            dynamic result = Newtonsoft.Json.JsonConvert.DeserializeObject<dynamic>(jsonTask.Result);
            if (result.ContainsKey("errors"))
            {
                throw new Exception(result.errors.ToString());
            }
        }

        [Fact]
        public void InsertLinks()
        {
            var links = CreateLinks();
            LinksSchema linksSchema = new(links, new DefaultServiceProvider());
            var jsonTask = linksSchema.ExecuteAsync(_ => { _.Query = @"
        mutation {
          insert_links(objects: [{ from_id: 1, to_id: 1 }, { from_id: 2, to_id: 2 }]) {
            returning {
              id
              from_id
              to_id
            }
          }
        }
        "; });
            dynamic result = Newtonsoft.Json.JsonConvert.DeserializeObject<dynamic>(jsonTask.Result);
            if (result.ContainsKey("errors"))
            {
                throw new Exception(result.errors.ToString());
            }
        }

        [Fact]
        public void UpdateLinks()
        {
            var links = CreateLinks();
            LinksSchema linksSchema = new(links, new DefaultServiceProvider());
            var jsonTask = linksSchema.ExecuteAsync(_ => { _.Query = @"
        mutation {
          update_links(_set: { from_id: 1, to_id: 2 }, where: { from_id: { _eq: 2 }, to_id: { _eq: 2 } }) {
            returning {
              id
              from_id
              to_id
            }
          }
        }
        "; });
            dynamic result = Newtonsoft.Json.JsonConvert.DeserializeObject<dynamic>(jsonTask.Result);
            if (result.ContainsKey("errors"))
            {
                throw new Exception(result.errors.ToString());
            }
        }

        [Fact]
        public void DeleteLinks()
        {

            var links = CreateLinks();
            LinksSchema linksSchema = new(links, new DefaultServiceProvider());
            var jsonTask = linksSchema.ExecuteAsync(_ => { _.Query = @"
        mutation {
          delete_links(where: { from_id: { _eq: 1 }, to_id: { _eq: 1 } }) {
            returning {
              id
              from_id
              to_id
            }
          }
        }
        "; });
            dynamic result =
            ↪  Newtonsoft.Json.JsonConvert.DeserializeObject<dynamic>(jsonTask.Result);
            if (result.ContainsKey("errors"))
            {
                throw new Exception(result.errors.ToString());
            }
        }

        [Fact]
        public void CreateZeroZeroAndUpdateToOneOneById()
        {
            var links = CreateLinks();
            LinksSchema linksSchema = new(links, new DefaultServiceProvider());
            var jsonTask = linksSchema.ExecuteAsync(_ => { _.Query = @"
            mutation {
              insert_links_one(object: {from_id: 0, to_id: 0}) {
                id
                from_id
                to_id
              }
            }
            "; });
            var jsonSerializer = new JsonSerializer();
            var jsonResponse = jsonTask.Result;
            Assert.False(JObject.Parse(jsonResponse).ContainsKey("errors"));
            jsonTask = linksSchema.ExecuteAsync(_ => { _.Query = @"
            mutation {
              update_links(_set: { from_id: 1, to_id: 1 }, where: { id: {_eq: 1} }) {
                returning {
                  id
                  from_id
                  to_id
                }
              }
            }
            "; });
            dynamic result =
            ↪  Newtonsoft.Json.JsonConvert.DeserializeObject<dynamic>(jsonTask.Result);
            if (result.ContainsKey("errors"))
            {
                throw new Exception(result.errors.ToString());
            }
            Assert.True(1 == Convert.ToInt32(result.data.update_links.returning[0].id));
        }
    }
}
```

1.5 ./csharp/Platform.Data.Doublets.Gql.Tests/QueryTest.cs

```
using GraphQL;
using GraphQL.SystemTextJson;
using Newtonsoft.Json.Linq;
using Platform.Data.Doublets.Gql.Schema;
using Platform.Data.Doublets.Memory;
using Platform.Data.Doublets.Memory.United.Generic;
using Platform.IO;
using Platform.Memory;
using Xunit;
using TLinkAddress = System.UInt64;

namespace Platform.Data.Doublets.Gql.Tests
{
    public class QueryTests
    {
        public static ILinks<ulong> CreateLinks() => CreateLinks<ulong>(new TemporaryFile());

        public static ILinks<TLinkAddress> CreateLinks<TLinkAddress>(string dataDbFilename)
        {
            var linksConstants = new LinksConstants<TLinkAddress>(true);
            return new UnitedMemoryLinks<TLinkAddress>(new
            ↪  FileMappedResizableDirectMemory(dataDbFilename),
            ↪  UnitedMemoryLinks<TLinkAddress>.DefaultLinksSizeStep, linksConstants,
            ↪  IndexTreeType.Default);
        }

