Vision Plan

For MVC Reporting Website Framework

(MVC RWF)

Version 1.1

Submitted in partial fulfillment of the Masters of Software Engineering Degree.

Thaddeus Thomas Tuck

CIS 895 – MSE Project

Department of Computing and Information Sciences

Kansas State University

Committee Members

Dr. Daniel Andresen

Dr. Mitchell Nielsen

Dr. William Hsu

Change Log

Version #	Changed By	Release Date	Change Description
1.0	Thaddeus Tuck	3/25/2018	Initial Release
1.1	Thaddeus Tuck	4/19/2018	Correction of Requirement

List of Figures

Figure 1: Project Overview 7
Figure 2: Report Website Framework Data Flow Diagram 12
Figure 3: Developer Installation Use Case 14
Figure 4: Developer Standard Use Case 15

Table of Contents

Change Log **2**

List of Figures **3**

1. Introduction 6
   - 1.1 Motivation 6
   - 1.2 ASP.NET MVC 6
   - 1.3 Reporting Website Framework 7
   - 1.4 Terms & Definitions 8
   - 1.5 References 10
2. Project Overview 11
   - 2.1 Introduction 11
     - 2.1.1 Controller 11
     - 2.1.2 Service 11
     - 2.1.3 Repository 11
   - 2.2 Project Goal 13
   - 2.3 Project Purpose 13
3. Project Requirements 13
   - 3.1 Model Requirements 15
     - 3.1.1 MRI 1 [Critical Requirement] 15
     - 3.1.2 MRI 2 [Critical Requirement] 16
     - 3.1.3 MRI 3 [Critical Requirement] 16
     - 3.1.4 MRI 4 [Critical Requirement] 16
     - 3.1.5 MRI 5 16
     - 3.1.6 MRI 6 [Critical Requirement] 16
   - 3.2 Structural Requirements 17
     - 3.2.1 SRI 1 [Critical Requirement] 17
     - 3.2.2 SRI 2 [Critical Requirement] 17
     - 3.2.3 SRI 3 [Critical Requirement] 17
   - 3.3 View Requirements 18
     - 3.3.1 VRI 1 [Critical Requirement] 18

3.3.2 VRI 2 [Critical Requirement] 18
3.3.3 VRI 3 [Critical Requirement] 18
3.3.4 VRI 4 [Critical Requirement] 18
3.4 Extensibility Requirements 19
3.4.1 ERI 1 [Critical Requirement] 19
3.4.2 ERI 2 19
3.4.3 ERI 3 19
3.5 Performance Requirements 20
3.5.1 PRI 1 [Critical Requirement] 20
4. Assumptions 20
5. Constraints 20
6. Environment 21

1. Introduction

1.1 Motivation

The motivation of the project is to develop a framework that can be used by any C# developer to quickly deploy a reporting website while providing optimal customizability for an enterprise level website.

One of the biggest challenges when trying to develop a web product is the various skills of the developers involved and the threat of copying and pasting complex HTML and JavaScript between similar pages. Creating a framework that allows developers to use C# to instantiate models that can then be processed into a web page (View) without having to redefine the actual html each time creates an environment of reusability and extensibility.

By using standardized views and models it is relatively easy to introduce complex standardized functionality such as filtering, column selection, and column order for table based reports. To facilitate this, standardized patterns need to be created and followed.

ASP.NET MVC which stands for model, view controller is an application programming interface (API) that provides the functions and underlying utilities for build dynamic websites using C#.

1.2 ASP.NET MVC

“ASP.NET MVC gives you a powerful, patterns-based way to build dynamic websites that enables a clean separation of concerns and that gives you full control over markup for enjoyable, agile development. ASP.NET MVC includes many features that enable fast, TDD-friendly development for creating sophisticated applications that use the latest web standards.” [1]

1.3 Reporting Website Framework

The basis for the Reporting Website Framework is to extend base classes that contain standard functions and operations to prevent the reimplementation of standardized code. The goal of this project is to implement the full architecture of MVC with a service layer for additional extensibility. In addition, by creating overridable default layouts and views the framework can provide entry-level web developers with a fully functioning website while still allowing for extensibility as their skills progress as well as configurability without having to have extensive experience with JavaScript, jQuery, or CSS.

