Table of Contents

[Installation guide 4](#_Toc492873744)

[Build the back-end solution 4](#_Toc492873745)

[Built the front-end solution 5](#_Toc492873746)

[Front-end for Task 1 – Hotel establishments 6](#_Toc492873747)

[Home Page 6](#_Toc492873748)

[Search Filter Page 7](#_Toc492873749)

[Mobile View 8](#_Toc492873750)

[Sample for iPhone6 8](#_Toc492873751)

[Angular Artifacts 9](#_Toc492873752)

[UI for Task 2 – Flight segments 10](#_Toc492873753)

[Backend Host 11](#_Toc492873754)

[Business 12](#_Toc492873755)

[Engines 12](#_Toc492873756)

[\*Command Engines 12](#_Toc492873757)

[Request Engines 12](#_Toc492873758)

[\*Managers 12](#_Toc492873759)

[Data 13](#_Toc492873760)

[EF6 13](#_Toc492873761)

[Soft Delete 13](#_Toc492873762)

[DataRepositories 14](#_Toc492873763)

[NuGets 15](#_Toc492873764)

[Contracts 15](#_Toc492873765)

[Mediator 15](#_Toc492873766)

[SoftDelete 15](#_Toc492873767)

[\*PipelineFramework 15](#_Toc492873768)

[\*Code repository 15](#_Toc492873769)

[\*Continuous Integration and Delivery 15](#_Toc492873770)

[Tests 16](#_Toc492873771)

[Unit Tests 16](#_Toc492873772)

[\*Acceptance Tests - Selenium 16](#_Toc492873773)

[\*Integration Tests 16](#_Toc492873774)

[Strategies Used 17](#_Toc492873775)

[BaseClasses 17](#_Toc492873776)

[ActionFilterAttribute – ExplicitDisposeAttribute and IDisposeAwareController 18](#_Toc492873777)

[IoC/DI – MEF2 19](#_Toc492873778)

[Bootstrap 19](#_Toc492873779)

[FontAwesome 19](#_Toc492873780)

[Aurelia Single Page application – this the counter part of Angular 20](#_Toc492873781)

[Extension Methods 20](#_Toc492873782)

[Logging – NLog and Application Insights 20](#_Toc492873783)

[Async/Await 20](#_Toc492873784)

[LINQ via Fluent Api 20](#_Toc492873785)

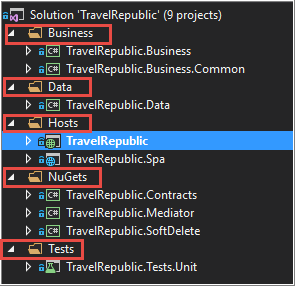
[Generics 20](#_Toc492873786)

[Abstract factory – ClassFactory 21](#_Toc492873787)

[Notes 21](#_Toc492873788)

Solution Architecture Overview

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



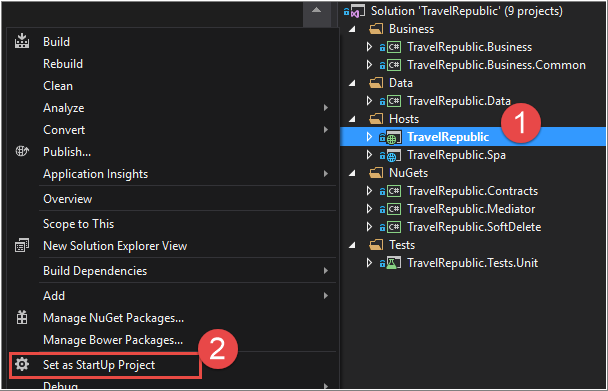
The solution is built using Visual Studio 2017

– with .Net 4.6.2 and .NetCore 1.1 (*for TravelRepublic.Spa only*)