        [InlineData(@"
        {
          links {
            id
          }
        }
        ")]
        [InlineData(@"
        {
          links(
            where: { id: { _eq: 1 }, from_id: { _eq: 1 }, to_id: { _eq: 1 } }
            distinct_on: [from_id]
            order_by: { id: asc }
            offset: 0
            limit: 1
          ) {
            id
            from_id
            from {
              id
              from_id
              to_id
            }
            out {
              id
              from_id
              to_id
            }
            to_id
            to {
              id
              from_id
              to_id
            }
            in {
              id
              from_id
              to_id
            }
          }
        }
        ")]
        [InlineData(@"
        {
          links(
            where: { id: { _eq: 1 }, from_id: { _eq: 1 }, to_id: { _eq: 1 } }
            distinct_on: [from_id]
            order_by: { id: asc }
            offset: 0
            limit: 1
          ) {
            id
            from_id
            from {
              id
              from_id
              to_id
            }
            out(
              where: { from_id: { _eq: 1 }, to_id: { _eq: 1 } }
              distinct_on: [from_id]
              order_by: { id: asc }
              offset: 0
              limit: 1
            ) {
              id
              from_id
              to_id
            }
            to_id
            to {
              id
              from_id
              to_id
            }
            in(
              where: { from_id: { _eq: 1 }, to_id: { _eq: 1 } }
              distinct_on: [from_id]
              order_by: { id: asc }
              offset: 0
              limit: 1
            ) {
              id
              from_id
              to_id
            }
          }
        }
        ")]
        [Theory]
        public void QueryData(string query)
        {
            var links = CreateLinks();
            LinksSchema linksSchema = new(links, new DefaultServiceProvider());
            var jsonTask = linksSchema.ExecuteAsync(_ => { _.Query = query; });
            var response = JObject.Parse(jsonTask.Result);
            var error = response.ContainsKey("errors");
            Assert.False(error);
        }
    }
}
```

1.6 ./csharp/Platform.Data.Doublets.Gql.Tests/TestExtensions.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Threading;

namespace Platform.Data.Doublets.Gql.Tests;

public static class TestExtensions
{
    public static Process RunServer(string tempFilePath)
    {
        var currentAssemblyDirectory = Directory.GetCurrentDirectory();
        var currentProjectDirectory = Path.GetFullPath(Path.Combine(currentAssemblyDirectory,
         ↪  "..", "..", ".."));
        var serverProjectDirectory = Path.GetFullPath(Path.Combine(currentProjectDirectory,
         ↪  "..", "Platform.Data.Doublets.Gql.Server"));
        var processStartInfo = new ProcessStartInfo { WorkingDirectory = serverProjectDirectory,
         ↪  FileName = "dotnet", Arguments = $"run -f net5 {tempFilePath}",
         ↪  RedirectStandardOutput = true, RedirectStandardInput = true};
        var process = Process.Start(processStartInfo);
        if (null == process || process.HasExited)
        {
            throw new Exception("Failed to start server process");
        }
        return process;
    }

    public static Uri GetEndPointFromServerProcess(Process process)
    {
        while (true)
        {
            var standartOutput = process?.StandardOutput;
            if(standartOutput == null)
            {
                Thread.Sleep(TimeSpan.FromSeconds(1));
                continue;
            }
            var processOutputLine = standartOutput.ReadLine();
            if (string.IsNullOrEmpty(processOutputLine))
            {
                Thread.Sleep(TimeSpan.FromSeconds(1));
                continue;
            }
            if (processOutputLine.Contains("Unable to start"))
            {
                throw new Exception("Unable to start.");
            }
            if (processOutputLine.Contains("Now listening on: "))
            {
                var index = processOutputLine.IndexOf("Now listening on: ",
                 ↪  StringComparison.Ordinal) + "Now listening on: ".Length;
                var uriString = processOutputLine.Substring(index);
                return new Uri($"{uriString}/v1/graphql");
            }
        }
    }
}
```

Index

./csharp/Platform.Data.Doublets.Gql.Tests/ClientTests.cs, 1
./csharp/Platform.Data.Doublets.Gql.Tests/DeepGenericLinksTests.cs, 2
./csharp/Platform.Data.Doublets.Gql.Tests/GenericLinksTests.cs, 3
./csharp/Platform.Data.Doublets.Gql.Tests/MutationTests.cs, 4
./csharp/Platform.Data.Doublets.Gql.Tests/QueryTest.cs, 6
./csharp/Platform.Data.Doublets.Gql.Tests/TestExtensions.cs, 8