Table of Contents

[I. Hosts 4](#_Toc517772663)

[A. Accounts 4](#_Toc517772664)

[B. Dashboard 5](#_Toc517772665)

[C. MVC Areas 6](#_Toc517772666)

[1. UserManagers 6](#_Toc517772667)

[2. Users 7](#_Toc517772668)

[3. Admins 10](#_Toc517772669)

[D. Strategies Used 11](#_Toc517772670)

[1. BaseClasses 11](#_Toc517772671)

[2. ActionFilterAttribute – DisposableService and IServiceAwareController 13](#_Toc517772672)

[3. \*Utility – Mailer 13](#_Toc517772673)

[4. Asp.Identity 13](#_Toc517772674)

[5. IoC/DI – MEF 13](#_Toc517772675)

[6. PartialViews 14](#_Toc517772676)

[7. Bootstrap 3.3.7 (don’t upgrade to 4) 15](#_Toc517772677)

[8. FontAwesome 15](#_Toc517772678)

[9. JQuery/Ajax 15](#_Toc517772679)

[10. HTML Helpers 17](#_Toc517772680)

[11. Extension Methods 18](#_Toc517772681)

[12. Logging – NLog and Application Insights 18](#_Toc517772682)

[13. Async/Await 18](#_Toc517772683)

[14. LINQ via Fluent Api 19](#_Toc517772684)

[15. Generics 19](#_Toc517772685)

[16. Abstract factory – ClassFactory 19](#_Toc517772686)

[17. Razor default namespaces 19](#_Toc517772687)

[18. Bundling and Minification 19](#_Toc517772688)

[II. Business 20](#_Toc517772689)

[A. Engines 20](#_Toc517772690)

[1. Request Engines 20](#_Toc517772691)

[III. Data 21](#_Toc517772692)

[A. Ado.Net 21](#_Toc517772693)

[B. EF6 – Code First 21](#_Toc517772694)

[C. Soft Delete (optional) 22](#_Toc517772695)

[D. DataRepositories 23](#_Toc517772696)

[1. DataRepositorySoftDeleteInt 23](#_Toc517772697)

[E. Automatic data migrations 23](#_Toc517772698)

[IV. Code repository 24](#_Toc517772699)

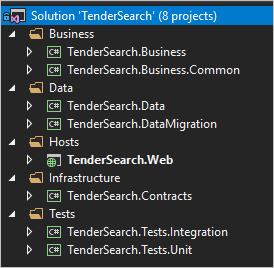
[V. Tests 24](#_Toc517772700)

[A. Unit Tests 24](#_Toc517772701)

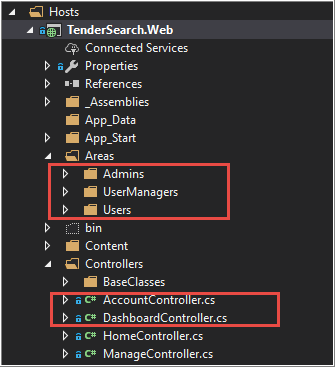
[B. Integration Tests 24](#_Toc517772702)

Solution Architecture Overview

In bird’s eye view, the solution is separated into 5 different areas namely: **Hosts**, **Business**, **Data**, **Infrastructure** *and* **Tests**.



# Hosts



## Accounts

Uses Asp.Identity to handle and manage user registration and roles.

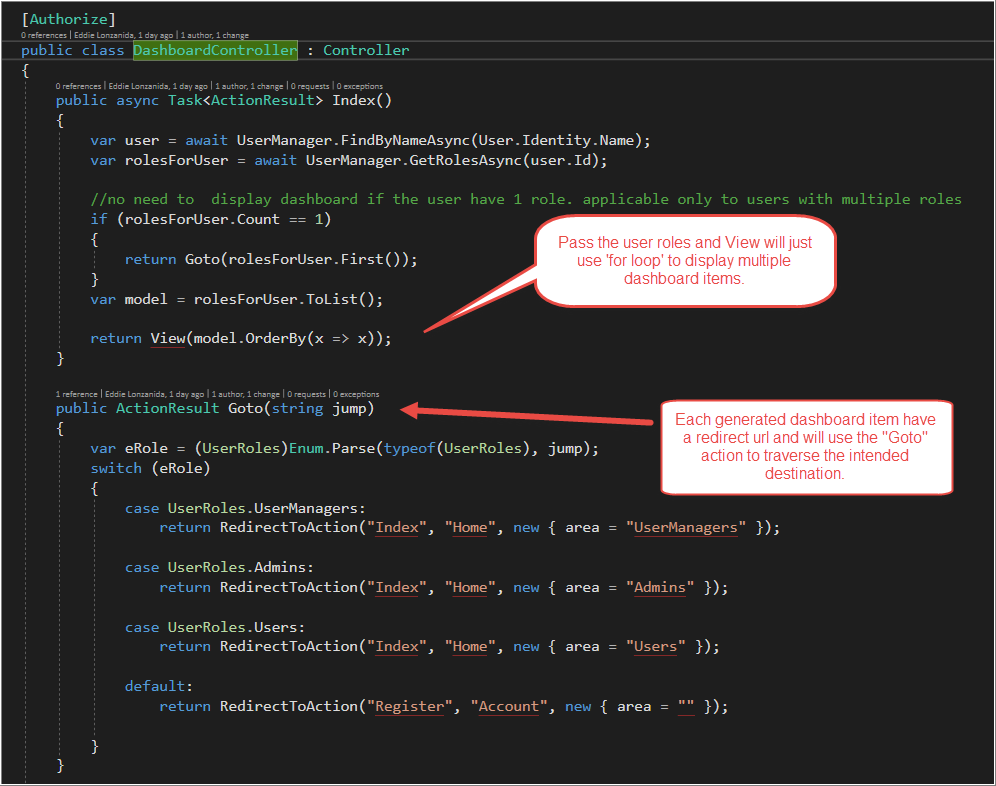
It is an out of the box feature from MVC template with a few modifications to suit the dashboard functionalities.

Some of the features :

1. Included usernames instead of email during login.
   1. Created entities to make it work.
2. Email notification.
3. Fixed some redirect links during the login and registration.

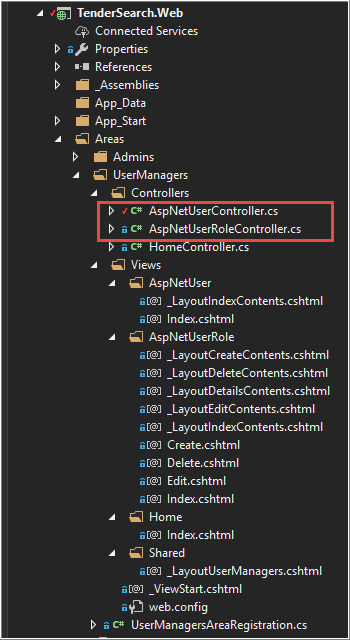
## Dashboard

After the user’s self registration, all the “UserManagers” will be notified via email. After which, role(s) will be assigned so the user can access the dashboard items. Behind the scenes, DashboardController will retrieve the assigned role(s) for the current user and perform some logic.