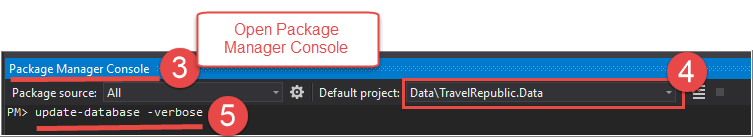
# Installation guide

## Build the back-end solution

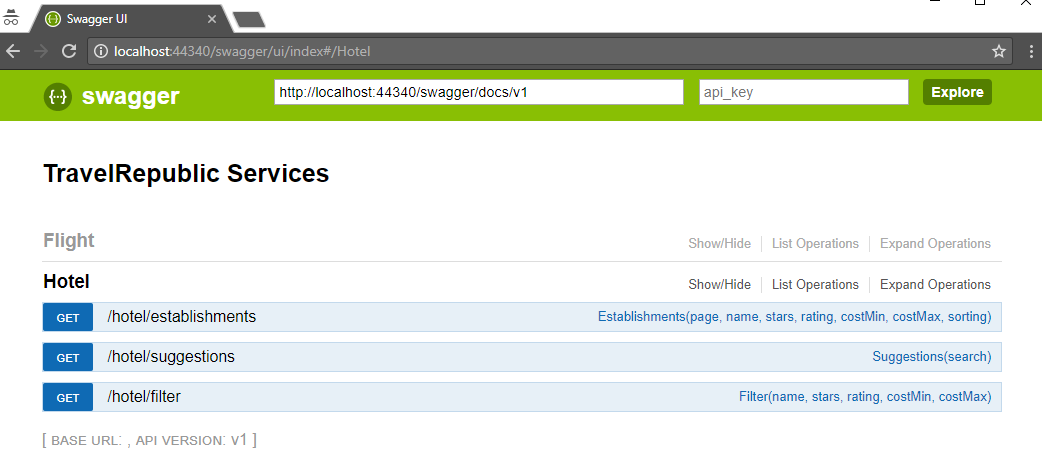
Set start-up Project



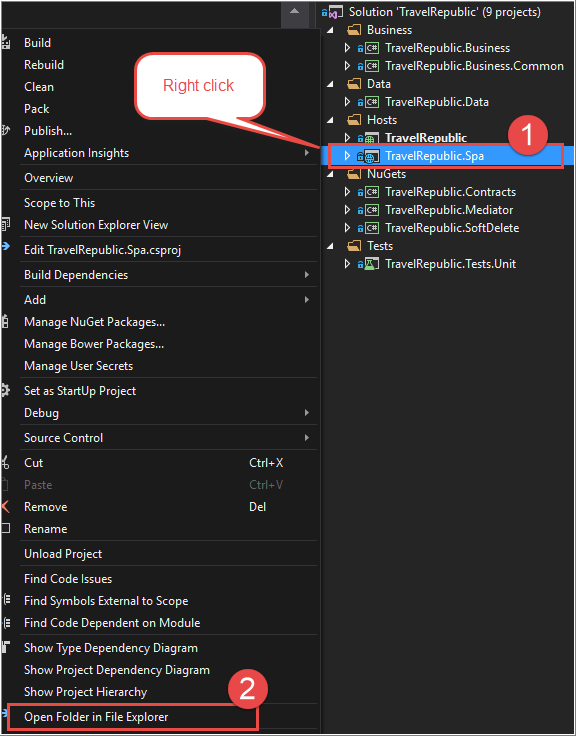
Seed the database via Entity Framework.

