**Asp.Net Mvc Assessment Exercise**

**Part I**

**Overview**

The purpose of this exercise was to create a ASP.NET MVC 5+ application that will accept several different contact fields and store the data to an associated Database table using Entity Framework 6+.

**Approach**

The approach I took was to first focus on the main functionality aspects first and then then user interface. I began by creating the Database Project to ensure that the model I was using will align with the requirements listed. After using the provided Schema to create the database table on my local environment to test with, I switched my focus to start creating the skeleton and structure of the solution itself. Since I already had the Database project the next project I created was the Core project where I will be able to keep the model and any other core components that could be required in the future. After completing the core project, the next aspect, I created was the Data Access layer, which house the connections to any databases needed and would allow me access to implement CRUD operations on the data stored within the database. The first thing I needed to do while creating the Data Access layer was to ensure I was using Entity Framework 6+ and to begin to create the necessary Db Context. After setting up the context I created the repository files to implement the CRUD operations on the table. After all the plumbing has been setup for the Data layer to update and alter the data, I turned my focus on creating the Infrastructure project which would be the area where all the managers will be for all the various controllers that will exist on the dot net site and any possible gateways for the future. The next step I took in the project was to create the ASP.Net Mvc 5+ web project and begin to setup the index page as well as the controller that will utilize the manager that was created in the Infrastructure project to interact with the Data itself in the Database. After the base index page was setup, I did run into a slight issue when setting up the controller due to a dependency issue that was occurring because I was using a controller constructor with parameters for the main index controller, which caused the MVC project to throw an error. To resolve that issue, I decided to implement Structure Map to ensure that any further dependency injections would be handled properly. Once the dependencies were handled and I was able to successfully save a contact to the database table I changed my focus to the data validation and UI aspects. I did have to do a little research while implementing the options for the 15 min increments needed for the Best Time to Call requirement but was able to get the validation working successfully. After implementing all the validations required I implemented the CAPTCHA which was straight forward. After double checking all the requirements and validations required where in place and correct, I began to create the CSS stylesheet to give the form an appealing style and format the form. The design I choose was to implement the contact form in a simple and easy to understand format.

**Part III**

Whenever I am working on something I always try to follow the principal KISS (Keep It Simple Stupid) to keep things as easy to understand and easy to understand. With keeping it simple in mind I would start to build/write the skeleton of the form with a custom Id, so we can pull the information in the form by Id if needed using the submit button off screen. I would then start with the creating the container and header and working my way down. To create the selection parts for the form described, I would start with creating a radio button for each selection option, "Paint", "Carpet/Floors", and "Blinds" creating the labels at the top of the selection area. I would also add in a specific CSS class for each of the different options and apply it to each radio button to get the desired color scheme and visual effect. After creating the radio buttons, I would then create the "Notes" section using a simple textbox. Then I would create the "Choose Photo" input and utilize the file type. I would also implement an ajax call to ensure the name of the file is updated in case the user changes the file.

Once the skeleton has been created I would then go back in and begin formatting and creating any missing CSS classes needed to apply the formats required. I would start by making a div for the body to ensure it’s in the right place on the page if needed, then I would create the div for the header to get placement/style/color and continue formatting working my way down for each of the objects.