

xDataService

it is a Part of xDashboard on SaherElm IT Center which provides:

- all requirements to Implement Data Access Layer in a **xDashboard** based Project.

this module has following dependencies :

- xModels
- xCommons

for configure and use this Module refer to DI.XDIHelperExtension.cs file.

Implementation

Supported Providers

- MySQL
- SQLite
- Oracle
- SQLServer
- MongoDB

Extends Data Access

Entiy

an Entity usually maps a Table Row to Specified Class and Describe it's Columns as Class Properties and Specified Data types.

Design Entities for a Business is Very Important Step.

for using xDataService Features, you Hvae to Extends Entities from XBaseEntity.

XBaseEntity

all xDataService based Entities must inherit from this base Class.

Id and Deleted Properties set to all TEntities.

- Id: is Entity Primary Key;
- Deleted: used to Filter Soft Deleted Entities;

```
public abstract XBaseEntity<TKey> {}
```

using Key Type for Entity specified by Providing Generic Type which names TKey.

there are some predefines common Key types in xDataService:

- **XBaseIntIdEntity;**
- **XBaseLongIdEntity;**
- **XBaseGuidIdEntity;**
- **XBaseStringIdEntity;**

so for Defining an Entity we can use them as Parent class.

```
public class XTest : XBaseGuidIdEntity {
    public string Firstname { get; set; }
    public string Lastname { get; set; }
}
```

next thing which required for each Entity is a Class for Configuring Entity Properties. this class is used to Describe each property Mapping to Table Column.

this is Optional step. but if you need, you have to extends your Entities Configurations from **XBaseEntityTypeConfiguration** class.

```
public class XTestTypeConfiguration : XBaseEntityTypeConfiguration<XTest, Guid>
{
    public override void Configure(EntityTypeBuilder<XTest> builder)
    { }
}
```

Context

if your DataLayer required to use EF Core. next step is to Create a DbContext.

for using xDataService features you have to extends your Database Context from XDbContext.

```
public class XSaherElmDbContext : XDbContext {
    //
    #region DbSets ...
    public DbSet<XTest> Tests { get; set; }
    #endregion

    //
    #region Navigation Property Entities ...
    #endregion

    public XSaherElmDbContext(
        DbContextOptions options,
        XDataServiceConfiguration config,
        XEntityRegisterarHandlerService dynamicEntityHandlers
    ) : base(
        // Database nae in Data Service Configurations ...
        "xApiDb",
```

```

        options,
        config,
        dynamicEntityHandlers,
        // a List allowed Dynamic Entity Registerars in this Context ...
        new string[] { nameof(XStringEntityRegistrar) }
    )
{ }

//
public override void OnXModelCreating(ModelBuilder modelBuilder)
{ }

//
public override void OnXConfiguring(DbContextOptionsBuilder optionsBuilder)
{ }
}

```

Dynamic Entities

xDataService provides a solution for Dynamic Entity Registration in it's Contexts out of box. this is useful when some modules needs to persist data.

IXEntityRegistrar: an interface for Describe how to Provides Dynamic Entities. **XBaseEntityRegistrar**: an abstract of Implementing Dynamic Entity Providers.

```

public interface IXEntityRegistrar
{ }

public abstract class XBaseEntityRegistrar : IXEntityRegistrar
{ }

```

for example we have **xStringService** which it is a module for Provides String resources in an Application. this module required to has Data Persist using Specified Entities in Implemented Application Contexts.

```

public class XString : XBaseIntIDEntity
{
    [Required]
    [StringLength (10)]
    public string Language { get; set; }

    [Required]
    [StringLength (255)]
    public string ResourceTitle { get; set; }

    [Required]
    public string TranslatedValue { get; set; }
}

public class XStringEntityConfiguration : XBaseEntityTypeConfiguration<XString,

```

```

int>
{
    public override void Configure(EntityTypeBuilder<XString> builder)
    {
        builder.ToTable("Strings");
    }
}

public class XStringEntityRegistrar : XBaseEntityRegistrar, IEntityRegistrar
{
    public XStringEntityRegistrar()
    : base(nameof(XStringEntityRegistrar))
    {
        //
        // Add XString Entity ...
        AddEntity<XString, int>();
    }

    /// <summary>
    /// Configure Entities ...
    /// </summary>
    /// <param name="modelBuilder"></param>
    public override void ConfigureEntities(ModelBuilder modelBuilder)
    {
        base.ConfigureEntities(modelBuilder);
    }
}

```

by this mechanism, provided Entity Dynamically Registered on Application DbContext automatically.

Repository Pattern

by this feature you can manipulate specified Entity using Repository Pttren implementation.

for each exists Entity, you need to Create a Repository:

- **IXBaseRepository<TEntity, TKey>**: a base interface which defines all Commons Repository Patterns Actions.
- **XBaseEFRepository<TEntity, TKey, TContext>**: a base Implementation of Commons Repository Patterns Actions Specified Using EF Core.
- **XBaseMongoRepository<TEntity, TKey>**: a base Implementation of Commons Repository Patterns Actions for MongoDB.
- **XBaseInMemoryRepository<TEntity, TKey>**: a base Implementation of Commons Repository Patterns Actions for InMemory Data Stores.