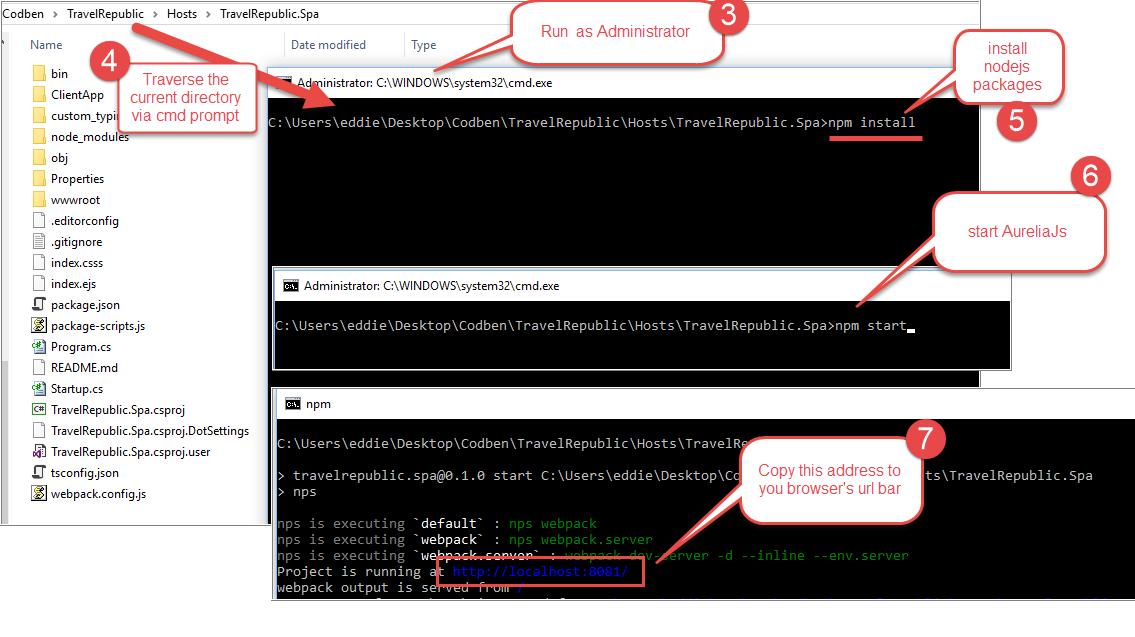


Press F5 to view the WebApi documentation via swagger



## Built the front-end solution





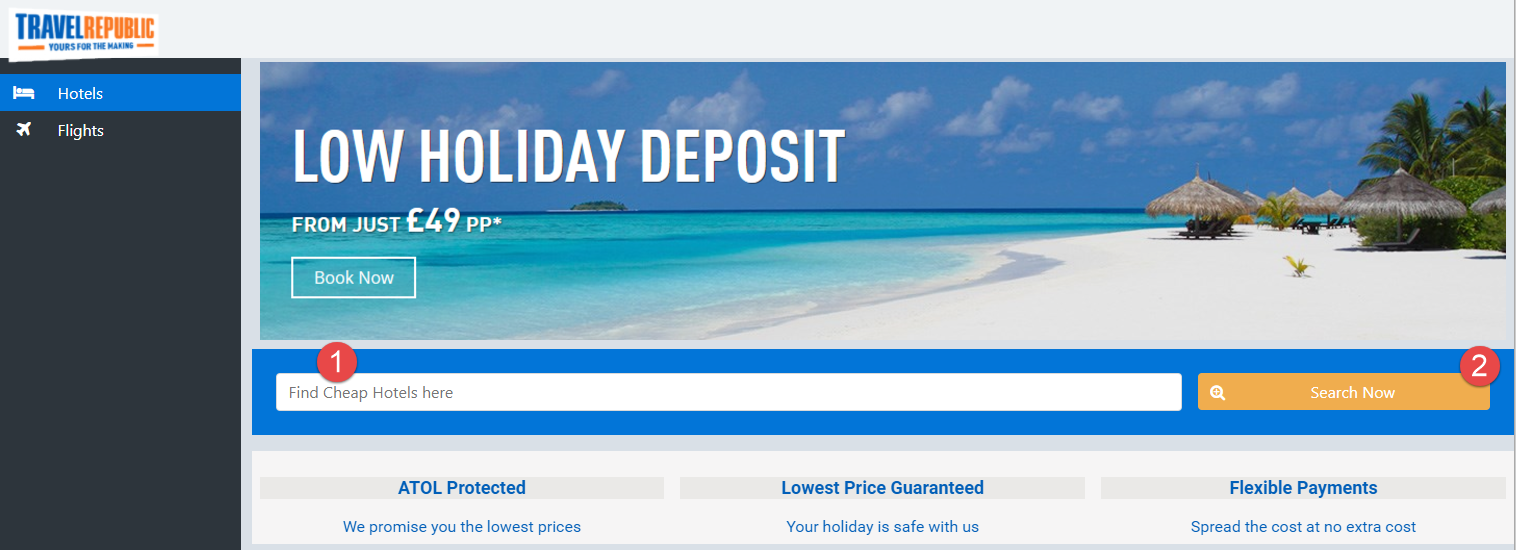
Note in step 6: Start Angular

Note: Keep the WebApi solution running while browsing the front-end

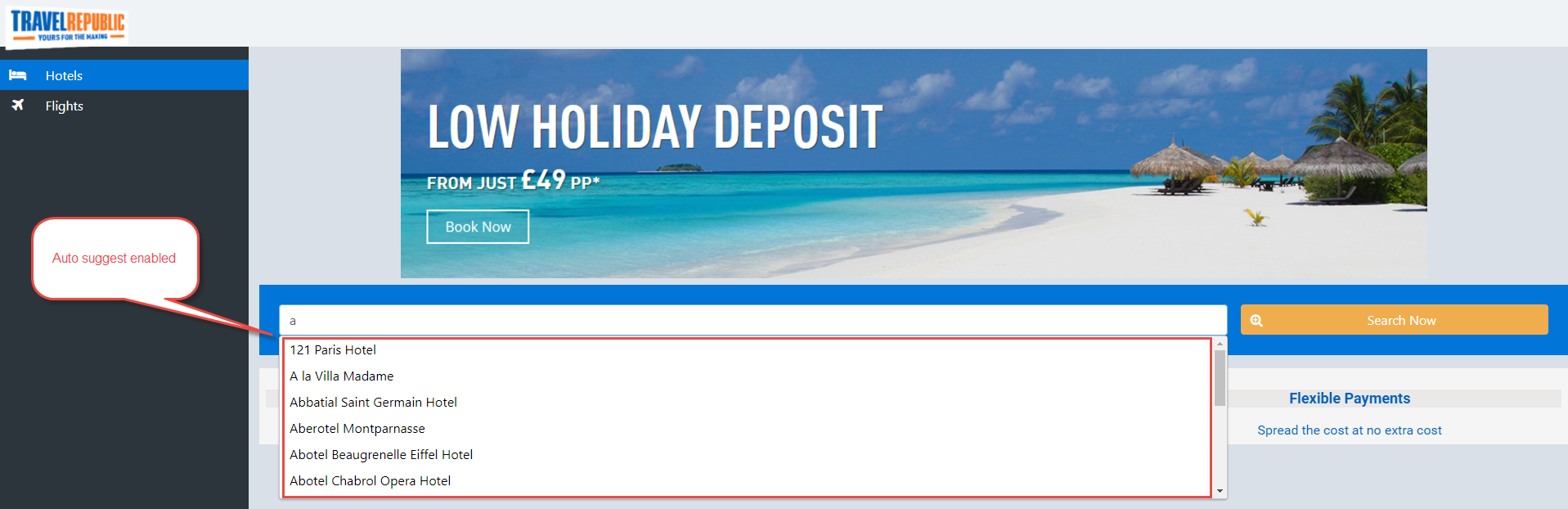
# Front-end for Task 1 – Hotel establishments

## Home Page

Enter search keywords then hit search.

