# Project Structure

The code is developed with Visual Studio 2015 and C#. NET version 4.5.2 using the MVC 5.2.3 Framework.

If you open the CatsWebApp.sln file you will notice two projects namely –

1. CatsWebApp – The MVC web application that contains all the code needed to complete this exercise
2. CatsWebApp.Tests – The MS Test project that supplies a simple yet comprehensive unit test harness for the various C# objects used in the application.

## CatsWebApp

Technologies used – C# 6.0, ASP.NET 4.5, MVC 5.2.3, Resharper 2015.1.3

Nuget packages – Log4net, Elmah, NewtonSoft.Json, AngularJS

Considerations – use Linq, HttpClient and JQuery where required

The project is inherited from the Vanilla MVC application that comes with VStudio. I have taken care to remove any unreferenced code however there might be some remnants in JavaScript.

In a nutshell the project uses C# to extract the backend web service Json information using the model and controller objects. The data is then sampled to produce the required output, which then is converted to Json and handed over to AngularJS for rendering.

The following is the folder structure –

|  |  |
| --- | --- |
| Folder | Description |
| App\_Start | Contains the App startup code.  I have modified the BundleConfig.cs to initialize AngularJS.  The WebApiConfig.cs logs an entry to log4net indicating the application has started |
| Controllers | Contains the HomeController that hosts the back-end logic for the Index.cshtml page.  The GenerateResults method gets invoked by the UI, which in turn extracts the data from the web service and generates results to be sent back to the UI.  As you will notice the code is written with proper error handling where internal errors are consumed by log4.net and user is only informed with a high-level error message.  In addition all web errors that aren’t trapped here get reported to ELMAH.  You may notice I have an internal GenerateResults method that is exposed to the unit test using InternalVisibleTo in the AssemblyInfo.cs |
| Models | This folder comprises of the Models.cs which stores all the input and output data objects.  You will also notice I have two JsonConveters namely GenderConverter and PetTypeConverter which are used by the Owner and Pet model objects for serialization/deserialization to enums |
| Scripts | This folder contains two files app.js and service.js for initializing the Angular app and service instances.  The service.js incorporates a generateResults method that invokes the GenerateResults method from the HomeController |
| Scripts\Controllers | Contains the catsWebAppController.js file that’s instrumental in extracting the Json result from the service and forwarding it to the screen-rendering engine. |
| Utilities | This folder contains some files that do the grunt work for the back-end   1. DataExtractor.cs contains the code to extract Json data from the webservice, convert this data into C# readable format and produce and output object for the UI. 2. Logger.cs contains all the code to perform log4Net and event logging. |
| Views/Home | Contains the Index.cshtml which renders the data collected by the CatsWebAppController using AngularJs directives |
| Views/Shared | \_Layout.cshtml has been modified to display the desired output. |
| / | The following files have been modified –   1. Global.asax – to instantiate Log4Net 2. Startup.cs – To bind the web.config log4net settings to the application 3. Web.config – contains the WebServiceAddress pointing to the input web service and the nitty-gritties for Log4net and Elmah |

## CatsWebApp.Tests

This project is laid out in the same hierarchy of the CatsWebApp. Each folder contains the unit tests for their corresponding classes. At the time of writing this document there are 49 unit tests developed that comprehensively test all known conditions in the application.

# Workflow

The following steps were taken while developing this project (in chronological order) –

1. I checked for the validity and functioning of the source web service. I noted down the requirements, developed a rough skeleton of the project structure and started writing my unit tests.
2. I went on with data modeling and started designing my model classes.
3. I started modifying the MVC project by adding dependencies and removing unwanted code.
4. Once I had designed the controller on paper I started off enhancing the unit tests and moving on with connecting the controller to the model. Once the code was in place I refactored the code by moving the essentials into the Utilities folder.
5. The UI design done in AngularJS simply by rendering the outputs to the landing page as per the requirement.

I estimated 1.5 days of effort for the entire exercise had I some more time I would have made the following enhancements –

1. Added a drop down list with Pet types for user to filter different types and not be constrained to only Cats as per the requirement
2. Used Selenium for UI testing
3. Used SpecFlow to incorporate BDD instead of TDD
4. Used MoQ for end-to-end testing
5. Developed an end-to-end build and deploy process using CruiseControl or Jenkins.

* Mukesh Bhakta

25th August 2016