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 *foreign key* is added 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.

2. ProductDAL

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

   First Interface *IRepository<Entitiy, Key>* is created by Generic and implemented depending on Product context.

   Then Interface *IProductUnitOfWork* is created which contains *2 IRepository* Interface which are Product Repository and ProductOption Repository.

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

   First the "*AutoMapper*" library is imported 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".* This interface will be used in other project.

4. ProductService

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

   First, depending on requirement, the interface “IProductService” and “IProductOptionService” are created which contains the logic of product and product options. Both of them will be injected to WebApi level.

   Then all the logic is implemented depending on the Mapper, Unitofwork.

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

5. refactor-me

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

   First controllers are refactored from 1 to 2 which are product and production option controllers. Together with that, the Route rule is refactored too.

   After this, the business logic is removed and service interface calling replace them.

   Then “*Autofac*” is used to build Dependency Injection to inject Services.

   Following that, 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
   “Moq” library is used to mock the Unitofwork and Repository. Then test all the service logic. Some lists perform as the database to test all the “CRUD” operation.

2. APIControllerTest

   “Moq” library is used to mock the Services. Then test all the controller logic. This is easier, we just check the mock services’ methods are run.

TODO:

1. Exception handler and Log: Currently, if some error happens, such as insert same key product, the exception message will be returned directly,
2. Authorization: this is not required in this project, but it should be an important part of WebAPi.
3. Dependency injection on Service, and UnitOfWork, currently I just add it on controller.
4. Some extreme situation on Unit test, currently I just add basic cases.

   If there is enough time, I should do simple implementation on them. But I am so sorry that I have some other schedule decided earlier at the weekend. ☺