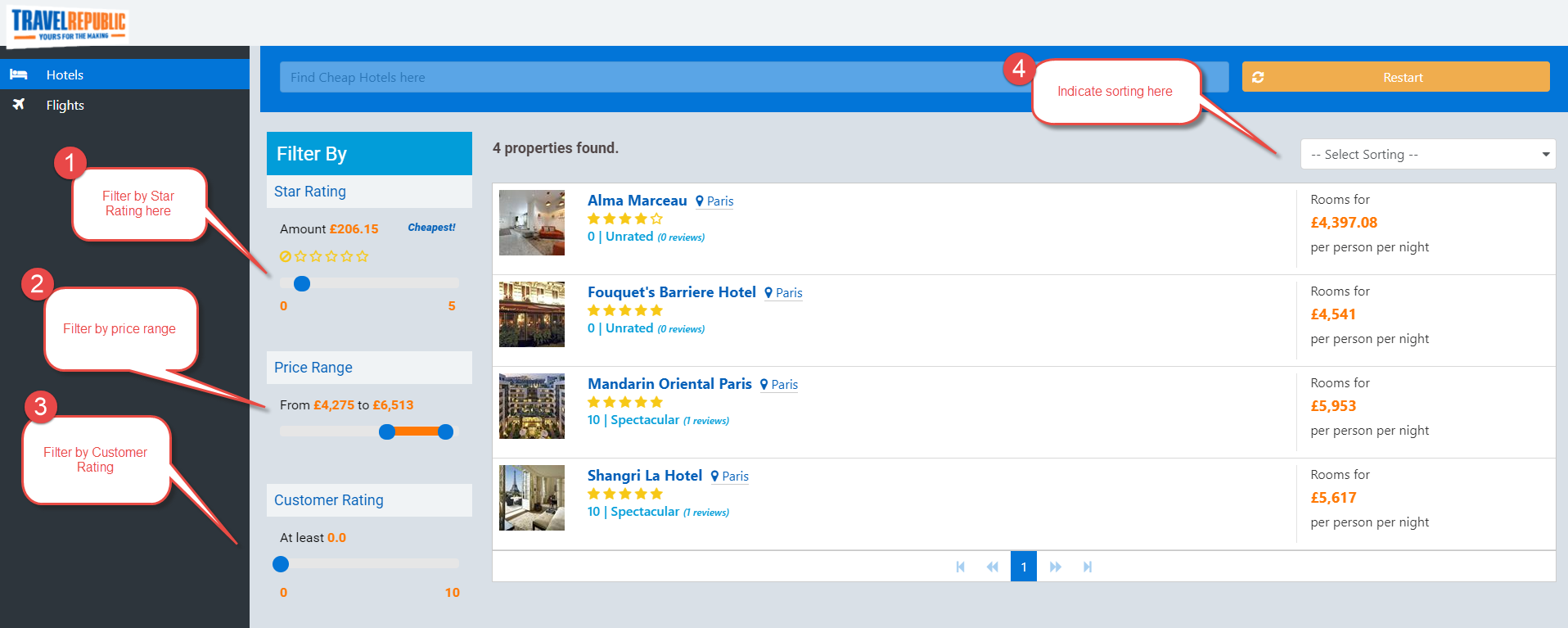


Or select from the suggested Hotels

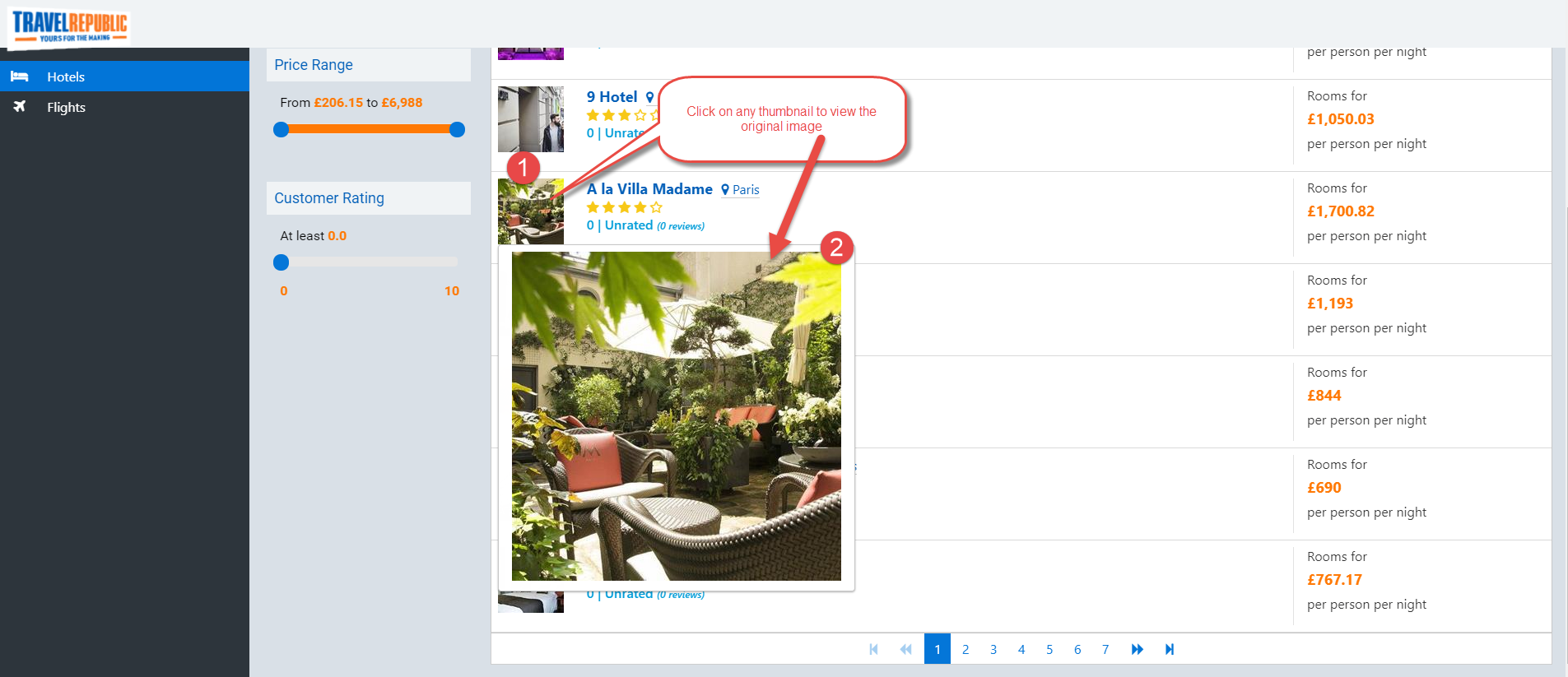


## Search Filter Page

After the search result, it can be sub-filtered in four ways:



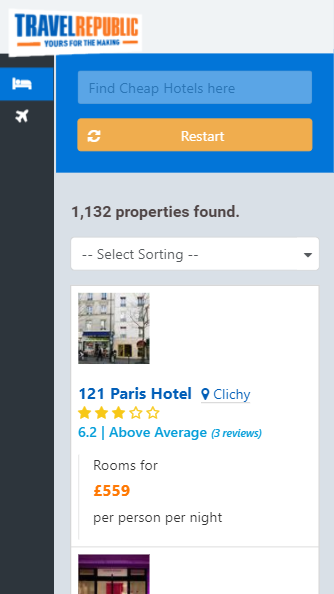
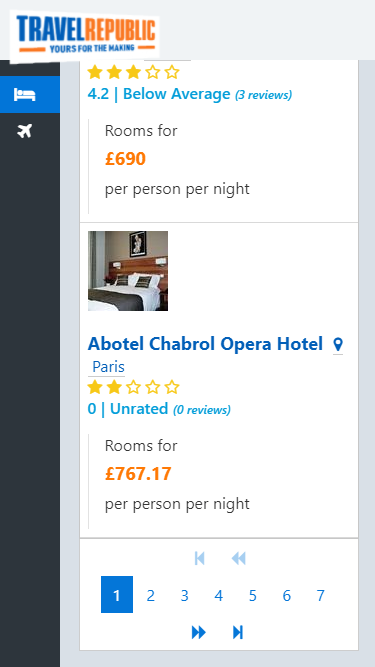
View a bigger picture by clicking on the thumbnail.



## Mobile View

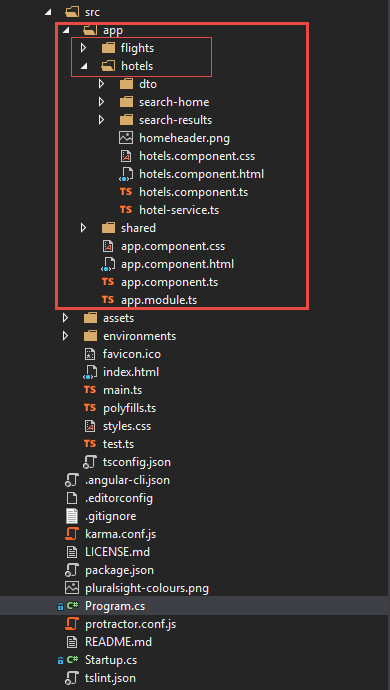
This sample page is designed for mobile browsing as well.