Key Generator

if you need to using KeyGenerator Services based on your Entity you have to Create Custom Key Generator based on each Entity:

- **IXKeyGenerator<TEntity, TKey>**: base interface for Providing Specified Key based Entities ID Generator.
- **XKeyGenerator<TEntity, TKey>**: base KeyGenerator Implementation abstraction.

based on Default Implemented Key types for XBased Entity, there some Default KeyGenerator Implementations for usage:

- XIntKeyGenerator
- XLongKeyGenerator
- XGuidKeyGenerator
- XStringKeyGenerator

```
public interface IXTKeyGenerator : IXKeyGenerator<XT, Guid>
{ }

public class XTKeyGenerator : XGuidKeyGenerator<XT>, IXTKeyGenerator
{ }
```

Events

events is a mechanism which notified when a Insert, Update or Delete action happens on Spacified Entity using Repository Implementation.

if you need to using RepositoryEvents Services based on your Entity you have to Create Custom Repository Events based on each Entity:

- **IXBaseRepositoryEvents**: a base interface for Describe several supported Repository Data Manipulation Events based on each Entity.
- **XBaseRepositoryEvents**: base Repository Event Implementation abstraction.

```
public interface IXTRepositoryEvents : IXBaseRepositoryEvents<XT>
{ }

public class XTRepositoryEvents : XBaseRepositoryEvents<XT>,
IXTRepositoryEvents
{ }
```

after preparing requirements you can define Repository Interface and Implementation based on selected Db Provider:

```
//
// Global Interface for Repository Pattern based on XT Entity ...
public interface IXTRepository : IXBaseRepository<XT, Guid>
{ }

//
```

```
// EF Repository Implementation based on XTest Entity ...
public class XTestEFRepository : XBaseEFRepository<XTest, Guid,
XSaherElmDbContext>, ITestRepository
{
    public XTestEFRepository(
        IUnitOfWork<XSaherElmDbContext> unitOfWork,
        XDataServiceConfiguration configuration,
        ITestKeyGenerator keyGenerator = null,
        ITestRepositoryEvents baseRepositoryEvents = null
    ) : base(
        unitOfWork,
        configuration,
        keyGenerator,
        baseRepositoryEvents
    )
    { }
}

//
// Mongo Db Repository Implementation based on XTest Entity ...
public class XTestMongoRepository : XBaseMongoRepository<XTest, Guid>
{
    public XTestMongoRepository(
        XDataBaseConfiguration dbConfiguration,
        XDataServiceConfiguration configuration,
        string collectionName = null,
        ITestKeyGenerator keyGenerator = null,
        ITestRepositoryEvents baseRepositoryEvents = null
    ) : base(
        dbConfiguration,
        configuration,
        collectionName,
        keyGenerator,
        baseRepositoryEvents
    )
    { }
}

//
// In Memory Store Repository Implementation based on XTest Entity ...
public class XTestInMemoryRepository : XBaseInMemoryRepository<XTest, Guid>
{
    public XTestInMemoryRepository(
        XDataServiceConfiguration configuration,
        ITestKeyGenerator keyGenerator = null,
        ITestRepositoryEvents baseRepositoryEvents = null
    ) : base(
        configuration,
        keyGenerator,
        baseRepositoryEvents
    )
    { }
}
```

as you can see in above Samples, the Repository Implementations has some Optional requirements which provides using Dependency Injections.

Specially if Key Generator and RepositoryEvents provided, Repository Implementation using them, if not, use Default mechanism for Keys and Events not Supported.

Data Seeding

in xDataService usages, when defining Specified Repositories, you can Use **DbSeeder** for adding Default Items based on each Entity in Database on App's Startup.

Seeding Mode

- **None**: ignore Seeding Items.
- **AddIfNotExists**: add items if not Exists.
- **AddOrUpdate**: add or update items.

for providing Seeding mechanism:

- **IXBaseDbSeeder<TEntity, TKey>**: a base interface for Describe Seeder's Actions based on specified Entity.
- **XBaseDbSeeder<TEntity, TKey>**: a base Seeder Implementation based on specified Entity.

if you need to Seed Items based on Each Entity of your Data Layer, you had to Create Seeder Services by following below:

```
public interface IXTestDbSeeder : IXBaseDbSeeder<XTest, Guid>
{
}

public class XTestDbSeeder : XBaseDbSeeder<XTest, Guid>, IXTestDbSeeder
{
    public XTestDbSeeder(
        IXTestRepository repository,
        XDataServiceConfiguration dataServiceConfiguration
    ) : base(
        database: "xApiDb",
        entity: nameof(XTest),
        repository: repository,
        dataServiceConfiguration: dataServiceConfiguration
    )
    {
    }
}
```

as you see Seeding only provided when using Repository Pattern.

GraphQL Support

one of the most important features of xDataService module which is provided for usage, is the ability of GraphQL usage of entities.

for this you have to follow some steps:

GraphQL Entity Type

first step is to define GraphType based on each entity:

- **XBaseGraphObjectType<TEntity, TKey>**: base abstraction implementation of entity graph type.

```
public class XTestGraphType : XBaseGraphObjectType<XTest, Guid>
{
    public XTestGraphType() : base()
    {
        Field(x => x.Firstname);
        Field(x => x.Lastname);
    }
}
```

GraphQLTypeHelper

each entity graph type must have a GraphQLType helper for providing requirements of GraphQL in the data layer.