## MVC Areas

### UserManagers

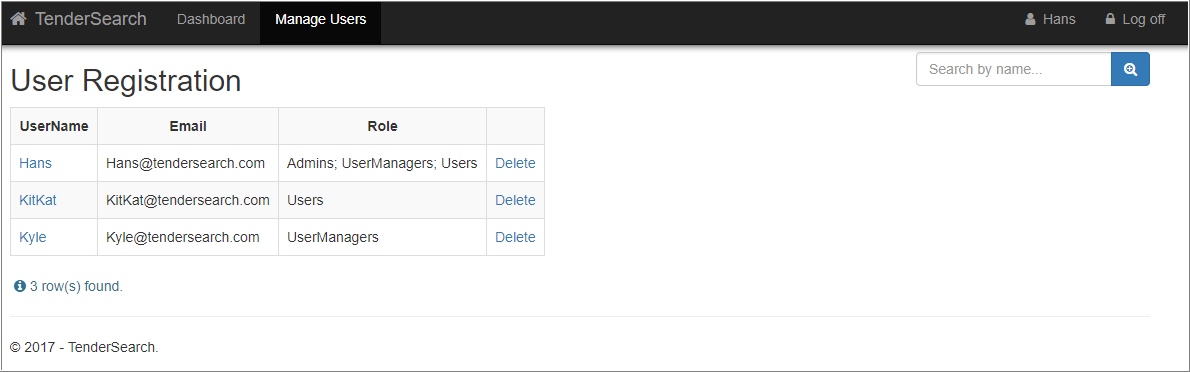
The email notification triggered by the user’s self-registration will contain a link redirecting to this page.

#### Controllers

Derived from UserManagerBase class. Used to perform “Roles” CRUD operations.

Below are the controllers found in this area:

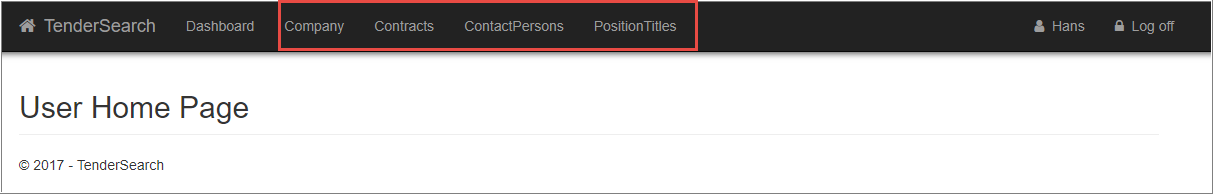
##### AspNetUserController

The landing page. Retrieve and display available roles for each user. Newly registered users will also appear here.

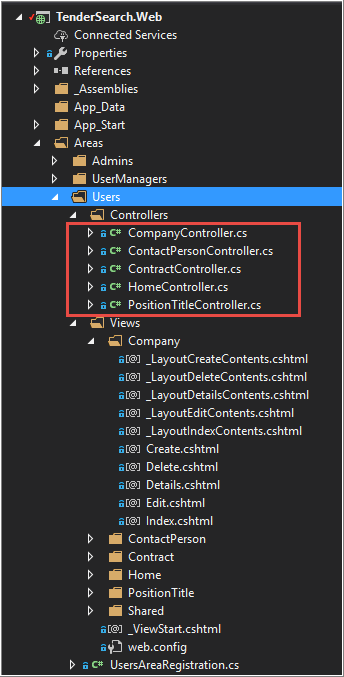
##### AspNetUserRoleController

CRUD operations for user roles.

### Users

This is where we can see the Company’s maintenance page along with other table maintenance controllers.

#### Controllers

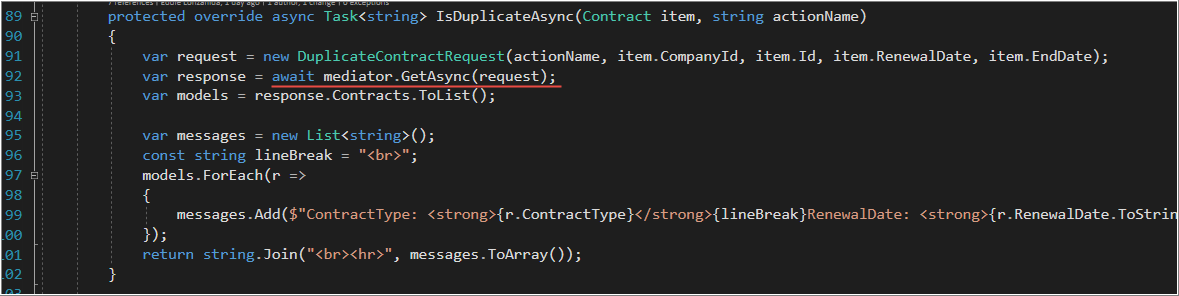
Derived from TableMaintenanceBaseclass. Used generics to perform CRUD operations.

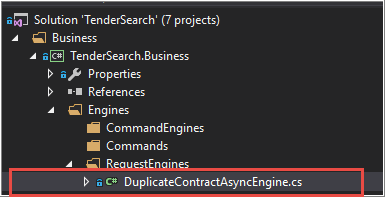
##### CompanyController

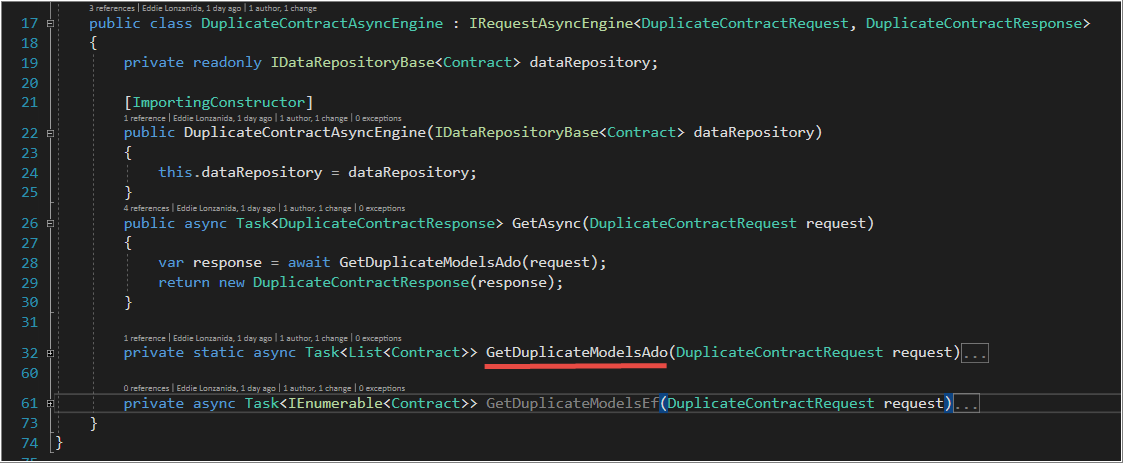
Derived from TableMaintenanceBase class.

##### ContractController

Derived from TableMaintenanceChildBase class.