### Sample for iPhone6

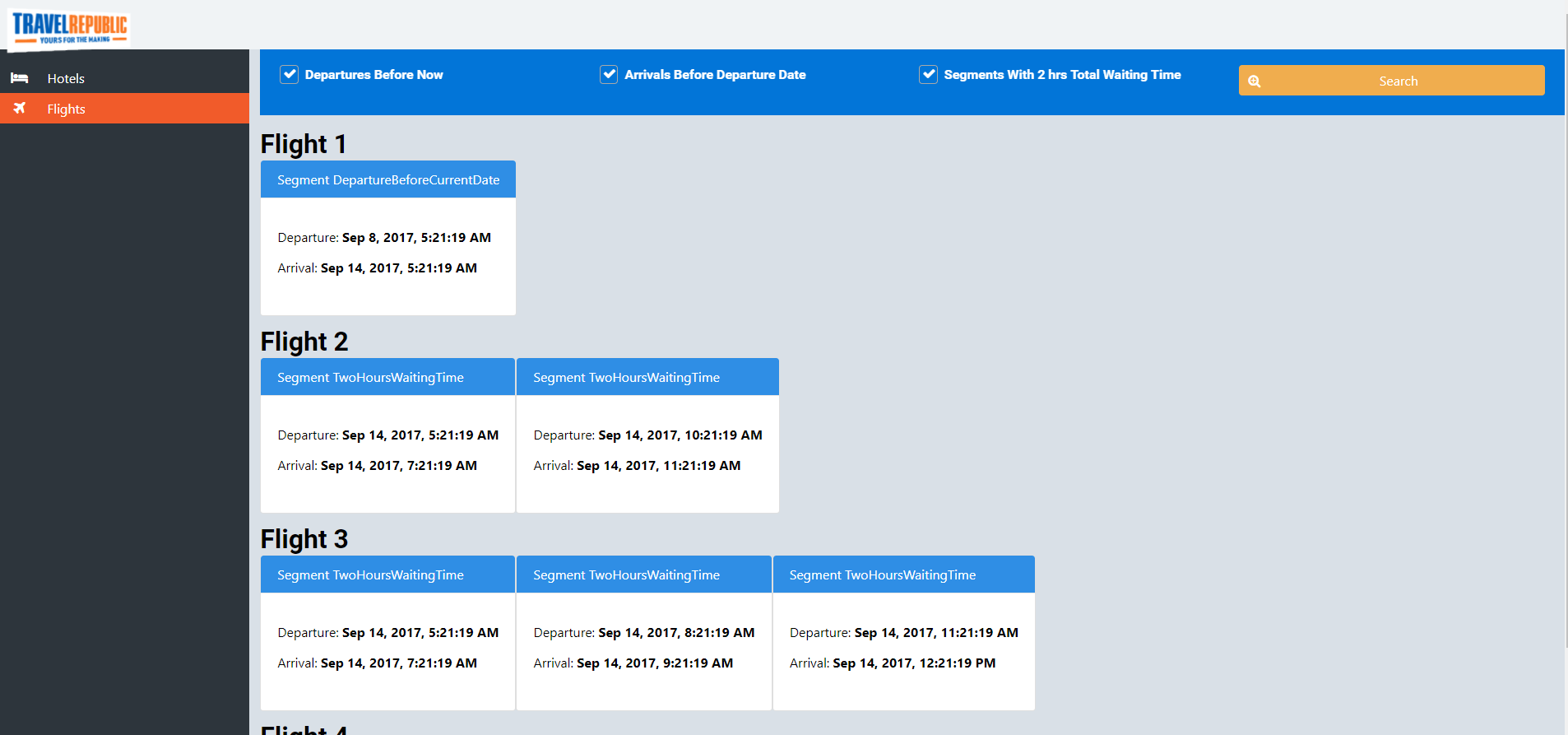


## Angular Artifacts

Hotels and Flight artifacts are located under the app folder as shown below:

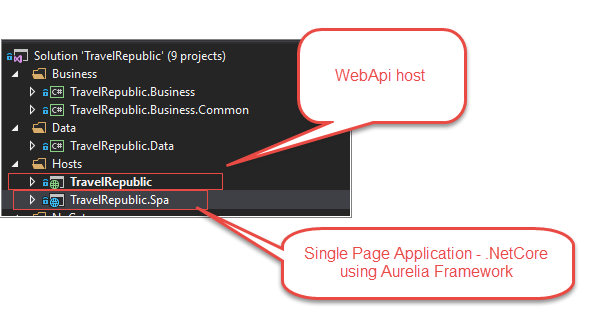


# UI for Task 2 – Flight segments

For easier checking if my truth table is correct. 

# Backend Host

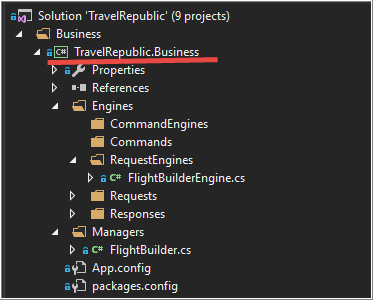
Contains the WebApi projects



Note: It has changed from Aurelia to **Angular**

# 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

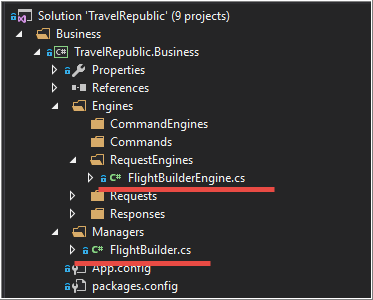
Engines are the code representation of one specific business use case or functionality. This is used by the Mediator. It provides a clearer way of implementing a functionality through its ‘single responsibility’ characteristic.

### \*Command Engines

Use case that only performs a functionality without returning anything.

### Request Engines

Use case that performs a functionality then returns a reponse.



## \*Managers

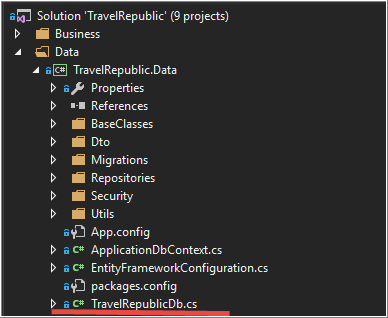
This is the code representation of all Uses cases. One large chunk of class that implements related sets of functionality. Ex: Reservations or Inventories.

# Data

## EF6

This project uses EF6 as its main ORM. Handles automatic data migration, creation of database, tables(including sample data). I used this to facilitate the creation of environment in a breeze. (see instructions on how to build the solution)

The dbcontext is located in:

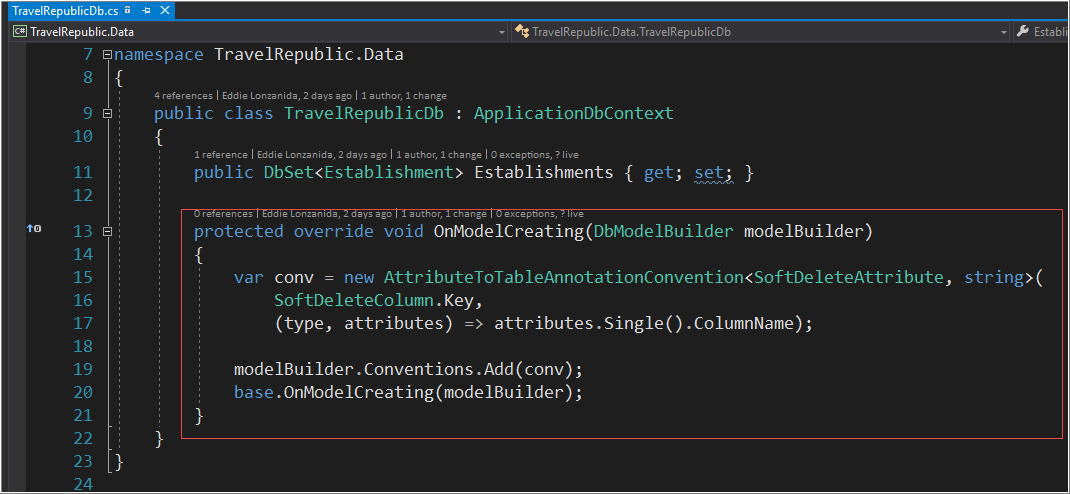


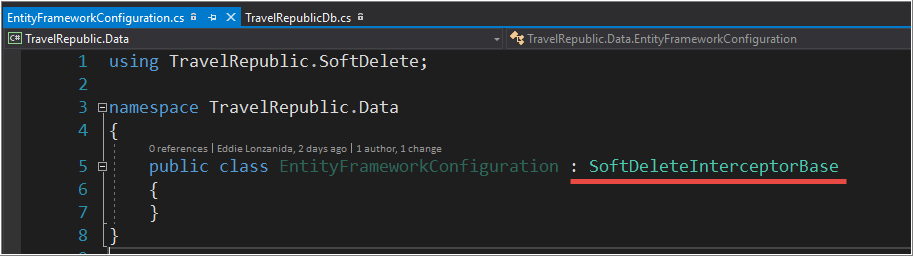
## Soft Delete

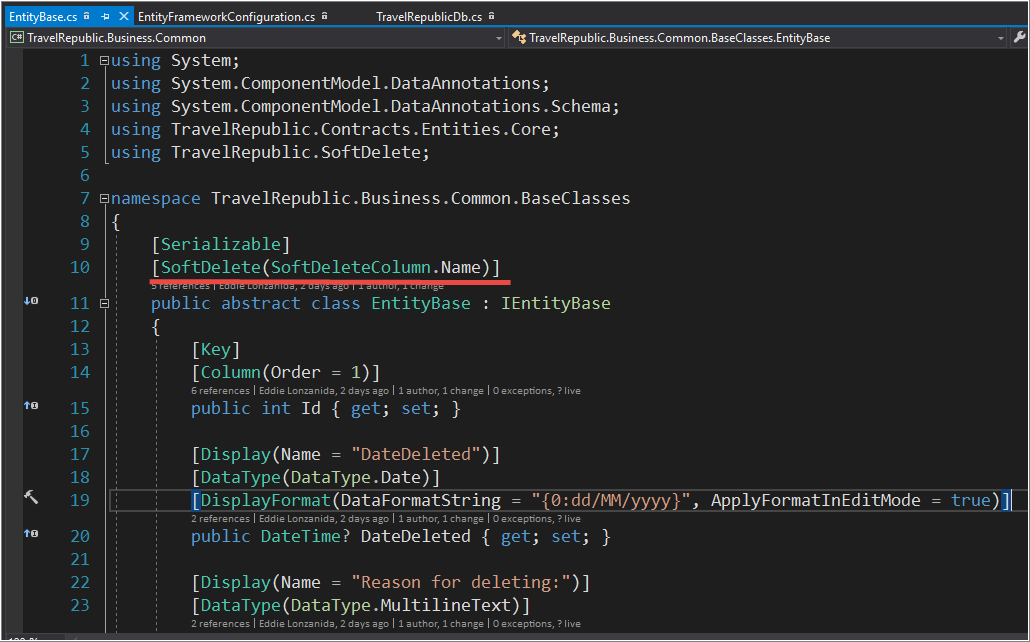
Some countries have laws that prohibit certain records to be deleted right away.

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

Soft delete is configured in three places:







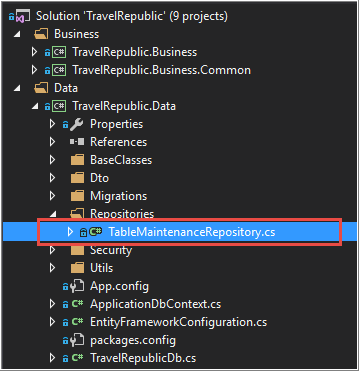
## 

## DataRepositories

TableMaintenanceRepository

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

Located in:



# NuGets

NuGet is my choice of distributing packages across the project and across developers. Below are the components that can be NuGet-ized to minimize project clutters.

