## Brief introduction

To refactor it, I build 7 projects including 2 Test projects to make it follow Web project structure in my mind. I will introduce them one by one.

## Development project.

1. ProductData

This project is the data entity project.

First I add *foreign key* in database to connect products and options.

Then I use *Entity frame work 6 database first* to generate Product context and entities. There are 2 entities: Product and ProductOption.

1. ProductDAL

This project is Data Access Layer project. It contains the Repository and Unit of Work.

First I create Interface *IRepository* by Generic and implement it depending on Product context.

Then I create Interface *IProductUnitOfWork* which contains *2 IRepository* Interface which are Product Repository and ProductOption Repository.

1. ProductCore

This is the project contains some general logic or data structures for our solution. Currently only has mapper logic.

First I use the “*AutoMapper*” library as the foundation of the mapper logic.

Then I create the interface “*IModelMappper*” to wrap the mapper library to generic logic. And I implement it by AutoMapper apis.

After this, I create the interface “*IEntityModelMapper*” to meet the requirement of our general entity and model mapper to each other logic and implement by the “*IModelMappper”.*

Finally we can use the mapper class “*EntityModelMapper*” in our solution.

1. ProductService

As I believe, the WebApi level is only for the http request handle functionality. All the business logic should be encapsulate in Services and inject them to controllers.

First, I create the interface “IProductService” and “IProductOptionService” which contains the logic of product and product options.

Then I implement all the logic depending on the Mapper, Unitofwork.

Meanwhile, I add model in Service level too. Add all the model classes with model valid attributes.

1. refactor-me

This is the WebApi project, I did not change the name.

First I refactor controllers to add calling service logic with service interrface.

Then I use the “*Autofac*” to build Dependency Injection to inject Services.

Then I add a filter attribute ” ModelVaildAttribute” inherit from “ActionFilterAttribute” to make model validation , if model is not valid, we will return bad request.

Finally add some view model to meet the requirement the endpoint.

## Unit Test Project

Two unit test projects are created to test the *controllers* and *services* respectively.

1. ServiceTest

I use “Moq” library to mock the Unitofwork and Repository. Then test all the service logic.

A list acts as the database, all the logic is verified by the list

1. ControllerTest

I use “Moq” library to mock the Services. Then test all the controller logic.