- **IXBaseGraphQLTypeHelper<TEntity, TKey>**: a based interface for requirement.
- **XBaseGraphQLTypeHelper<TEntity, TKey>**: a based implementation for requirement.

```

public interface IXTestGraphQLTypeHelper : IXBaseGraphQLTypeHelper<XTest, Guid>
{ }

public class XTestGraphQLTypeHelper : XBaseGraphQLTypeHelper<XTest, Guid>,
IXTestGraphQLTypeHelper
{
    public XTestGraphQLTypeHelper(
        XDataServiceConfiguration configuration
    ) : base(configuration)
    {
    }

    public override string GetInQueryCollectionName()
    {
        return "tests";
    }

    public override string GetInQuerySingleName()
    {
        return "test";
    }
}

```

GraphQLQuery

each Entity has Specified Queries for Support Data Manipulations and Supports them.

all supported GraphQL actions Defines in this class.

- **IXBaseGraphQLQuery<TEntity, TKey, TGraph, TGraphKey>**: a base interface for Describing some Defaults Repository Actions for GraphQL.
- **XBaseGraphQLQuery<TEntity, TKey, TGraph, TGraphKey>**: a base Implementation abstraction for providing some Defaults Repository Actions for GraphQL.

for each Entity in your Data Layer you have to Define a GraphQL Query for Providing Query Actions.

```

public class XTestGraphQLQuery : XBaseGraphQLQuery<XTest, Guid, XTestGraphType,
GuidGraphType>
{
    public XTestGraphQLQuery(
        XDataServiceConfiguration configuration,
        IXTestRepository repository,
        IXTestGraphQLTypeHelper helper
    ) : base(
        configuration,
        repository,
        helper
    )
}

```

```
{ }
}
```

GraphQLSchema

a Graph Schema Provides all Requirements of a GraphQL endpoint.

based on each Entity, you have to Provides a GraphQL endpoint using Schema.

```
public class XTestGraphSchema : Schema
{
    public XTestGraphSchema(IServiceProvider services) : base(services)
    {
        Query = (XTestGraphQuery)services.GetService(typeof(XTestGraphQuery));
    }
}
```

Configure xDataService

all of Requirements for Data Layer must be Configured using appSettings and provided by IConfiguration to Modules.

fo using xDataService you Have to Configure the Service in you appSettings.json file.

this configurations mapped to **XDataServiceConfiguration** class.

```
/// <summary>
/// Represent Configurations of DataService Module ...
/// </summary>
public partial class XDataServiceConfiguration
{
    /// <summary>
    /// the base path for providing GraphQL ...
    /// </summary>
    /// <value></value>
    public string GraphQLBasePath { get; set; } = "/graphql";

    /// <summary>
    /// Data Bases Configuration ...
    /// </summary>
    /// <typeparam name="XDataBaseConfiguration"></typeparam>
    /// <returns></returns>
    public IDictionary<string, XDataBaseConfiguration> Databases { get; set; } =
    new Dictionary<string, XDataBaseConfiguration>();

    /// <summary>
    /// Enable Soft Delete Entities or Not ...
    /// </summary>
    /// <value></value>
```

```

public bool EnableSoftDelete { get; set; }

/// <summary>
/// Specified Seeding Mode ...
/// </summary>
public XDbSeedingMode SeedingMode { get; set; } = XDbSeedingMode.None;

/// <summary>
/// Enable Tracking of Entities ...
/// Only Used on EFCore ...
/// </summary>
/// <value></value>
public bool EnableTracking { get; set; } = false;

/// <summary>
/// Enable Logging Details of Errors ...
/// Only Used on EFCore ...
/// </summary>
/// <value></value>
public bool EnableDetailedErrors { get; set; } = false;

/// <summary>
/// Enable Logging Sensitive Data ...
/// Only Used on EFCore ...
/// </summary>
/// <value></value>
public bool EnableSensitiveDataLogging { get; set; } = false;

/// <summary>
/// this is a way to provide Default Pagination Data on XQuery based requests
...
/// </summary>
/// <returns></returns>
public PagingConfiguration PagingConfiguration { get; set; } = new
PagingConfiguration();
}

/// <summary>
/// Describe a Database ...
/// </summary>
public class XDataBaseConfiguration
{
    /// <summary>
    /// Provider Type ...
    /// </summary>
    /// <value></value>
    public XDbProviders Provider { get; set; }

    /// <summary>
    /// Connection String which provide Requires Data to Connect to Db Provider
    ...
    /// </summary>
    /// <value></value>
    public string ConnectionString { get; set; }