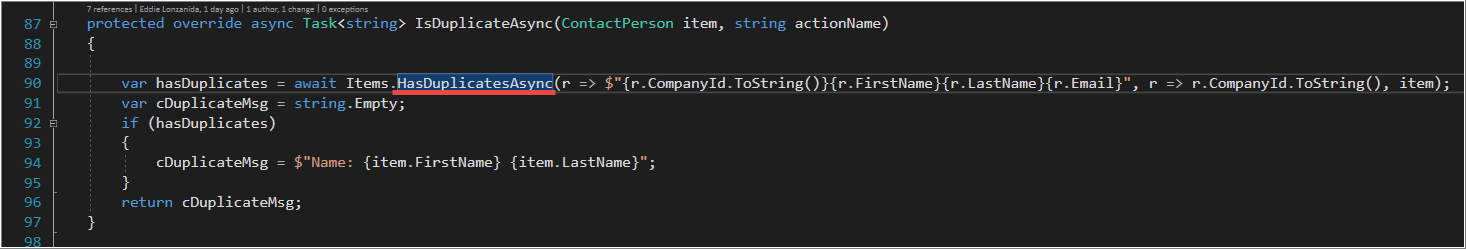
Overrides the IsDuplicateAsync for contract validation. Uses mediator pattern to execute the logic.

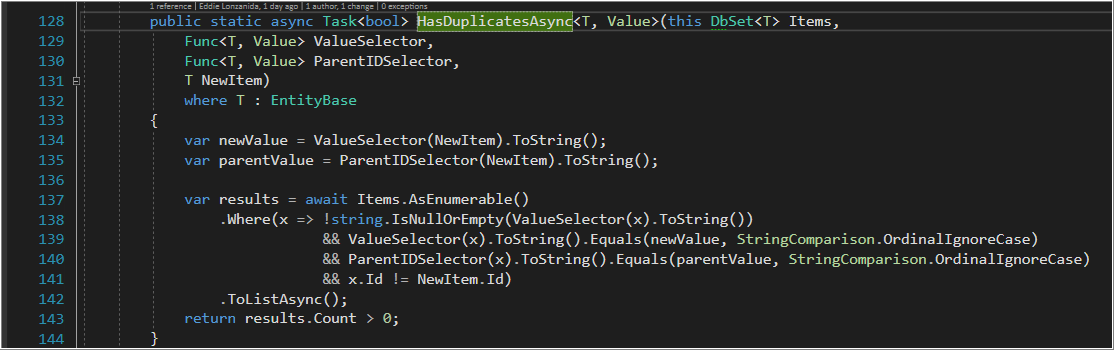
Validates valid contract via the DuplicateContractAsyncEngine:

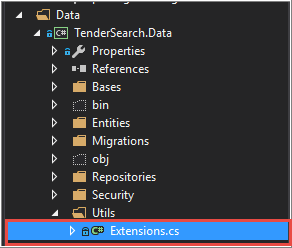
Uses Ado.Net to execute stored procedure: GetContractDuplicates

##### ContacPersonController

Derived from TableMaintenanceChildBase class.

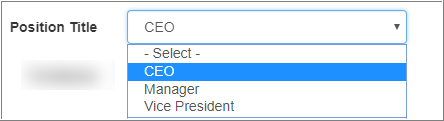
Overrides the IsDuplicateAsync for contract validation. Searching for duplicates using the combination of CompanyId, FirstName, LastName and Email.

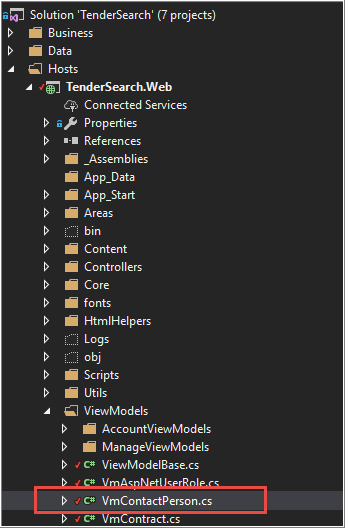
Uses the extension method HasDuplicatesAsync.

HasDuplicatesAsync is located in TenderSearch.Data

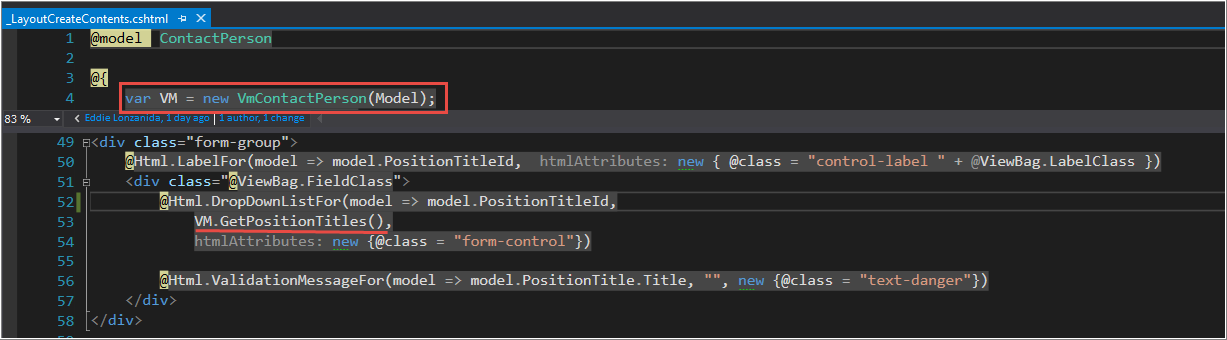
##### PositionTitleController

Derived from TableMaintenanceBaseclass

Just the normal CRUD operations for the dropdown menu.

The dropdown menus are retrieved from the database via the VmContactPerson located at:

VmContactPerson is derived from **ViewModelBase**

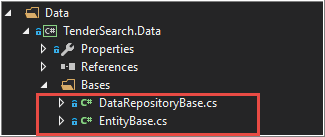
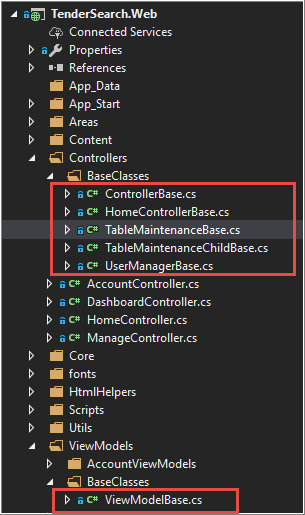
The VmContactPerson is consumed in the ContactPerson view:

### Admins

Just a place holder if something came-up.

## Strategies Used

### BaseClasses



#### HomeControllerBase

Used to determine if the Dashboard will be visible.

#### ControllerBase

Contains NavigationHelpers that accepts strongly typed parameters.

From:



To:



(Notice the removal of ‘magic strings’)

#### TableMaintenanceBase

Does all the heavy lifting during the Table Maintenance operations.

Such as:

* UI Search functionality with Intellisense
* CRUD operations
* Validation
* Error handling
* Etc..

