**User Subscription Management API**

Notes:

1. WCF error handler is implemented using FaultContract for custom exceptions thrown by services and a global error hanlder is also implemented to be able to catch any unhandled exception in a global way
2. WEB API error hanlder is implemented as a HandleExceptionFilter to be able to catch any exception thrown by the WCF service in a global way
3. Autofac is used as IOC container for injecting dependent classes into constructors for the both the server side (WCF) and the client side (WEB API)

Architecture:

The application has 2 main components:

**Client Side**: A web API consuming a WCF service (Server side) for managing users and subscription data

The client side is a simple web API containing the following:

1. Controllers: User controller for handling all users’ operations and Subscription controller for handling all subscriptions’ operations (Details of all available operation will be described below)
2. Filters: HandleExceptionFilter class inheriting from ExceptionFilterAttribute to catch unhandled exceptions in Web API that can originate from any controller methods. This is a global way of handling exceptions in Web API instead of having to catch exception in every controller‘s method. In case the exception is specific to the controller’s action method, it should be handled inside the controller but as all of the current actions are catching exception thrown by WCF service (FaultException) so handling them in a global way can be suitable
3. **App\_Start:** contains a dependency injection (DIModule) module which is used to inject the WCF services’ contracts inside the controllers’ constructors using AutoFac

**Server Side**: WCF service accessing the database for reading and writing users and subscriptions data to and from the database

The service side is implemented using the **onion** **architecture**:

**For the Core :**

1. [UserSubscriptionsManagement.Domain](https://github.com/ranazamzam/UserSubscriptionsManagement/tree/master/UserSubscriptionsManagement.Domain) : This layer contains the Models representing the User, Subscritpion and UserSubscritpion models. These are the main models of our application. This layer contains also Interfaces (IRepostiroy, IUnitOfWork) to be implemented by the Infrastructure layer based on the database used. These interfaces should be used by the service layer for getting and reading data from and to the database without having any idea of how the database access is implemented or which ORM are we using.
2. [UserSubscriptionsManagement.Contracts](https://github.com/ranazamzam/UserSubscriptionsManagement/tree/master/UserSubscriptionsManagement.Contracts) : This layer is also a Core layer. It contains the WCf Data contracts and Service Contracts to be used by both the WCf service and the Web api.

**Note: Please note that I used shared contracts for the both the client and the server as they are currently contained in the same solution and for simplification purpose. If the client is not in the same solution, equivalent contracts can be implemented by the client in this case.**

**For the Business:**

1. UserSubscriptionsManagement.Services : this layer contains the services used by the WCF service, UserService and SubscriptionService which are making read and write calls to the database.

**Note: These services will be injected to the WCF service using AutoFac using the following code , which makes the WCF service itself separated from the business logic and makes the testing much more easier**

<%@ ServiceHost Language="C#" Debug="true"

Service="UserSubscriptionsManagement.Contracts.ServiceContracts.IUserService, UserSubscriptionsManagement.Contracts"

Factory="Autofac.Integration.Wcf.AutofacServiceHostFactory, Autofac.Integration.Wcf" %>

1. UserSubscriptionsManagement.Utility: this layer contains a global error hanlder for the WCF service to catch any unhandled exception and also to catch custom exceptions thrown by the services.

**For the Infrustrusture:**

1. UserSubscriptionsManagement.Infrastructure: this is the layer responsible for reading and writing to and from the database. Entity framework is used as an ORM mapping between the data and our models

Deployment Steps:

1. Change the connection string in the web.config of the UserSubscriptionsManagement.WCFService project to point to your database

<connectionStrings>

<add name="ConnectionString"

providerName="System.Data.SqlClient"

connectionString="Server=(localdb)\mssqllocaldb;Database=UserSubscriptionsManagement;Trusted\_Connection=True;"/>

</connectionStrings>

1. For the database, set UserSubscriptionsManagement.Infrastructure as a startup project then run update-database command
2. If there is any problem creating the database, create an empty database in MS SQL Server with the following name UserSubscriptionsManagement and run the script in the solutions folder (DBScript.sql) to create the database with some test data
3. Right click on the solution, click properties , set both the UserSubscriptionsManagement.WCFService and the UserSubscriptionsManagement.WebAPI as start up projects
4. Access the web API through the following URLS: <http://localhost:58628/api/users/> and http://localhost:58628/api/subscriptions/