Develop Enterprise based Asp.Net MVC Web Application

**Introduction**

I like to develop a demo asp.net MVC web application to enable online sales of chinese tea. I assumed that there are three basic tea catalog and each tea should has its details in database. The business logic extract two classes as below

public class Category {

public int CategoryID { get; set; }

public string Name { get; set; }

public DateTime? DateCreated { get; set; }

public DateTime? DateUpdated { get; set; }

public virtual List<Tea> Teas { get; set; }

}

public class Tea

{

public int ID { get; set; }

public string Name { get; set; }

public string Description { get; set; }

public decimal Price { get; set; }

public string Image { get; set; }

public int CategoryID { get; set; }

public Category Category { get; set; }

}

Web application I will develop should be able

* to create, delete, edit, and list the new tea details in web pages, therefroe, we need to create CRUD actions in MVC controllers.
* In order to meet the requirement of CRUD from controllers, services that controllers will call should reflect those CRUD data manipulation in database.
* Services will use reposiitory design pattern to connect to database, database will be created by code first entity framework. A seed of data for tables created in code first database are created in database for testing. Code first will allow us to easily update our database from nuget command consle.
* Respostory design pattern will allow us to update the data access funationality easily and quickly without damaging the structure.
* Dependency injection is necessary to inject microservices into MVC controllers.
* AutoMapper is required to allow you to map the viewmodel and data model in a tandard way.
* Error handing is enabled in aplication level
* and ILogger log system ise used to log the work flow or data in log system.

This web applicaiton can be scaled up easily without offensing the SOLID principle

**Get Started**

* open VS 2017 to create a new asp.net MVC web application as below



* add a new folder and named it as "Models", and then add two entity classes Category and Tea as below



those two classes will be used to create two tables called Teas, and Categories in Database.

**Code First Database**

* Code first to create a new database called JayThom, at first, add a new connection string in web.config file as below



* Press F5 to run web applicaiton and click register link in the index page as below, this will create a new database in SQL server automatically see image shown below





Register action in MVC controller will create a default security related tables in database automatically, this will enable us to quickly create a new account and login to th secured web apges in a default way. From tables we can see in this database, we know we have not created business related tables Teas and Categories yet.

* Now it is our time to create those two code first tables in database, at first, we need to add a new connection string in web.config as below



Defaul connection string in web config is used to create default security reated tables in database, this new connection string is used to create custom tables in database.

* Create two code first configuration files in Models folder as below



* Create a DBContext class called TeaEntities under the root folder as below, this is used by Nuget command console for database updating



* Open Nuget command console, run Enable-Migrations -contexttypeName TeaEntities, this will create a configuration file for code first





* Update seed method as the code below and run the update-database command again to nsert the example data into Teas and Categories two tables in database.



* Database is ready for access, we can close Nuget command console window

**Data Access - Repository**

Repository design pattern is used to create a data access layer for middleware web services that can access to database in backend. Here, we create a gateway to send the request and receive the response from backend.

* Add a new folder called Infrastructure to the root of the project, create some generic repository classes into it , those calsses is the interfaces to talk to tables in database via DbContext object, the classes are listed below



this generic classes will be used by different entity classes for CRUD manipulation in the correspondent tables in database. The implementation will be created in different child classes based on the SOLID "L" principle.



this repositorybase lass will init the DbContext dta object in the object and inherited by different entity classes for its own data CRUD operations.



This UnitofWork class simply does the ATOM tranasaction by calling DbContext.Commit() method to manipulate the data Read/Write/commit transaction in a secure level.

After those three basic repository classes are created, We can use those factory classes to create different child classes based on different entity classes for their own CRUD operations

* Add a new folder called Repositories and create two child repository classes as below





Those repository classes does not mplement any methods to do CRUD operations, they should be called by the business and service middleware classes to connect to database. service classes here implement the businee logic and organize the request and handle the response

**Services for Business logic and data**

Our data access layer gateway now is ready for us to call. Front end needs to use service classes here to communicate with data access layer. Service classes would be the class library, web services, or web APIs etc.

* Add a folder called Servces in the root folder of the project and then create some services classes into it.





we inject child repository interfaces and classes into this service class via Dependency Injection technique. We use Autofac DI container here. open Nuget package manager and add Autofac package into project.



We inject repository into data service classes here via autofac DI container. Now the service classes are ready for us to go to be caled by the front end.

**Front end - Asp.net MVC controllers**

Asp.Net MVC controller now is excited to see the data service interfaces that can be injected into controller as the HomeController codes shown below



View Models normally is differnet from entity models in asp.net MVC web application, when we get response data from backend, data structure normally is the entity model class based. Razor view uses viewmodel class to present the data , therefore, we need a mapping between vewmodel and entity model classes. Manually maping is a tedious task, lckily we can have a AutoMapper package we can get from Nuget to work for us. See one example code about Mapper below



**Front end - Razor Views**

We created TeaList, Edit, Details, and Delete views to indicate the data CRUD manipulaton via Asp.Net MVc Razor views. The reults are snapshotted below









**Deployment**

After web application is developed, we need o publish a staging web server for testing, We can use publish feaure in visual studio to publish , see example image below



click publish in menu, we then create a profile to publish release version of code into a folder called publishoutput.

from IIS web server, we create a virtual directory point to this publishoutput folder and convert the directory into web application, update sql server connection string in web.config file, we now can run this web application in IIS web server such as the otput image below



Commercial publish operation is done by continuous integration software such as TeamCity, Octopus, and TFS, etc.

After Quality assurance and performance tests, we finally can push the web site in staging server to live production server.

**Summary**

This demo project is the simle version of one of the projects I have developed for the business, it is used in Australia government businesses.

Code first allows us to update database based on the change of the business requirement. For example, if we want to add Customer table to database, we create a Customer entity class and code first configuration file, then we can run update database command to create a new table in database.

Repository factory allows us to create child repository for different entity classes. For exmple, we have a new Customer entity class and have created customer table in database, now we can create customer repository class from factory class based on Solid principle.

Service classes can be created based on the child repository classes. for example, after customer child repository class is created, we can create a customer service class to deploy the CRUD operation from customer child repository class. Customer child repository class can be injected into service classes via DI container

Actions in MVC controllers now can consume service classes to send request and receive response from backend. for examaple, after customer service class is created, we can inject this class interface into MVC controller via Autofac DI container, MVC controller now can interact with backend via service classes

Based on customer MVC controller, we can create Razor views to reflect the presentation this MVC controller required and the user experience. we can create Customer view model to project data into browser. Data from service classes normally is entity class based. therefore, we need a Mapper to map the data from entity class to view model class before data is presented in web browser.

Error handling is application level, in this project, we simply redirect error operation to default view. log process class is developed to log the critical workflow into physical file such as txt file in server. for example, if we want to monitor the payment workflow from customer, we need to log all activities this customer has done in our system such as login, make order, make payment, print invoice, and leave time,etc.

This web application is the loose-couple based modern web application that can be used to build enterprise based and high traffic web site. Repository data access layer provides a reliable gateway for services. Service middle tier can be classes library, web apis, and other microservices so service middle tier can be scaled up easily. We can host the service components in IIS web server to separate it from database backend.

The default Asp.net MVC front end is Razor view, we can enable mobile responsive capability in Razor view. we also can use other front end technology to talk with MVC controllers via HTTP service or ajax call such as angualar js , knockout js, react js, and html 5 css 3, etc. Therefore, this enterprise based web application can be expanded with multiple front end technology for different business requirement.