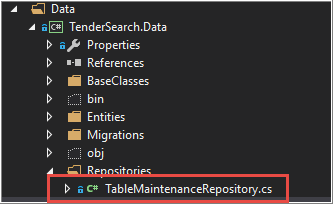
#### TableMaintenanceChildBase

Extra layer for child entities such as Contracts and ContactPersons. This will ensure the virtual methods in TableMaintenanceBase are implemented in Contracts and ContactPersons.

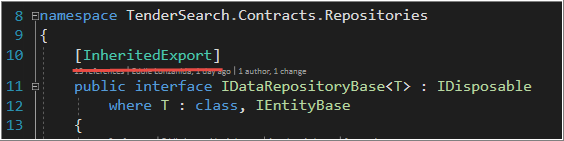
#### UserManagerBase

Does all the heavy lifting during the User Management operations. This is separated from the TableMaintenanceBase due to the fact that I’m dealing Asp.Identity wherein all, if not most of the internal operations are encapsulated.

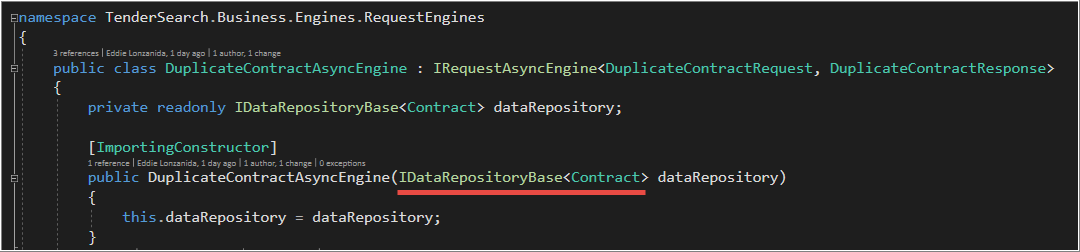
#### DataRepositoryBase

This demonstrate the use of DataRepositories such as TableMaintenanceRepository

Any class that inherits from DataRepositoryBase will be discoverable in MEF IoC/DI. Because of the InheritedExport attribute:



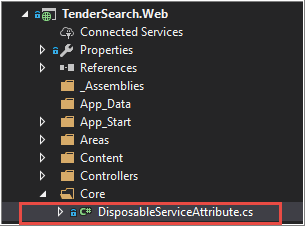
DataRepositoryBase also takes advantage of the new **MEF2** feature called ‘Open Generics’

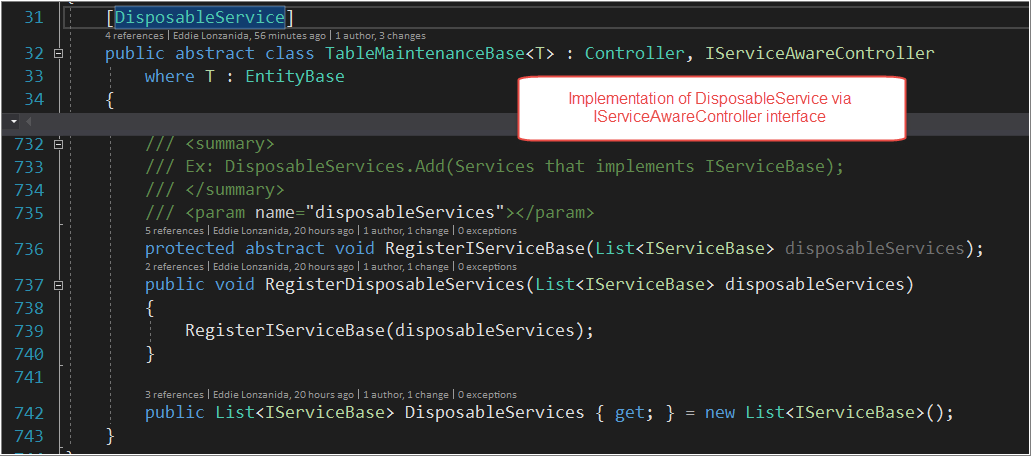


#### EntityBase

To ensure all entities have the following properties: Id, DateDeleted, DeletionReason

### ActionFilterAttribute – DisposableService and IServiceAwareController

This will ensure that all IServiceBase are explicitly disposed. Having memory leaks or undisposed long-running services is a developer’s nightmare.



### \*Utility – Mailer

A simple static class that generates mail messages.

### Asp.Identity

An ‘out of the box’ functionality from Microsoft which I modified a little bit to suit my needs.

### IoC/DI – MEF

#### [Mediator](https://github.com/EddLonzanida/Eml.Mediator.Demo)

A small yet powerful library that will allow to effectively translate business use cases into reusable blocks of code. Made possible through heavy usage of Generics and MEF2 discoverability features.

This will all come-in handy when migrating from an old platform into a new one. Because of its single responsibility feature, it will be possible to dissect large use cases into small maintainable chunks of code.

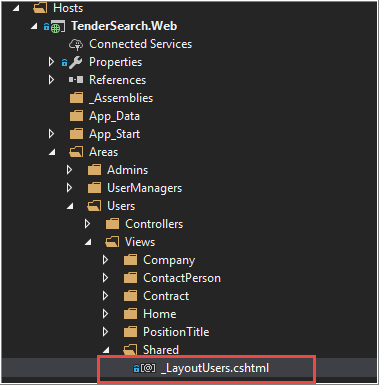
This will also help create highly testable codes.

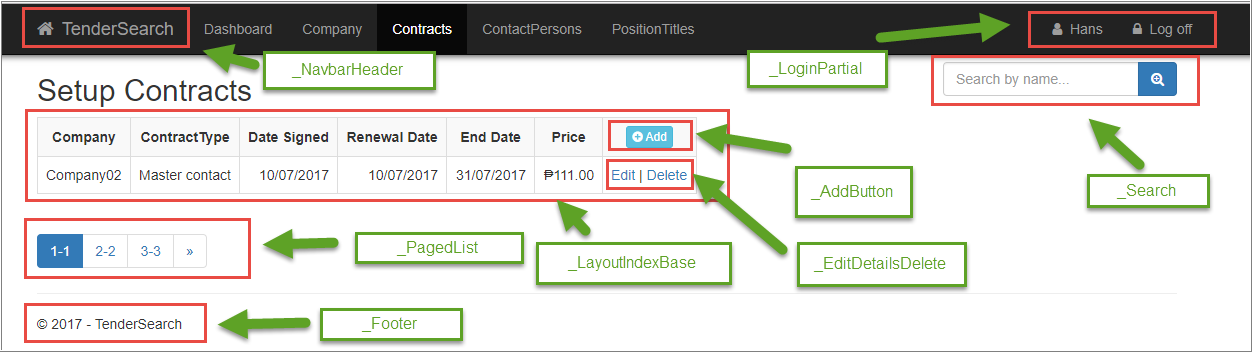
#### \*Modules

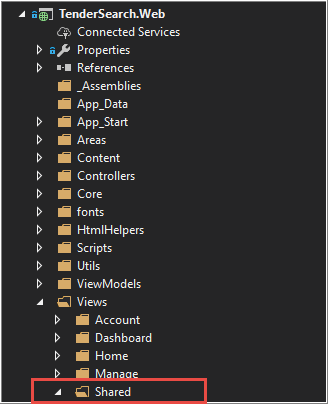
This feature will become apparent when a certain business process follows a recurring series of steps. (an example could be an accounting cycle – step1, step2,step3.. etc). A module is a representation of ‘a business process step’ converted into codes. A module can also reside on a separate project. This is handled by the PipelineFramework which relies heavily on modular pattern and discoverability. Combine this with the Mediator and a project will be a little bit easier to manage.