## Contracts

## Mediator

## SoftDelete

## \*PipelineFramework

# \*Code repository

TFS Git/GitHub

# \*Continuous Integration and Delivery

TFS Online

Build Definitions

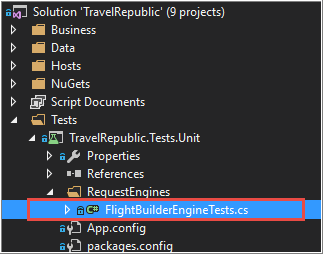
Build Agent configurations

# 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:

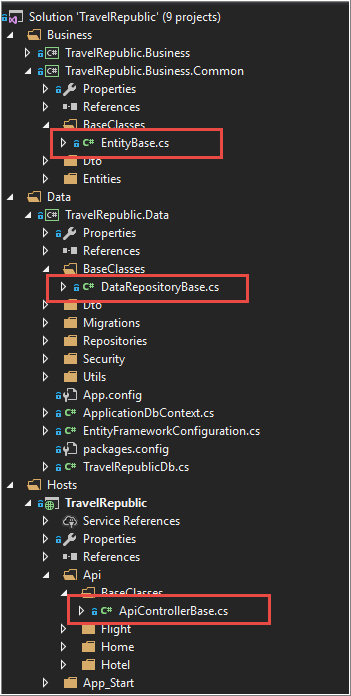


## \*Acceptance Tests - Selenium

## \*Integration Tests

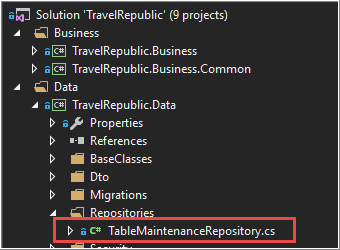
# Strategies Used

## BaseClasses

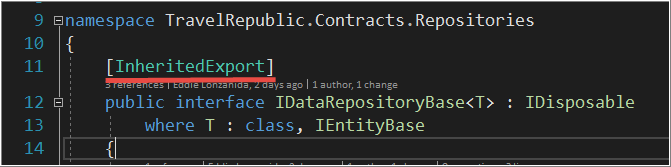


#### DataRepositoryBase

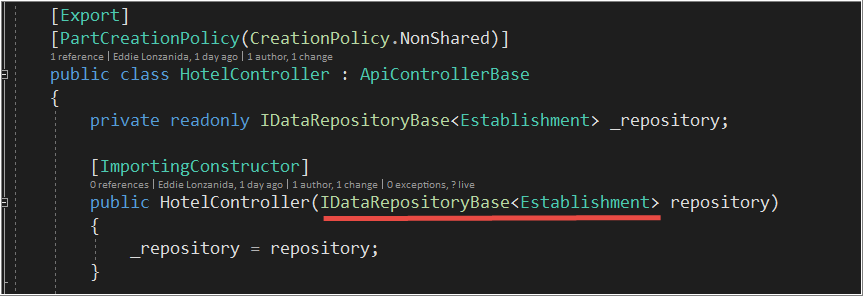
This demonstrate the use of DataRepositories such as



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’

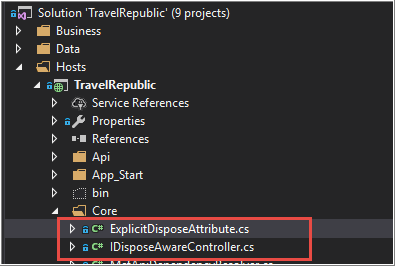


#### EntityBase

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

## ActionFilterAttribute – ExplicitDisposeAttribute and IDisposeAwareController

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



## IoC/DI – MEF2

#### Mediator

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.

## Bootstrap

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 ☺

## Angular Single Page application – this the counter part of Angular

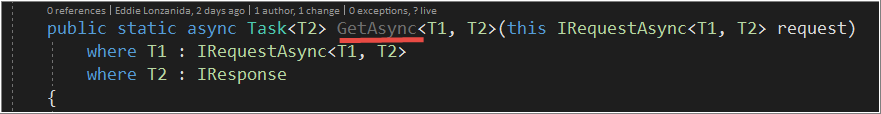
Uses WebPack 3 for bundling and minifications, etc.

Uses ES6

## Extension Methods

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

Below is an example with generics:



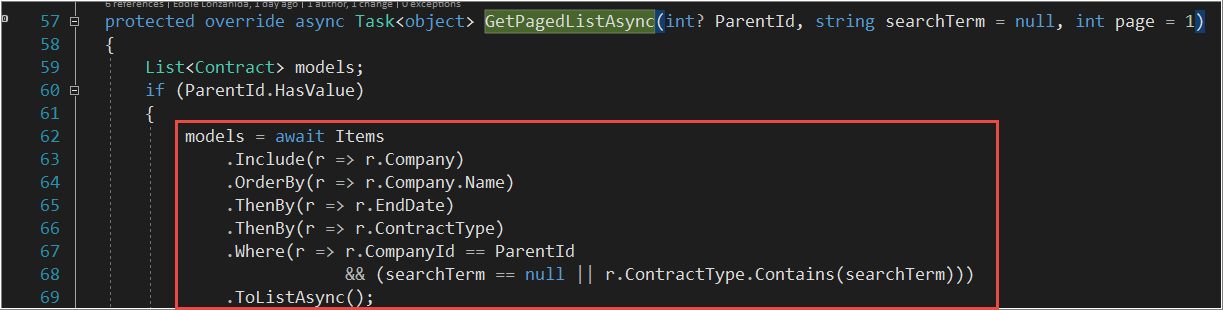
## Logging – NLog and Application Insights

This application is equipped with logging mechanisms. I’ve used NLog and Application Insights.

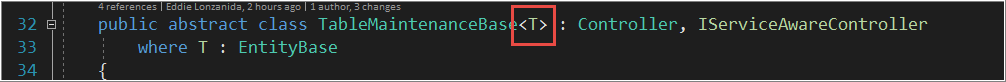
## Async/Await

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

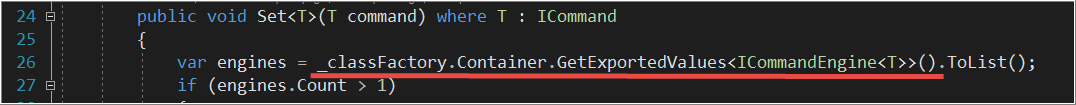
## LINQ via Fluent Api

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.

# Notes

Items with asterisks (\*) are for showcase purposes only. These are not implemented in this project. It is intended for future use if the project becomes more complex.