```

```

    /// <summary>
    /// Holds Seeding items ...
    /// nameof(TEntity) => TEntity
    /// </summary>
    public IDictionary<string, List<object>> SeedItems { get; set; } = new
Dictionary<string, List<object>>();
}

    /// <summary>
    /// this is a way to provide Default Pagination Data on XQuery based requests ...
    /// </summary>
    public partial class PagingConfiguration
    {
        /// <summary>
        /// Default Page Size ...
        /// </summary>
        /// <value></value>
        public int DefaultPageSize { get; set; } =
XDataServiceConstants.DEFAULT_PAGE_SIZE;

        /// <summary>
        /// restrict Maximum Page Size ...
        /// </summary>
        /// <value></value>
        public int MaxAvailablePageSize { get; set; } =
XDataServiceConstants.MAX_AVAILABLE_PAGE_SIZE;

        /// <summary>
        /// restrice Minimum Page Size ...
        /// </summary>
        /// <value></value>
        public int MinAvailablePageSize { get; set; } =
XDataServiceConstants.MIN_AVAILABLE_PAGE_SIZE;
    }

```

a sample Data Service Configuration must like this:

```

{
    ...
    "DataServiceConfiguration": {
    "Databases": {
        "XSaherElmDb": {
            "Provider": "SQLITE",
            "ConnectionString": "Filename=./Db/XSaherElmDb.db",
            "SeedItems": {
                "XTest": [
                    {
                        "Firstname": "Hadi",
                        "Lastname": "Khazae Asl"
                    },
                    {

```

```

        "Firstname": "Amir Ali",
        "Lastname": "Khazaei Asl"
    }
]
}
}
},
"EnableTracking": true,
"EnableSoftDelete": false,
"EnableDetailedErrors": false,
"SeedingMode": "AddIfNotExists",
"EnableSensitiveDataLogging": true,
"PagingConfiguration": {
    "DefaultPageSize": 20,
    "MaxAvailablePageSize": 400,
    "MinAvailablePageSize": 5
},
"GraphQLBasePath": "/graphs"
},
...
}

```

as you know the Configurations is a Multi Context implementations. and also in Seed Items each Collection maps to Entity Names per DataBase. and also Configurations provided based on each Database.

next step after Configuring DataService is to Scribe Each Database by providing Type safe Definitions for Dependency Injection Container using following Provided Classes.

```

/// <summary>
/// Data Base Descriptor ...
/// </summary>
public interface IXDatabaseDescriptor
{
    ...
}

/// <summary>
/// Data Base Descriptor ...
/// </summary>
/// <typeparam name="TContext"></typeparam>
public interface IXDatabaseDescriptor<TContext> : IXDatabaseDescriptor
where TContext : XDbContext
{
    ...
}

```

```

/// <summary>
/// Data Base Descriptor ...
/// </summary>

```

```

public abstract class XDatabaseDescriptor : IXDatabaseDescriptor
{
    ...
}

/// <summary>
/// Data Base Descriptor ...
/// </summary>
/// <typeparam name="TContext"></typeparam>
public abstract class XDatabaseDescriptor<TContext> : XDatabaseDescriptor,
IXDatabaseDescriptor<TContext>
where TContext : XDbContext
{
    ...
}

```

```

/// <summary>
/// a Global non Generic Repository Descriptor ...
/// </summary>
public class XRepositoryDescriptor
{
    ...
}

/// <summary>
/// a Repository Descriptor ...
/// </summary>
/// <typeparam name="TEntity"></typeparam>
/// <typeparam name="TKey"></typeparam>
public class XRepositoryDescriptor<TEntity, TKey> : XRepositoryDescriptor
where TEntity : XBaseEntity<TKey>
{
    ...
}

/// <summary>
/// a Repository Descriptor ...
/// </summary>
public class XRepositoryDescriptor<
    TEntity,
    TKey,
    TKeyGeneratorService,
    TKeyGeneratorImplementation
> : XRepositoryDescriptor<TEntity, TKey>
where TEntity : XBaseEntity<TKey>
where TKeyGeneratorService : IXKeyGenerator<TEntity, TKey>
where TKeyGeneratorImplementation : XKeyGenerator<TEntity, TKey>
{
    ...
}

```

```

/// <summary>
/// a Repository Descriptor ...
/// </summary>
public class XRepositoryDescriptor<
    TEntity,
    TKey,
    TKeyGeneratorService,
    TKeyGeneratorImplementation,
    TEventsService,
    TEventsImplementation
> : XRepositoryDescriptor<
    TEntity,
    TKey,
    TKeyGeneratorService,
    TKeyGeneratorImplementation
>
where TEntity : XBaseEntity<TKey>
where TEventsService : IXBaseRepositoryEvents<TEntity>
where TKeyGeneratorService : IXKeyGenerator<TEntity, TKey>
where TEventsImplementation : XBaseRepositoryEvents<TEntity>
where TKeyGeneratorImplementation : XKeyGenerator<TEntity, TKey>
{
    ...
}

/// <summary>
/// a Repository Descriptor ...
/// </summary>
public class XRepositoryDescriptor<
    TEntity,
    TKey,
    TKeyGeneratorService,
    TKeyGeneratorImplementation,
    TEventsService,
    TEventsImplementation,
    TGraph,
    TGraphKey,
    TGraphQuery,
    TGraphTypeHelperService,
    TGraphTypeHelperImplementation,
    TGraphSchema
> : XRepositoryDescriptor<
    TEntity,
    TKey,
    TKeyGeneratorService,
    TKeyGeneratorImplementation,
    TEventsService,
    TEventsImplementation
>
where TGraphSchema : Schema
where TGraphKey : IGraphType
where TEntity : XBaseEntity<TKey>
where TGraph : XBaseGraphObjectType<TEntity, TKey>
where TEventsService : IXBaseRepositoryEvents<TEntity>