### PartialViews

To make the UI consistent, below is an illustration on how the whole page is constructed via PartialViews

Each MVC Area uses a separate Layout.

Each Layout page is basically composed of 8 partial pages:

Partial pages can be found in shared folder:

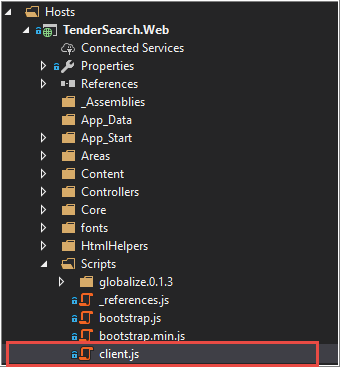
### Bootstrap 3.3.7 (don’t upgrade to 4)

For responsive layouts. It can handle different browsers multiplied by the number of different desktop and mobile devices. This is a must, I might say.

### FontAwesome

Cute icons ☺

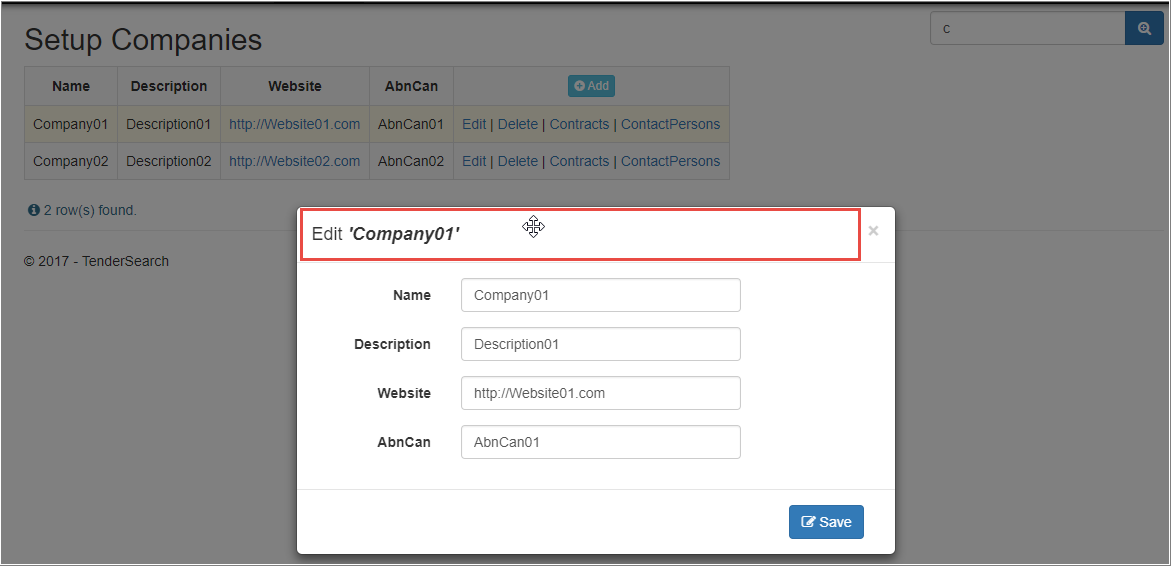
### JQuery/Ajax

All my javacripts can be found in:

Used to facilitate front-end interactions:

#### Intellisense

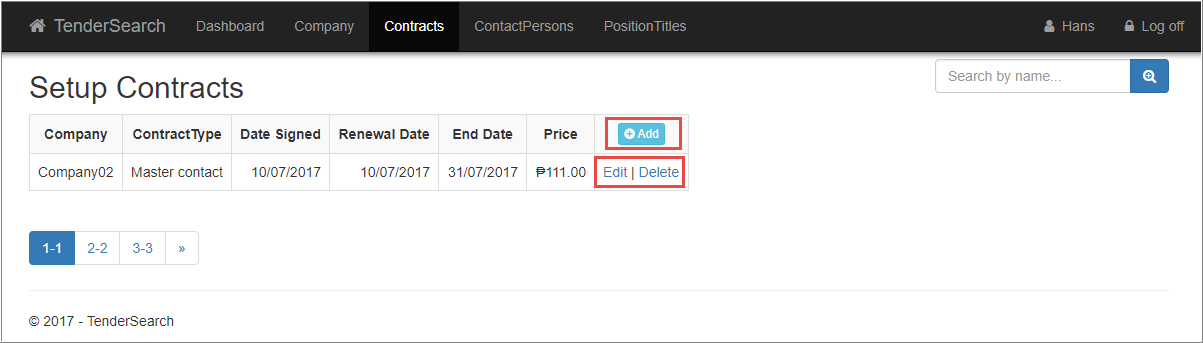
#### Draggable dialog



#### Paging

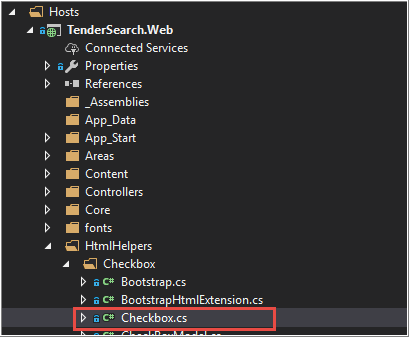
#### Date pickers

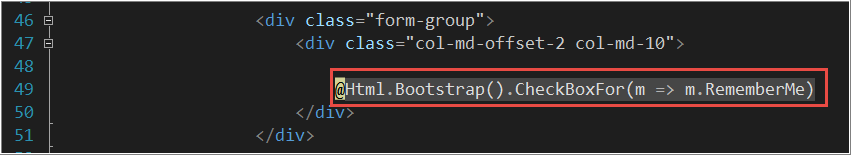
#### CRUD links



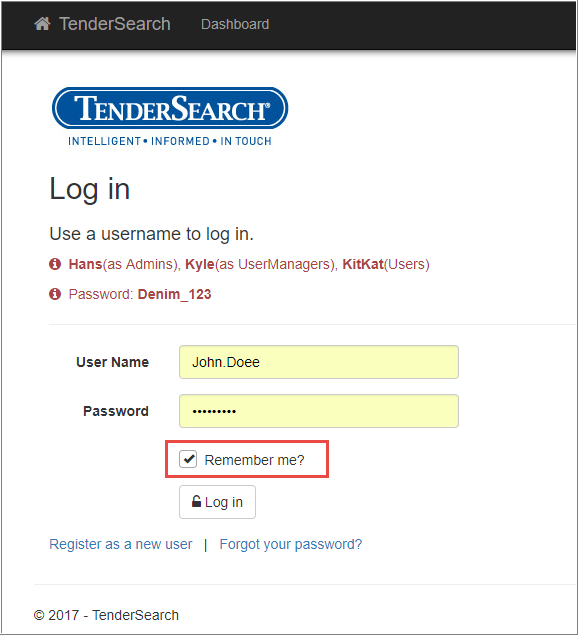
### HTML Helpers

I created this because the checkbox input provided by Microsoft is separated from the label.