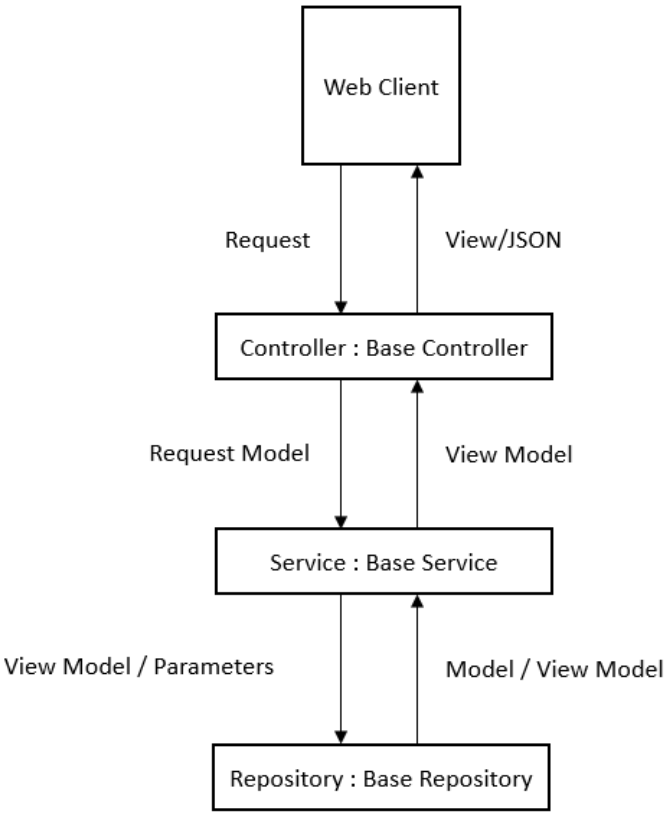

Figure 1: Project Overview

1.4 Terms & Definitions

MVC – "The Model-View-Controller (MVC) architectural pattern separates an application into three main components: the model, the view, and the controller." [2]

Models – "Model objects are the parts of the application that implement the logic for the application's data domain. Often, model objects retrieve and store model state in a database. For example, a Product object might retrieve information from a database, operate on it, and then write updated information back to a Products table in a SQL Server database." [2]

View – "Views are the components that display the application's user interface (UI). Typically, this UI is created from the model data. An example would be an edit view of a Products table that displays text boxes, drop-down lists, and check boxes based on the current state of a Product object." [2]

Controller – "Controllers are the components that handle user interaction, work with the model, and ultimately select a view to render that displays UI. In an MVC application, the view only displays information; the controller handles and responds to user input and interaction. For example, the controller handles query-string values, and passes these values to the model, which in turn might use these values to query the database." [2]

Action – Actions are public methods within a Controller that serve as invokable methods accessed from the web application by /{Controller}/{Action}.

Service Layer – "A service layer is an additional layer in an ASP.NET MVC application that mediates communication between a controller and repository layer. The service layer contains business logic. In particular, it contains validation logic." [3]

Repository – "A repository acts like a middleman between the rest of the application and the data access logic. A repository isolates all the data access code from rest of the application." [4]

Layout – "This layout defines a top level template for views in the app. Apps don't require a layout, and apps can define more than one layout, with different views specifying different layouts." [5]

Razor Syntax – "Razor is a simple programming syntax for embedding server code in web pages. The Razor syntax gives you all the power of ASP.NET,

but is using a simplified syntax that's easier to learn if you're a beginner, and makes you more productive if you're an expert." [6]

ViewData – "ViewData is a dictionary object that you put data into, which then becomes available to the view. ViewData is a derivative of the ViewDataDictionary class, so you can access by the familiar "key/value" syntax." [7]

ViewBag – "The ViewBag object is a wrapper around the ViewData object that allows you to create dynamic properties for the ViewBag." [7]

DataTables – "DataTables is a plug-in for the jQuery Javascript library. It is a highly flexible tool, build upon the foundations of progressive enhancement, that adds all of these advanced features to any HTML table." [8]

Extension – "Extension methods enable you to add methods to existing types without creating a new derived type, recompiling, or otherwise modifying the original type. An extension method is a special kind of static method, but they are called as if they were instance methods on the extended type." [9]

1.5 References

[1]"MVC", *The Official Microsoft ASP.NET Site*, 2018. [Online]. Available: https://www.asp.net/mvc. [Accessed: 13- Mar- 2018].