```

```

where TKeyGeneratorService : IXKeyGenerator<TEntity, TKey>
where TEventsImplementation : XBaseRepositoryEvents<TEntity>
where TKeyGeneratorImplementation : XKeyGenerator<TEntity, TKey>
where TGraphTypeHelperService : IXBaseGraphQLTypeHelper<TEntity, TKey>
where TGraphQuery : XBaseGraphQLQuery<TEntity, TKey, TGraph, TGraphKey>
where TGraphTypeHelperImplementation : XBaseGraphQLTypeHelper<TEntity, TKey>
{
    ...
}

/// <summary>
/// a Repository Descriptor ...
/// </summary>
public class XInMemoryRepositoryDescriptor<
    TEntity,
    TKey,
    TKeyGeneratorService,
    TKeyGeneratorImplementation,
    TEventsService,
    TEventsImplementation,
    TGraph,
    TGraphKey,
    TGraphQuery,
    TGraphTypeHelperService,
    TGraphTypeHelperImplementation,
    TGraphSchema,
    TRepositoryService,
    TRepositoryImplementation
> : XRepositoryDescriptor<
    TEntity,
    TKey,
    TKeyGeneratorService,
    TKeyGeneratorImplementation,
    TEventsService,
    TEventsImplementation,
    TGraph,
    TGraphKey,
    TGraphQuery,
    TGraphTypeHelperService,
    TGraphTypeHelperImplementation,
    TGraphSchema
>
where TGraphSchema : Schema
where TGraphKey : IGraphType
where TEntity : XBaseEntity<TKey>
where TGraph : XBaseGraphObjectType<TEntity, TKey>
where TEventsService : IXBaseRepositoryEvents<TEntity>
where TKeyGeneratorService : IXKeyGenerator<TEntity, TKey>
where TRepositoryService : IXBaseRepository<TEntity, TKey>
where TEventsImplementation : XBaseRepositoryEvents<TEntity>
where TKeyGeneratorImplementation : XKeyGenerator<TEntity, TKey>
where TGraphTypeHelperService : IXBaseGraphQLTypeHelper<TEntity, TKey>
where TRepositoryImplementation : XBaseInMemoryRepository<TEntity, TKey>
where TGraphQuery : XBaseGraphQLQuery<TEntity, TKey, TGraph, TGraphKey>

```

```

where TGraphTypeHelperImplementation : XBaseGraphQLTypeHelper<TEntity, TKey>
{
    ...
}

/// <summary>
/// a Repository Descriptor ...
/// </summary>
public class XInMemoryRepositoryDescriptor<
    TEntity,
    TKey,
    TKeyGeneratorService,
    TKeyGeneratorImplementation,
    TEventsService,
    TEventsImplementation,
    TGraph,
    TGraphKey,
    TGraphQuery,
    TGraphTypeHelperService,
    TGraphTypeHelperImplementation,
    TGraphSchema,
    TRepositoryService,
    TRepositoryImplementation,
    TSeederService,
    TSeederImplementation
> : XRepositoryDescriptor<
    TEntity,
    TKey,
    TKeyGeneratorService,
    TKeyGeneratorImplementation,
    TEventsService,
    TEventsImplementation,
    TGraph,
    TGraphKey,
    TGraphQuery,
    TGraphTypeHelperService,
    TGraphTypeHelperImplementation,
    TGraphSchema
>
where TGraphSchema : Schema
where TGraphKey : IGraphType
where TEntity : XBaseEntity<TKey>
where TGraph : XBaseGraphObjectType<TEntity, TKey>
where TSeederService : IXBaseDbSeeder<TEntity, TKey>
where TEventsService : IXBaseRepositoryEvents<TEntity>
where TKeyGeneratorService : IXKeyGenerator<TEntity, TKey>
where TRepositoryService : IXBaseRepository<TEntity, TKey>
where TSeederImplementation : XBaseDbSeeder<TEntity, TKey>
where TEventsImplementation : XBaseRepositoryEvents<TEntity>
where TKeyGeneratorImplementation : XKeyGenerator<TEntity, TKey>
where TGraphTypeHelperService : IXBaseGraphQLTypeHelper<TEntity, TKey>
where TRepositoryImplementation : XBaseInMemoryRepository<TEntity, TKey>
where TGraphQuery : XBaseGraphQLQuery<TEntity, TKey, TGraph, TGraphKey>
where TGraphTypeHelperImplementation : XBaseGraphQLTypeHelper<TEntity, TKey>