Located in:

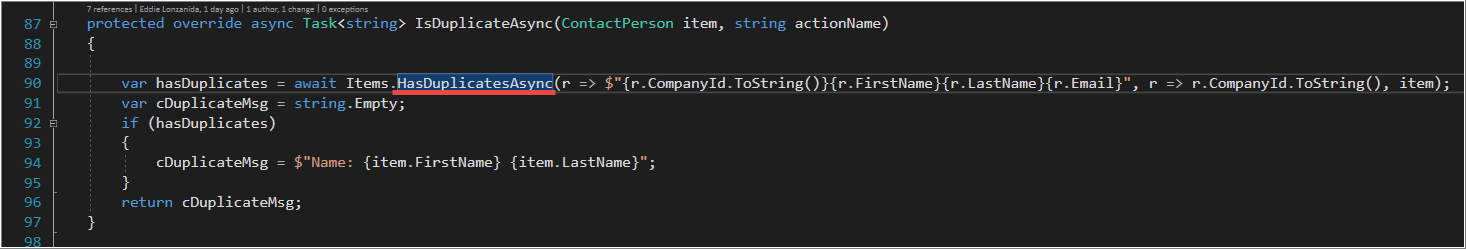
Sample Implementation in Login.cshtml:

Sample output:



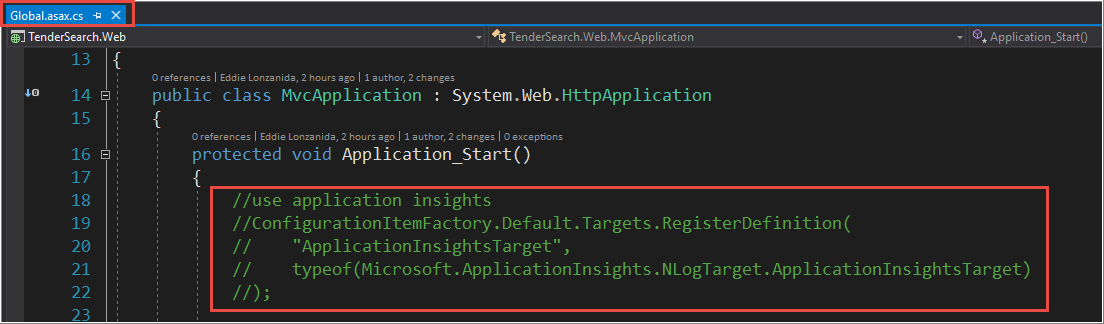
### Extension Methods

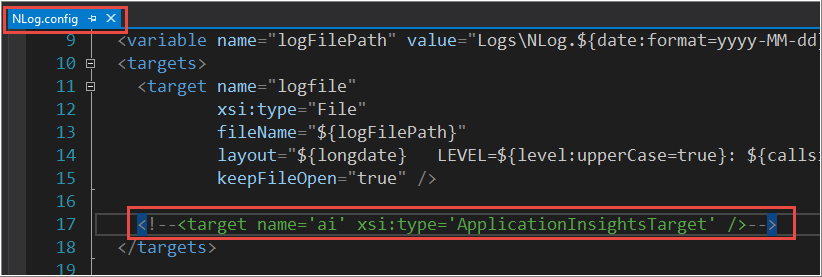
Extensions in this project is all over the place. It is an indispensable tool for me.

Below is an example with generics:

### Logging – NLog and Application Insights

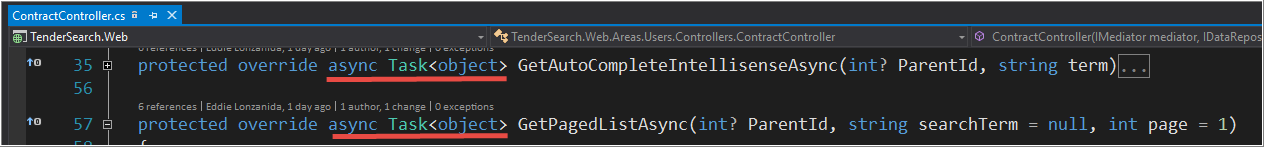
This application is equipped with logging mechanisms. I’ve used NLog. Feel free to uncomment the application insight configuration if needed.

Global.asax.cs

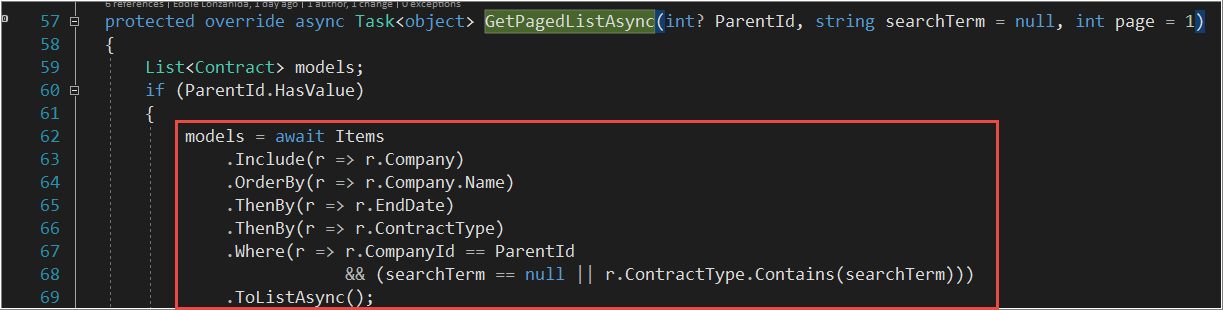
NLog.config

### Async/Await

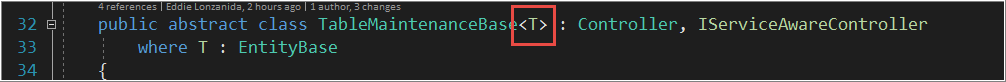
This is a replacement for “background worker”. Used for multi threading. GetAutoCompleteIntellisenseAsync and GetPagedListAsync are using async/await pattern.