[2]"ASP.NET MVC Overview", *Msdn.microsoft.com*, 2018. [Online]. Available: https://msdn.microsoft.com/en-us/library/dd381412(v=vs.108).aspx. [Accessed: 13- Mar- 2018].

[3]"Validating with a Service Layer (C#)", *Docs.microsoft.com*, 2018. [Online]. Available: https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/models-data/validating-with-a-service-layer-cs. [Accessed: 13- Mar- 2018].

[4]B. Joshi, "Using the Repository Pattern in C# with ASP.NET MVC and Entity Framework", *Codeguru.com*, 2018. [Online]. Available: https://www.codeguru.com/csharp/.net/net_asp/mvc/using-the-repository-pattern-with-asp.net-mvc-and-entity-framework.htm. [Accessed: 13- Mar- 2018].

[5]S. Smith, "Layout in ASP.NET Core", *Docs.microsoft.com*, 2018. [Online]. Available: https://docs.microsoft.com/en-us/aspnet/core/mvc/views/layout. [Accessed: 13- Mar- 2018].

[6]"ASP.NET Razor C# Syntax", *W3schools.com*, 2018. [Online]. Available: https://www.w3schools.com/asp/razor_syntax.asp. [Accessed: 13- Mar- 2018].

[7]R. Appel, "When to use ViewBag, ViewData, or TempData in ASP.NET MVC 3 applications – Rachel Appel", *Rachelappel.com*, 2018. [Online]. Available: http://www.rachelappel.com/when-to-use-viewbag-viewdata-or-tempdata-in-asp-net-mvc-3-applications/. [Accessed: 13- Mar- 2018].

[8]"DataTables | Table plug-in for jQuery", *Datatables.net*, 2018. [Online]. Available: https://datatables.net/. [Accessed: 13- Mar- 2018].

[9]P. Mehra, "Extension Methods in C#", *C-sharpcorner.com*, 2009. [Online]. Available: https://www.c-sharpcorner.com/UploadFile/puranindia/extension-methods-in-C-Sharp-3-0/. [Accessed: 13- Mar- 2018].

2. Project Overview

This section provides information about the structures and goals of the Reporting Framework project.

2.1 Introduction

The block diagram in Figure 2 gives an overview of the internal working of the framework with an example of a standard work flow for the website when the framework is used to create a table-based report.

2.1.1 Controller

The action calls the service to be used to create the table model. Then sets any necessary view settings for the action. Finally, the action passes the view model to the function to return the cshtml view or JSON Response.

2.1.2 Service

The service should contain at least two functions if complete extensibility is desired; one function that serves as the public accessor for the controller and one function that serves as the functional method that can be overridden by a consuming application. The public accessor will call a function of the base service class that will bind the user provided parameters into a request model that will then be passed into the functional method. The functional method will contain the business logic and calls to the repository to create the desired model to be used for/by the view.

2.1.3 Repository

The repository contains the logic that loads data from the data source typically Entity Framework or direct SQL queries. For this project the data source is not relevant as the repository should be the connection between the model that represents the view and the data. For the example of table views the repository is responsible for the creation of the columns and rows for the table model.

