# Understanding our Layers in Code

We have a lot of layers. This is what keeps us decoupled. This is critical to our ability to move quickly and make massive changes yet minimize the impact of those changes.

## Backend Layers

### WebService

**Description**

This layer contains logic for exposing endpoints.

**What goes here**

Code specific to the EndPoint. If you are adding code here that isn’t specific to the endpoint, you may be adding the code to the wrong place.

**References**

Layers outside the WebService layer should never reference items in the WebService layer. Libraries within the WebService layer may reference common and projects from the Services Layer. Example, all custom web services reference WebService.Common. WebServices.Common references the WebService behaviors. Also, third party libraries that are related to WebServices may be referenced.

**Changes**

Obviously if the route changes, then any callers would be affected. Changes in items in this layer should not affect much. See Web Service Model for the effects of changing a model.

#### WebService Model

**Description**

This is an optional sublayer within the WebService layer. This represents the model returned by a WebService action. It must implement the Entity interface. It should be the same Entity unless there is a compelling need for a custom entity. It should not reference other entities directly. This will come with the RelatedEntity features which will not require couple references.

**What goes here**

Code specific to the EndPoint model. If you are adding code here that isn’t specific to the endpoint model, you may be adding the code to the wrong place. This should have zero logic. Just a class limited to properties and attributes.

**References**

If a WebService model exists, only the WebService should ever reference this model. However, if the model is returned by the WebService, then the consumer will also need the model.

**Changes**

Changes here should only affect the WebService and the Service Model sublayer in JavaScript.

#### WebService Behavior

This is a sub layer of the WebServices layer. This layer is where EndPoint behavior is defined.

**What goes here**

Code specific to the EndPoint behaviors. If you are adding code here that isn’t specific to the endpoint behavior, you may be adding the code to the wrong place.

**References**

It can be referenced by things in the WebServices layer. However, it should not reference anything in the WebServices layer that is not also in this sublayer.

**Changes**

Changes may have an effect on the Model or on the EndPoints. Each change should be investigated for its affect. A behavior could change the model returned by custom serialization, either json or xml, or it could change the EndPoints and how they are called. Mostly, only the caller/consumer would be affected.

## Service

**Description**

This is the logic layer. Now, that doesn’t mean it holds all the logic. Many logic pieces belong in their own dlls, however, this layer would reference and use those logic dlls.

**What goes here**

If you aren’t sure where code should go, it probably should go here. Code specific to the Service, the business logic, etc. If you are adding code here, it is harder to know if it is in the right or wrong place.

Ask what this code is for? What layer would most use this code? If it is clear it that this is the layer, or you have no idea what layer the code should go in after asking those two questions, then you are probably in the right place. However, don’t forget to consult your dev lead.

If it is lots of code, maybe create a separate project and have the Service layer reference it.

**References**

The repository layer and model layers should never reference the Service layer. However, other layers can reference the Service Layer. Try to keep the reference to Services.Common. A custom web service will likely reference a custom Service of the same name. If anything but a custom WebService references a custom Service, it is probably wrong.

Libraries within the Service layer may reference each other. Example, all custom services reference Service.Common.

**Changes**

Changes here can affect the WebServices layer.

#### Service Model

**Description**

This is an optional sublayer within the Service layer. The model used by the Service. It must implement the Entity interface. It should be the same Entity unless there is a compelling need for a custom entity. It should not reference other entities directly.

**What goes here**

Code specific to the Service model. If you are adding code here that isn’t specific to the service model, you may be adding the code to the wrong place. This should have zero logic. Just a class limited to properties and attributes.

**References**

If a Service model exists, only the Service should ever reference this model. The Service should always return the interface, not its own custom model.

**Changes**

Changes here should only affect the Service layer. No other layer should be affected.

### Repository

**Description**

This is should only contain logic needed to talk to the repository. I should avoid having business logic or customizations as those belong in the Services layer.

**What goes here**

Code specific to the Repository. If you are adding code here that does something beyond basic CRUD tasks, you may be putting the code in the wrong place.

**References**

Nothing should ever reference a repository, with one exception. A custom Service. Even then this is only for convenience. Usually a custom service would load the Repostory using plugin loader, without a reference, however, since it is custom, why use PluginLoader if you know 100% that a certain repository will always be used.

**Changes**

If the repo changes, everything should continue to work exactly as is. Changes, should have zero affect on anything. With one exception, a custom service using a custom repo.

#### Repository Model

**Description**

This is an optional sublayer within the Repository layer. The model used by the Repository. It must implement the Entity interface. It should be the same Entity unless there is a compelling need for a custom entity. It should not reference other entities directly.

**What goes here**

Code specific to the Repository model. If you are adding code here that isn’t specific to the repository’s model, you may be adding the code to the wrong place. This should have zero logic. Just a class limited to properties and attributes.