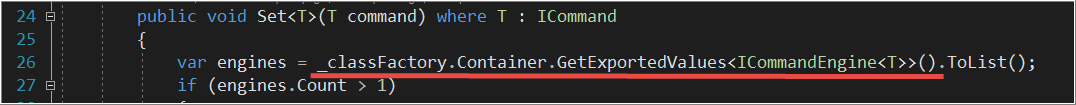
### LINQ via Fluent Api

LINQ made easy via EF6 Fluent Api. ☺

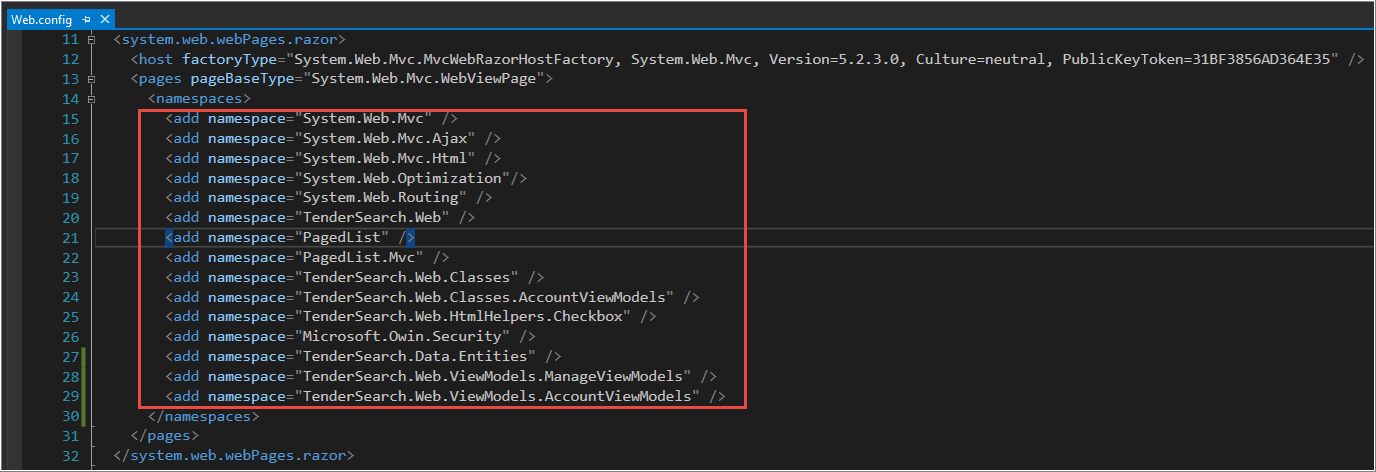
### Generics

This one of my favorite features in C# and .Net

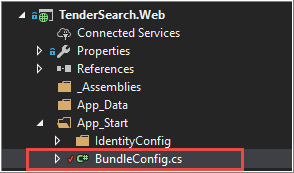
### Abstract factory – ClassFactory

Allows retrieval of concrete classes via generics.

### Razor default namespaces

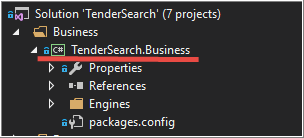
Just so I don’t have to type-in long namespaces in my razor views.

### Bundling and Minification

For faster page loads.

# Business

The purpose of this separate project is to capture the ‘business use cases’ into one location for reusability and maintainability. This is where most of the business logic is found.

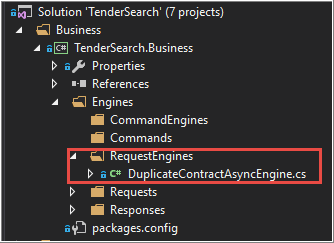


## Engines

Engines are the code representation of one specific business use case or functionality. This is used by the [Mediator](https://github.com/EddLonzanida/Eml.Mediator.Demo). It provides a clearer way of implementing a functionality through its ‘single responsibility’ characteristic.

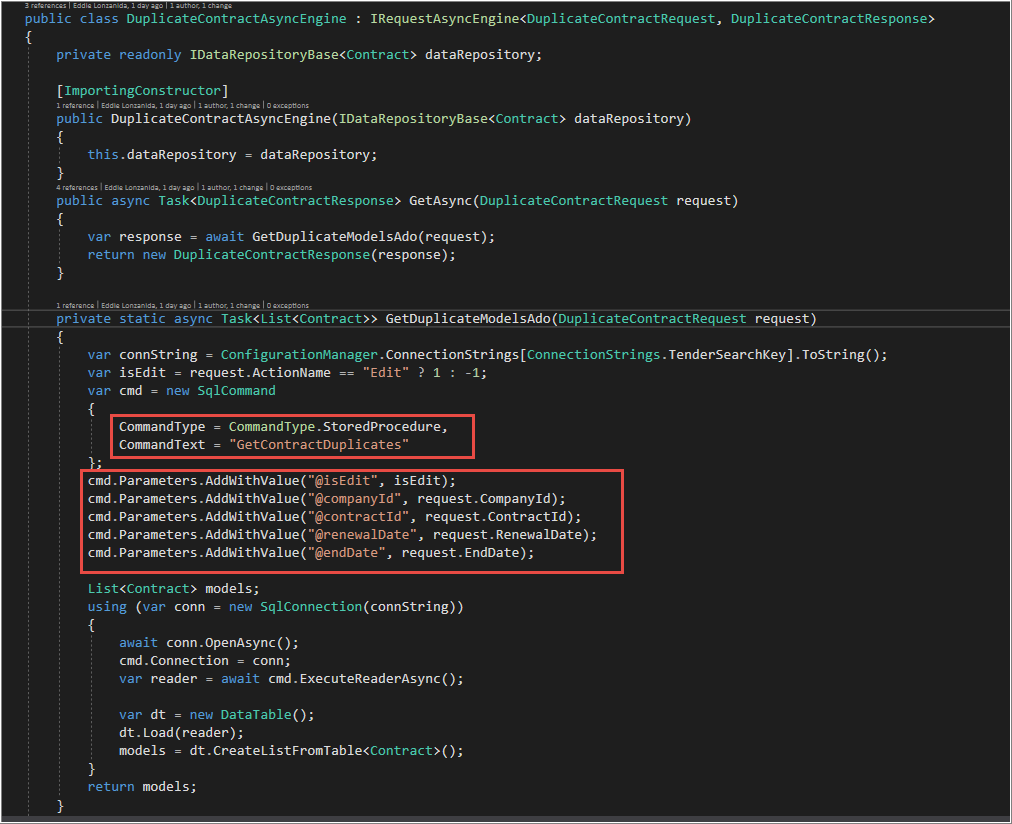
### Request Engines

Use case that performs a functionality then returns a Reponse type.



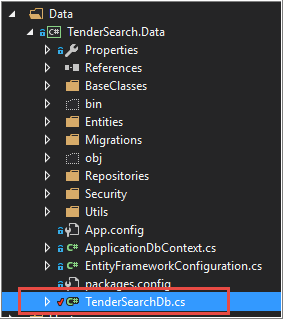
# Data

## Ado.Net

For the purpose of demonstration, I created a class that checks for duplicate contracts via Stored Procedure.