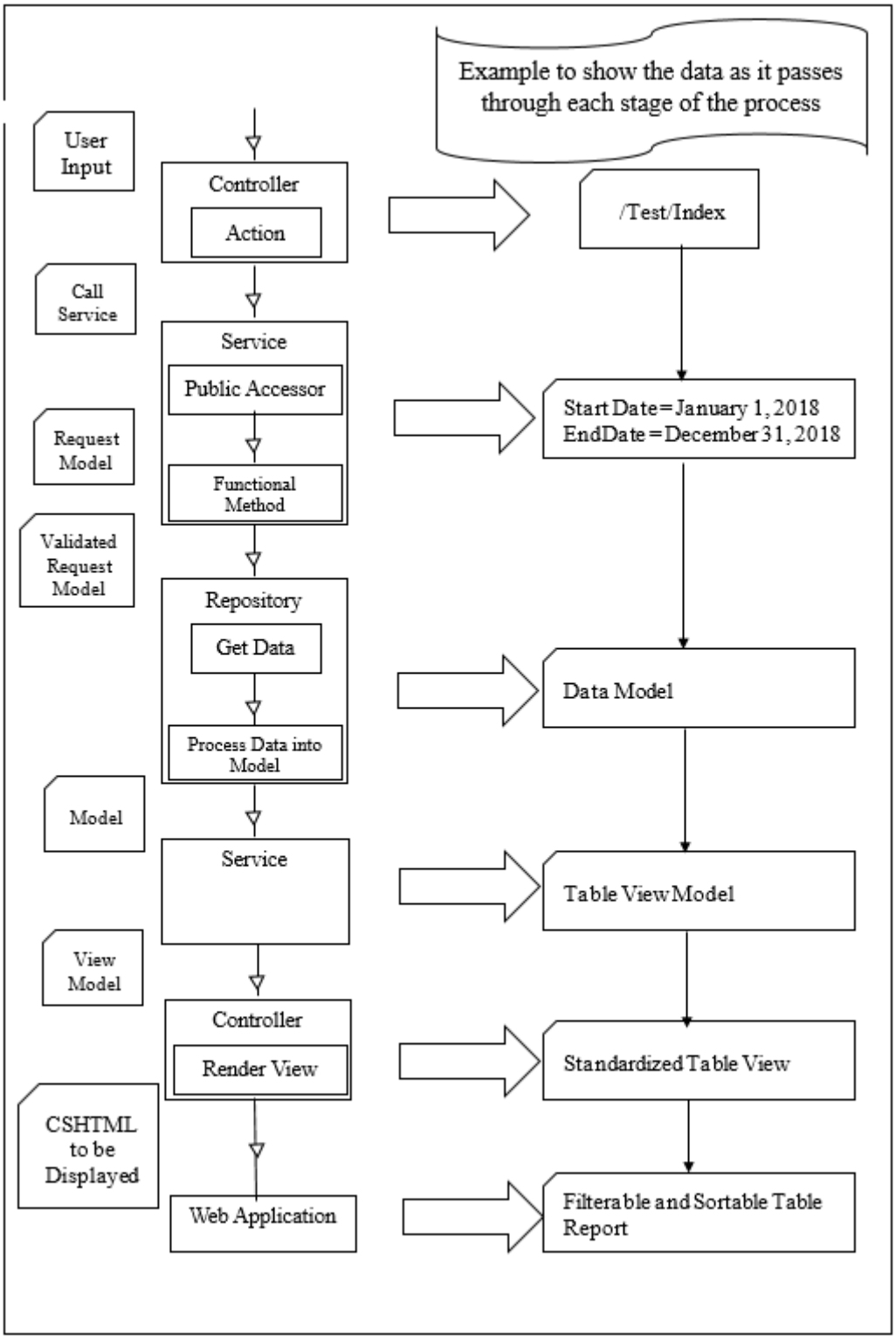

Figure 2: Report Website Framework Data Flow Diagram

2.2 Project Goal

The primary focus of the MVC Reporting Website Framework is to provide an ease of use and development in setting up an Enterprise level MVC web application without the need for extensive copy/paste. This project will ultimately be packaged as a NuGet package that when installed into an MVC ASP.NET project in Visual Studio can be utilized to quickly setup DataTable based reporting and other website features without the need to reimplement any HTML, JavaScript, or CSS. A C# developer should be able to use the framework to easy setup a new report using the pattern and instructions provided with the project.

2.3 Project Purpose

This framework is developed to fulfill the need for a development team of varying web-development experience to be able to implement, deploy, and maintain an Enterprise level web project while respecting best-practices and standardized implementations for features. The business side of most projects will tell development that they want a new feature that works like “a”, or that they want to make a change that will take effect on an entire feature set; e.g., a change to the way a filter works, or a new filter. By using this framework, the development team will be able to facilitate these requests quickly and easily without the need to duplicate code in multiple locations.

3. Project Requirements

This section provides information about the requirements for the MVC Reporting Framework. Each requirement is given a unique number and discussed in detail. Critical requirements will be indicated in the description and the targeted release that will fulfill the requirement.

Figure 3 is the use case diagram that visualizes a developer installing the framework. The developer will create an empty ASP.NET MVC C# project in Visual Studio then remove the default files as will be detailed in the installation instructions. The developer will then install the NuGet package for the reporting framework and setup any configuration changes they would like to make. The final step is to run the website and verify their configuration changes.