```

```

{
    ...
}

/// <summary>
/// a Repository Descriptor ...
/// </summary>
public class XMongoRepositoryDescriptor<
    TEntity,
    TKey,
    TKeyGeneratorService,
    TKeyGeneratorImplementation,
    TEventsService,
    TEventsImplementation,
    TGraph,
    TGraphKey,
    TGraphQuery,
    TGraphTypeHelperService,
    TGraphTypeHelperImplementation,
    TGraphSchema,
    TRepositoryService,
    TRepositoryImplementation
> : XRepositoryDescriptor<
    TEntity,
    TKey,
    TKeyGeneratorService,
    TKeyGeneratorImplementation,
    TEventsService,
    TEventsImplementation,
    TGraph,
    TGraphKey,
    TGraphQuery,
    TGraphTypeHelperService,
    TGraphTypeHelperImplementation,
    TGraphSchema
>
where TGraphSchema : Schema
where TGraphKey : IGraphType
where TEntity : XBaseEntity<TKey>
where TGraph : XBaseGraphObjectType<TEntity, TKey>
where TEventsService : IXBaseRepositoryEvents<TEntity>
where TKeyGeneratorService : IXKeyGenerator<TEntity, TKey>
where TRepositoryService : IXBaseRepository<TEntity, TKey>
where TEventsImplementation : XBaseRepositoryEvents<TEntity>
where TKeyGeneratorImplementation : XKeyGenerator<TEntity, TKey>
where TRepositoryImplementation : XBaseMongoRepository<TEntity, TKey>
where TGraphTypeHelperService : IXBaseGraphQLTypeHelper<TEntity, TKey>
where TGraphQuery : XBaseGraphQLQuery<TEntity, TKey, TGraph, TGraphKey>
where TGraphTypeHelperImplementation : XBaseGraphQLTypeHelper<TEntity, TKey>
{
    ...
}

/// <summary>

```

```

/// a Repository Descriptor ...
/// </summary>
public class XMongoRepositoryDescriptor<
    TEntity,
    TKey,
    TKeyGeneratorService,
    TKeyGeneratorImplementation,
    TEventsService,
    TEventsImplementation,
    TGraph,
    TGraphKey,
    TGraphQuery,
    TGraphTypeHelperService,
    TGraphTypeHelperImplementation,
    TGraphSchema,
    TRepositoryService,
    TRepositoryImplementation,
    TSeederService,
    TSeederImplementation
> : XRepositoryDescriptor<
    TEntity,
    TKey,
    TKeyGeneratorService,
    TKeyGeneratorImplementation,
    TEventsService,
    TEventsImplementation,
    TGraph,
    TGraphKey,
    TGraphQuery,
    TGraphTypeHelperService,
    TGraphTypeHelperImplementation,
    TGraphSchema
>
where TGraphSchema : Schema
where TGraphKey : IGraphType
where TEntity : XBaseEntity<TKey>
where TGraph : XBaseGraphObjectType<TEntity, TKey>
where TSeederService : IXBaseDbSeeder<TEntity, TKey>
where TEventsService : IXBaseRepositoryEvents<TEntity>
where TSeederImplementation : XBaseDbSeeder<TEntity, TKey>
where TKeyGeneratorService : IXKeyGenerator<TEntity, TKey>
where TRepositoryService : IXBaseRepository<TEntity, TKey>
where TEventsImplementation : XBaseRepositoryEvents<TEntity>
where TKeyGeneratorImplementation : XKeyGenerator<TEntity, TKey>
where TRepositoryImplementation : XBaseMongoRepository<TEntity, TKey>
where TGraphTypeHelperService : IXBaseGraphQLTypeHelper<TEntity, TKey>
where TGraphQuery : XBaseGraphQLQuery<TEntity, TKey, TGraph, TGraphKey>
where TGraphTypeHelperImplementation : XBaseGraphQLTypeHelper<TEntity, TKey>
{
    ...
}

/// <summary>
/// a Repository Descriptor ...

```

```

/// </summary>
public class XEFRepositoryDescriptor<
    TEntity,
    TKey,
    TKeyGeneratorService,
    TKeyGeneratorImplementation,
    TEventsService,
    TEventsImplementation,
    TGraph,
    TGraphKey,
    TGraphQuery,
    TGraphTypeHelperService,
    TGraphTypeHelperImplementation,
    TGraphSchema,
    TRepositoryService,
    TRepositoryImplementation,
    TContext
> : XRepositoryDescriptor<
    TEntity,
    TKey,
    TKeyGeneratorService,
    TKeyGeneratorImplementation,
    TEventsService,
    TEventsImplementation,
    TGraph,
    TGraphKey,
    TGraphQuery,
    TGraphTypeHelperService,
    TGraphTypeHelperImplementation,
    TGraphSchema
>
where TGraphSchema : Schema
where TContext : XDbContext
where TGraphKey : IGraphType
where TEntity : XBaseEntity<TKey>
where TGraph : XBaseGraphObjectType<TEntity, TKey>
where TEventsService : IXBaseRepositoryEvents<TEntity>
where TKeyGeneratorService : IXKeyGenerator<TEntity, TKey>
where TRepositoryService : IXBaseRepository<TEntity, TKey>
where TEventsImplementation : XBaseRepositoryEvents<TEntity>
where TKeyGeneratorImplementation : XKeyGenerator<TEntity, TKey>
where TGraphTypeHelperService : IXBaseGraphQLTypeHelper<TEntity, TKey>
where TGraphQuery : XBaseGraphQLQuery<TEntity, TKey, TGraph, TGraphKey>
where TRepositoryImplementation : XBaseEFRepository<TEntity, TKey, TContext>
where TGraphTypeHelperImplementation : XBaseGraphQLTypeHelper<TEntity, TKey>
{
    ...
}

/// <summary>
/// a Repository Descriptor ...
/// </summary>
public class XEFRepositoryDescriptor<
    TEntity,

