## Unit Tests

Moq and Nunit are included to conduct the unit tests (I have a personal copy of ReSharper, so have performed all test runs via that UI).

The ResourceAccess tests are fairly simple, ensuring that expected data is returned from the connected Data Store via the DataAccess Library.

Basing “required” business requirements off of the nullable/not-nullable fields in the DB Schema script, I placed business logic checks in the Manager library level to verify that expected data was coming through prior to executing a call to the Resource Access Layer - there are tests in place related to these checks (\* acknowledging that it is always better to check at the UI level for required fields before submitting a post, but making the assumption that this Address Component could be called from any number of requesting applications and there may be no solid assurance that these validation checks are occurring on the client).

SmokeTest/ValidationClient - hard-coded IDs representing the 3 addresses inserted in the DB scripts to retrieve a minimal set of addresses by ID to verify the concrete implementation. Links available to edit existing address or to create a new address to again verify concrete implementation. \*This project was wired up with the Unity for MVC library

## Design Choices

Based of my recent experiences at Allrecipes, I chose to continue down this pattern as this represents my experience with Dependency Injection and Unit Testing. It is debatable if there needs to be a separation between the ResourceAccess layer and the DataAccess Layer, but our consensus when discussing the architecture was to allow for unit-testable single responsibility of retrieving data, packaging it in our container with additional status information, and returning it to the business/manager layer. In the event the DatAccess layer would ever need to switch datastores, we felt that we could leave the ResourceAccess code and tests unaffected by such data store swap-outs.

When catching exceptions, I am presuming there is an existing error logging system available to call, so I indicated a Console.WriteLine() and a comment to log to the applicable system.

When calling the component to save an address, the client application will have the form to enter the address data and will be aware of what the end user had submitted, so I opted to go with the choice of simply returning a boolean value for the Save Address attempt (it seems like extra weight to return the submitted address).

The DataAccess layer performs a TryGet to lookup an address ID, so that if the query does not return an address for the specified ID, a return value of false can bubble up the calling chain to inform the client that an address does not exist for a specified ID.

## Thoughts about future use, upgrades, challenges, etc.

As for the DB Schema, the current state will only allow for US/Canada addresses, given the entries in the State table. Given our phone conversation, Paulas Choice has an Australian presence and plans to grow out into multiple European presences,

Citing only this StackOverflow [post](http://stackoverflow.com/questions/1159756/how-should-international-geographical-addresses-be-stored-in-a-relational-databa) and [this](http://stackoverflow.com/questions/929684/is-there-common-street-addresses-database-design-for-all-addresses-of-the-world), it seems likely that a brainstorming/planning session would need to happen as to how the DB Schema should be restructured to optimally account for all flavors of International Address Formats.

I have not had projects tasked with keeping up with the current state of address verification software - I can attest when I did have it as a requirement for a project back in ~ 2005, it was a little on the messy side. I would hope that companies providing these services would have made significant progress since then. From a business stakeholder perspective, validating an address prior to saving it the system is always ideal, so future enhancements to the system may involve validation against some Address Verification API that has the opportunity to prompt the users to correct invalid/unverifiable address data that they are submitting.