## EF6 – Code First

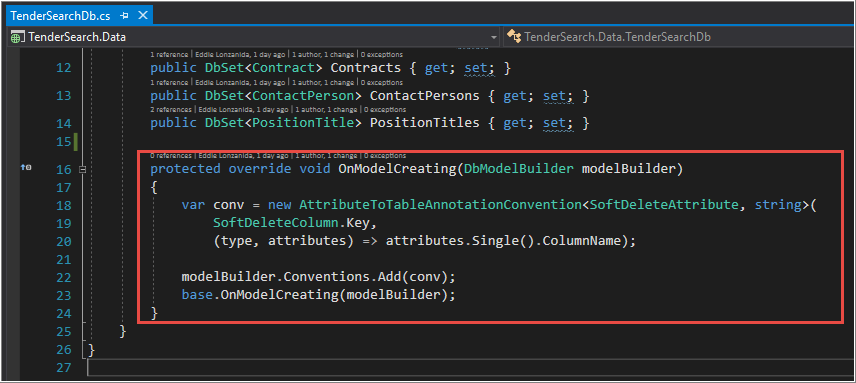
This project uses EF6 as its main ORM. The DbContext is located in:

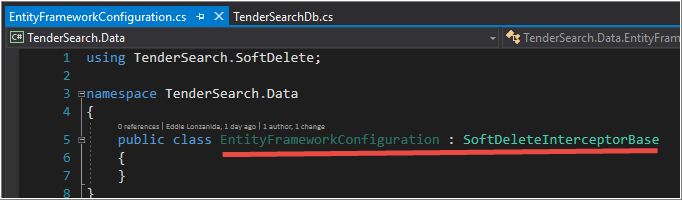


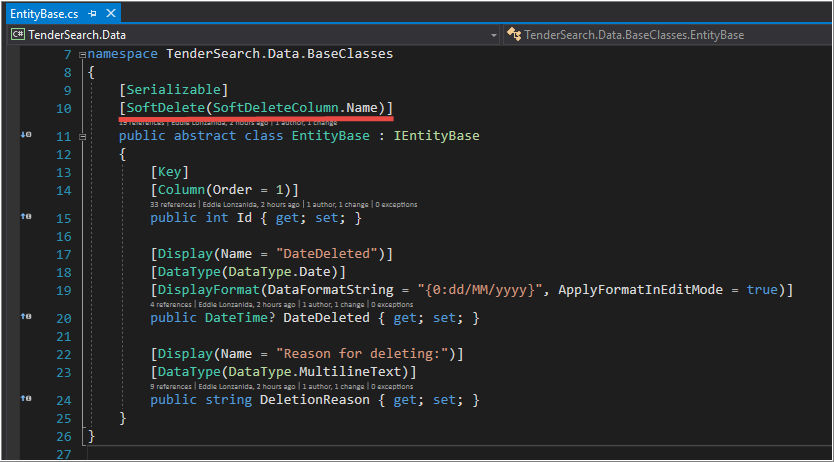
## Soft Delete (optional)

Some countries have laws that prohibit certain records to be deleted right away.

From the developer’s perspective, it’s better to be safe than sorry. ☺

Soft delete is configured in three places:



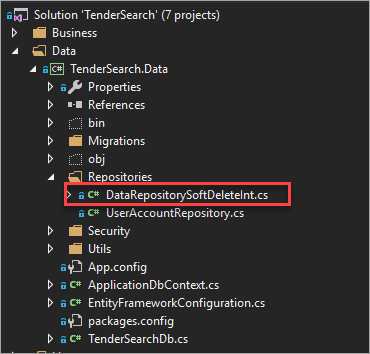


## [DataRepositories](https://www.nuget.org/packages/Eml.DataRepository/)

### DataRepositorySoftDeleteInt

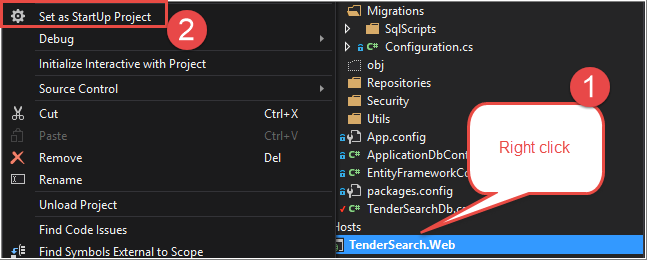
Discoverable via IoC/DI using MEF. Provides functionalities specific for data related tasks.

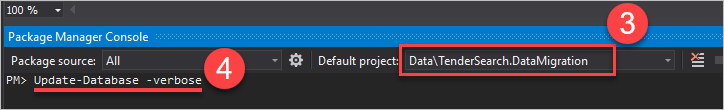
Located in:



## Automatic data migrations

This Entity Framework feature provides automatic creation of database, tables(including sample data) and stored procedures. I used this to facilitate the creation of db environment in a breeze.

How to use:



# Code repository

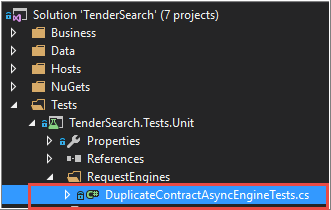
TFS Git/GitHub

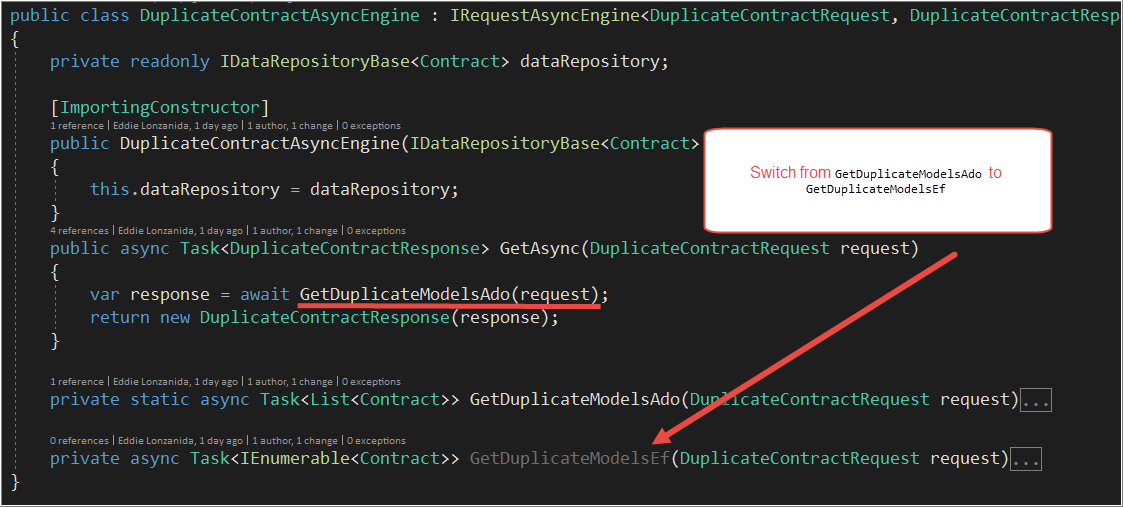
# Tests

Developing and deploying software products is just one side of the story. Maintaining an existing product is the other one. I prefer to include automated testing in all of my projects to maintain consistency and prevent regression bugs.

## Unit Tests

Below is a sample test I created:



Note: the test is intended to fail because it is using Ado.Net. Switch to EF6 before running.

## Integration Tests

In a bare minimum, below are the tests to ensure everything is discoverable and working fine. To keep the production and integration invironment in-sync, integration test re-uses the same production migration located in TenderSearch.DataMigration project but with different connection string (using local db). See App.config.