```

```

    TKey,
    TKeyGeneratorService,
    TKeyGeneratorImplementation,
    TEventsService,
    TEventsImplementation,
    TGraph,
    TGraphKey,
    TGraphQuery,
    TGraphTypeHelperService,
    TGraphTypeHelperImplementation,
    TGraphSchema,
    TRepositoryService,
    TRepositoryImplementation,
    TContext,
    TSeederService,
    TSeederImplementation
> : XRepositoryDescriptor<
    TEntity,
    TKey,
    TKeyGeneratorService,
    TKeyGeneratorImplementation,
    TEventsService,
    TEventsImplementation,
    TGraph,
    TGraphKey,
    TGraphQuery,
    TGraphTypeHelperService,
    TGraphTypeHelperImplementation,
    TGraphSchema
>
where TGraphSchema : Schema
where TContext : XDbContext
where TGraphKey : IGraphType
where TEntity : XBaseEntity<TKey>
where TGraph : XBaseGraphObjectType<TEntity, TKey>
where TSeederService : IXBaseDbSeeder<TEntity, TKey>
where TEventsService : IXBaseRepositoryEvents<TEntity>
where TSeederImplementation : XBaseDbSeeder<TEntity, TKey>
where TKeyGeneratorService : IXKeyGenerator<TEntity, TKey>
where TRepositoryService : IXBaseRepository<TEntity, TKey>
where TEventsImplementation : XBaseRepositoryEvents<TEntity>
where TKeyGeneratorImplementation : XKeyGenerator<TEntity, TKey>
where TGraphTypeHelperService : IXBaseGraphQLTypeHelper<TEntity, TKey>
where TGraphQuery : XBaseGraphQLQuery<TEntity, TKey, TGraph, TGraphKey>
where TRepositoryImplementation : XBaseEFRepository<TEntity, TKey, TContext>
where TGraphTypeHelperImplementation : XBaseGraphQLTypeHelper<TEntity, TKey>
{
    ...
}

```

based on using DbContext or not, you have to using above Descriptors for Describing your Databases.

```

//
// Describe xSaherElmDatabase ...
// XSaherElmDbContext => a Db Context implementation which extends XDbContext ...
public class XSaherElmDatabaseDescriptor :
XDatabaseDescriptor<XSaherElmDbContext>, IXDatabaseDescriptor<XSaherElmDbContext>
{
    public XSaherElmDatabaseDescriptor() : base(
        //
        // Name of Database which Configured in DataServiceConfigurations
        // in App Settings json File ...
        name: "XSaherElmDb",
        //
        // Describe all Exists Repositories ...
        repositories: new List<XRepositoryDescriptor>
        {
            //
            // XTest Entity ...
            // Since we Use EF Core, Create instance of
            // XEFRepository ...
            new XEFRepositoryDescriptor<
                XTest,                // Entity Type
                Guid,                 // Entity Key Type
                IXTestKeyGenerator,    // Repository Key Generation Interface
                XTestKeyGenerator,     // Repository Key Generation
                IXTestRepositoryEvents, // Repository Events Interface Type
                XTestRepositoryEvents, // Repository Events Implementation
                XTestGraphType,        // Entity GraphQL Object Type
                GuidGraphType,         // Entity GraphQL Key Object Type
                XTestGraphQuery,       // Entity GraphQL Query Type
                IXTestGraphQLTypeHelper, // Entity GraphQL Type Helper
                XTestGraphQLTypeHelper, // Entity GraphQL Type Helper
                XTestGraphSchema,       // Entity GraphQL Schema Type
                IXTestRepository,       // Entity Repository Interface Type
                XTestEFRepository,     // Entity Repository Implementation
                XSaherElmDbContext,    // Entity DB Context Type
                IXTestDbSeeder,        // Entity Data Seeder Interface Type
                XTestDbSeeder           // Entity Data Seeder Implementation
            >
            {}
        }
    )
    {
        //
        // Options Builder for EF Core which used to Provides

```

```

        // Migration Assembly to DataService ...
        OptionsBuilder = b => b.MigrationsAssembly("xApi");
    }
}

```

as you can see, in above Descriptor we use all Implemented Classes and Interfaces.

Note: there are Several Constructors of Repository Descriptors which you can use to Describe Several Types of Databases.

