# Understand application creation (steps):

We have created 4 class library projects inside our solution.

1. **Data**: It contains basic entity classes which will be mapped to actual tables.
2. **Service**: It is generic service which gives developer functionality of CRUD operation on particular passed entity.
3. **MyService**: It is entity specific service for doing CRUD operation.
4. **WebApp**: It is actual application to be interacting directly with user. Hence, it contains views, controllers and models. Here models are used for validation.

Below is pictorial layout how’s references are to be added in all 4 projects/components.

* WebApp depends on MyService component (specific to contract entity).
* MyService depends on Service component (generic CRUD operation service).
* Service depends on Data component (table mapping classes)

# Steps:

1. **Data:** Create Contract.cs and Status.cs files in Data component.
2. Also, create BaseEntity abstract class. Contract and Status class must inherit BaseEntity class.

We require BaseEntity to be used as parent class of Contract and Status child class objects.

1. **Service:** Create interface IDbContext.cs and TableContext.cs. TableContext inherits DbContext class and IDbContext interface. IDbContext has “Set” method which need to be implemented in TableContext.
2. Create IRepository interface and then Repository class. Repository must inherit IRepository interface and implement all methods. Repository class must take <TEntity> constructor level argument which is type of BaseEntity.
3. In constructor of Repository class, we have dependency of IDbContext context obj. In which we will pass TableContext Object. Create Mapping folder inside this application. Create contractMap and StatusMap class. Inside constructors, provide key, property and table mapping using HasKey(<expression>), Property(<expression>) and ToTable(“TableName”) respectively.
4. We need to seed some master data (i.e. 2 status records for this app). Hence, we need to use DbInitializer class. Hence, inside constructor of
5. **MyService:** Now, let’s focus on “Contract” entity on which we need CRUD operations.

Create IContractService interface. Create ContractService class which inherits IContractService.

1. Basically, we need to implement all method of CRUS in ContractService which will used by Contract Controller later on.
2. In ContractService constructor, we have dependency of IRepository<BaseEntity> contractRepository obj. In which we will pass Repository<BaseEntity> Object.

Hence, eventually we are creating local object of contractRepository here.

1. Using this local object, we can access all CRUD methods on genertic Repository we created in **Service** part.
2. **WebApp:** In this application, we need to create blank controller called ContractController.
3. In constructor of this controller, we need pass IcontractService from MyService component.

We need to pass ContractService object from MyService component. That we will do using Ninject 3rd party component.

1. Create all related action methods we need to use for CRUD. I have merged CreateContract and EditContract method to CreateEditContract. Other methods are DeleteContract, EditContract, DetailContract etc.
2. Create contractModel class in Models which will help you passing objects from view to controller and vice-versa. In all methods in controllers, we are using contractService object only to eventually call actual CRUD operation in Service>Repository class.
3. Create appropriate views in WebApp and pass model object. Provide HandleError and authorize attribute to ContractController to handle unexpected errors and authorization.
4. HandleError will show default Error Page (Views>Shared>Error.cshtml) and Authorize will lead to Default login page if user is not logged in. (Make sure you create WebApp project with IAM authentication)
5. Install Ninject, Ninject.Web.MVC from NuGet package. You will see NinjectWebcommon.cs in App\_Start folder. There is RegisterServices method available. We need to use Kernal object and call Bind method to map Interface and actual class object. We need to map IDbContext with TableContext, IRepository<> with Repository<>, IContractService with ContractService.
6. In web.config of WebApp and in app.config of Service component, we need connection string named as “DefaultConnection”. Create Unit test project to and provide reference of WebApp.

You can call all action methods of ContractController and check result using Assert class.

Drop me any query on [jay09.parekh@gmail.com](mailto:jay09.parekh@gmail.com)

Thank you !