**References**

If a repository model exists, only the repository should ever reference this model.

**Changes**

If the repository model changes, nothing outside of the repository layer should ever know about it.

## Web UI Layers

### View or UI

**Description**

This is your html/css, including the javascript <script> tags but not the javascript itself.

**What goes here**

Hml and css and javascript <script> tag

**References**

It references the javascript that it needs to reference. However, the UI should only call the javacsript methods to kick off binding.

**Changes**

Changes to the UI should affect look and feel and user experience only. It should not affect any other levels. However, there are naming schemes that could affect Binding and ViewModel layers.

### ViewModel

**Description**

This is the view model that the UI binds to. It may be the actual model, with no changes; however, this is usually not the model. This is where any calculated fields should be created for binding.

**What goes here**

ViewModels. Fields to bind to. Calculated fields. Code to convert one or more models into a ViewModel.

Note: Calculated fields should be just-in-time. That means they must go in this layer, so they don’t exist until the UI actually needs them.

**References**

It usually references the javascript service layer’s model and the ViewModel is build using that model.

The UI references this primarily with binding.

**Changes**

Changes here can affect the UI. However, the UI should drive the changes here, not the other way around.

### Controller

**Description**

This is the layer that connects the View and the ViewModel. It really shouldn’t do much more than this.

This is usually the first javascript code that html will call to execute a method.

It usually calls the Service Layer to get the model and gives the model to the ViewModel.

**What goes here**

Controller code to hook up layer interactions.

**References**

It usually doesn’t know about the UI, but does expect the UI to be binding ready. It usually doesn’t know anything about the ViewModel but expects the ViewModel properties to align with the UI binding.

**Changes**

Changes here should not affect much of anything.

### Service

**Description**

This is the layer in which all interactions with the Web Services should occur.

**What goes here**

The code needed to make calls to any web services.

**References**

It is often referenced by the Controller.

**Changes**

Changes to method names could affect the Controller, however, changes to the content of methods, should have no effect on the controller itself.

#### WebService Model

**Description**

This is the model that the WebService returns. It is often the same model the web service uses, but there maybe slight differences, due to serialization, or customizations.

**What goes here**

Usually nothing goes here. This usually comes from the Service as it is. However, in rare cases, it this may need to be manipulated. However, usually such manipulations should belong in the ViewModel.

**References**

It is often referenced by the Controller.

**Changes**

Changes to the model should affect the ViewModel that is the model’s primary consumer.

### More on Model Sublayers

Each layer has sub layer that is a model layer. The reason it is optional is because one model may work up and down a given layer. However, that is not always the case.

All the models will implement the same interface; however, the implementations can be different, if needed. However, that should only occur if needed; otherwise they *should* be the same models. The ability to quickly do data mode translations (DTS) between a model in one layer to a model in another layer, is done by the ToConcrete() method.

## Interfaces and Entities

These layers are different as they are not horizontal stack layers so much as code that used to communicate up and down the layers. However, the could be views as vertical layers the cross-cut through all the other layers.

### Interfaces

**Description**

Contracts that must be implemented between layers.

**What goes here**

Nothing but the contract.

**References**

Should have no references, except Interfaces.Common.

Anything that should reference an interface, can reference an interface.

**Changes**

Changes can be huge. This is the contract for communication between layers. A change could result in two or more layers needing to change to accommodate the contract change.

### Entities

**Description**

These are common entities that can traverse up and down layers. This is the common model. However, if a layer needs a custom model, don’t put it here, put it in that layer.

**What goes here**

Nothing but the model. It should have zero logic. Properties and Attributes are all that should exist here.

**References**

Should have two references, except Interfaces.Common, Interfaces.<SpecificEntity>. However, it can also reference libraries that provide attributes as needed.

**Changes**

Changes here can have affects in the repository and the returned web service object. However, changes here should only occur if a change also occurs to the interface.

## Other Layers

There may be custom layers that are not part of the framework. Or legacy layers.

### Authenticators

This is a plugin layer for authentication. All authentication should be done here.

### Token Validators

This is a plugin layer for Token validation. All token validation should be done here.

### License Generator

This includes the code needed to implement our primary LicenseGenerator service. Currently this is not in the above framework but can be moved there. This is actually broken up into multiple layers itself.

## Naurtech Licensed Device Layers

This is the layers needed for Naurtech’s custom licensing page to function. Currently this is not in the above framework but can be moved there. This is actually broken up into multiple layers itself.

### Data Access Layer (DAL)

This is a legacy layer used by the Naurtech layer. It is scheduled to be decommissioned when Naurtech moves to EAF from its custom services.

## Tips

If you are having a hard time deciding where code should go here are some hints:

1. Use the Just-in-time principle:
   1. Ask yourself: What layer most uses/needs this code?
   2. Answer: Usually that layer is where the code should go.