Register DataService

after preparation of all requirements, final steps is Register DataService.

there are two main step:

- **DI Registration:** in this phase, all Database Descriptors used for Registering all Services and Implementation of them.
- **Middleware Usage:** in this phase, all registre Repositories used for Seeding Data if Configured, and also Register GraphQL Supports if provided and Make available **/ui/playground** and **/ui/voyager** paths for GraphQL UI Supports when inside Development Environment.

```

public class Startup
{
    //
    private readonly XSaherElmDatabaseDescriptor saherelmDatabaseDescriptor;

    //
    public Startup(IConfiguration configuration)
    {
        //
        Configuration = configuration;

        //
        // Instance Database Descriptors Here ...
        saherelmDatabaseDescriptor = new XSaherElmDatabaseDescriptor();
    }

    //
    public void ConfigureServices(IServiceCollection services)
    {
        ...
        //
        // Registration Database ...
        // Each Database must Register in DI Container ...
        services.AddXDatabase(
            lifetime: lifeTime,
            configuration: Configuration,
            descriptor: saherelmDatabaseDescriptor
        );
        ...
    }
}

```

```
}

//
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    ...
    //
    // Use Database Middleware ...
    // Each Database must Used in Configuring Phase ...
    app.UseXDatabase(
        descriptor: saherelmDatabaseDescriptor,
        isDevelopmentEnvironment: env.IsDevelopment()
    );
    ...
}
}
```

Repository Service

a Repository Service is a Service based on XRepository Actions which Used Dto's instead of Entities.

Dto

a Dto (Data Transfer Object) is an instance of an Entity for Transferring across another Services for Business Logics.

XBaseEntityDto

all xDataService based Dtos must inherit from this base Class.

Id and Deleted Properties set to all TEntitys.

- Id: is Entity Primary Key;
- Deleted: used to Filter Soft Deleted Entities;

```
public abstract XBaseEntityDto<TKey> {}
```

using Key Type for Entity specified by Providing Generic Type which names TKey.

there are some predefines common Key types in xDataService:

- **XBaseIntIDEntityDto;**
- **XBaseLongIDEntityDto;**
- **XBaseGuidIDEntityDto;**
- **XBaseStringIDEntityDto;**

so for Defining an Entity we can use them as Parent class.

```
public class XTestDto : XBaseEntityDto<Guid>
{
    public string Firstname { get; set; }
    public string Lastname { get; set; }
}
```

- **IXBaseRepositoryService<TEntity, TDto, TKey>**: base interface for Describe Repository Service Actions.
- **XBaseRepositoryService<TEntity, TDto, TKey>**: base Abstraction of Repository Service Implementation .

foreach Repository, you have to inherit Repository Service from them:

```
public interface IXTestRepositoryService : IXBaseRepositoryService<XTest,
XTestDto, Guid>
{ }

public class XTestRepositoryService : XBaseRepositoryService<XTest, XTestDto,
Guid>, IXTestRepositoryService
{
    public XTestRepositoryService(
        IXTestRepository repository,
        IEnumerable<Profile> mapperProfiles = null
    ) : base(repository, mapperProfiles)
    { }
}
```

final step is Registration in DI Container:

```
...
service.AddScoped<IXTestRepositoryService, XTestRepositoryService>();
...
```

Data Controllers

a Data Controller provides all Default Requirements Actions for Base Repository Patterns Data Manipulation Actions based on Restfull Architecture.

there are two way to Provides Repositroy Actions through End Users:

Repository Controller

it usefull when you need to provides access to Specified Entity through Restfull Api.

- **IXBaseRepositoryController<TEntity, TKey>**: a base interface for Describing a Repository based Controller Actions.

- **XBaseRepositoryController<TEntity, TKey>**: a base Abstraction of Implementation of a Repository based Controller Actions.

```
public partial class XTestRepositoryController : XBaseRepositoryController<XTest, Guid>, IBaseRepositoryController<XTest, Guid> {
    public XTestRepositoryController(
        ILogger<XTestRepositoryController> logger,
        XAppConfiguration appConfiguration,
        XValidationProvider validationProvider,
        ITestRepository repository,
        Func<IQueryable<XTest>, IOrderedQueryable<XTest>> defaultOrderBuilder = null,
        Func<IQueryable<XTest>, IIncludableQueryable<XTest, object>> defaultIncludeBuilder = null
    ) : base(
        logger,
        appConfiguration,
        validationProvider,
        repository,
        defaultOrderBuilder,
        defaultIncludeBuilder
    ) { }
}
```

Service Controller

it usefull when you need to provides access to Dto of Specified Entity through Restfull Api.

- **IXBaseServiceController<TEntity, TDto, TKey>**: a base interface for Describing a Service based Controller Actions.
- **XBaseServiceController<TEntity, TDto, TKey>**: a base Abstraction of Implementation of a Service based Controller Actions.

```
public class XTestServiceController : XBaseServiceController<XTest, XTestDto, Guid>, IBaseServiceController<XTest, XTestDto, Guid> {
    public XTestServiceController(
        ILogger<XTestServiceController> logger,
        XAppConfiguration appConfiguration,
        XValidationProvider validationProvider,
        ITestRepositoryService repositoryService,
        Func<IQueryable<XTest>, IOrderedQueryable<XTest>> defaultOrderBuilder = null,
        Func<IQueryable<XTest>, IIncludableQueryable<XTest, object>> defaultIncludeBuilder = null
    ) : base(
        logger,
        appConfiguration,
        validationProvider,
```

```
        repositoryService,  
        defaultOrderBuilder,  
        defaultIncludeBuilder  
    )  
    { }  
}
```

Maintainer

Hadi Khazaee asl

<https://www.saherelm.ir>

hadi_khazaee_asl@yahoo.com