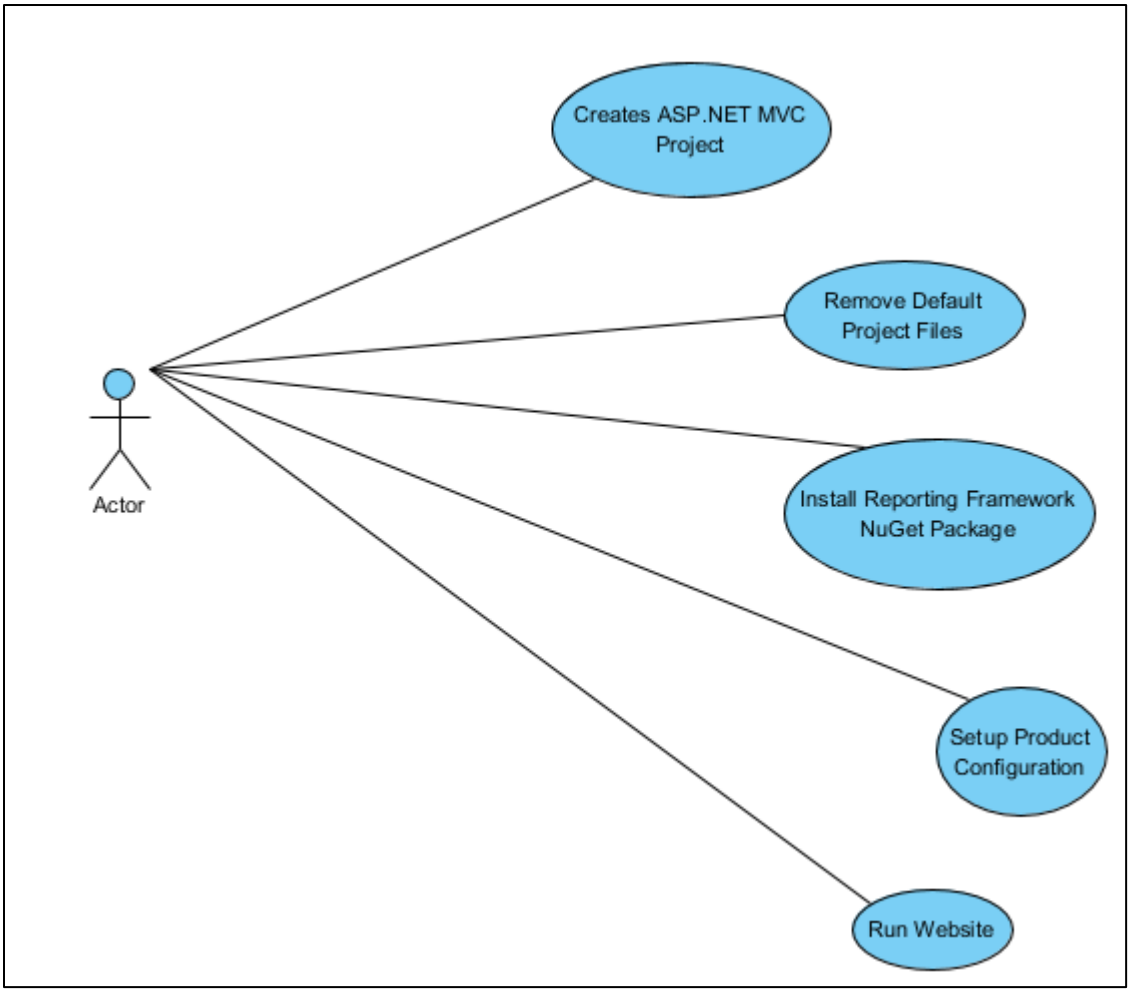

Figure 3: Developer Installation Use Case

Figure 4 is the use case diagram that visualizes a developer using the framework to create a feature or page of the product. The developer will create the models to contain data necessary for the views extending the framework models for the selected rendering framework e.g., DataTable, Grid, Calendar, or a Custom View. Once the models are created the developer will create a repository that extends the base framework repository to fill the models with the necessary data. A service is then extended from the framework base service to access and utilize the developer's repository. The last required task for the developer is to create a controller that extends the framework base controller with actions that return the desired view using the filled model from the service. An optional step for the developer is to define custom partial views for CSS and/or JavaScript.

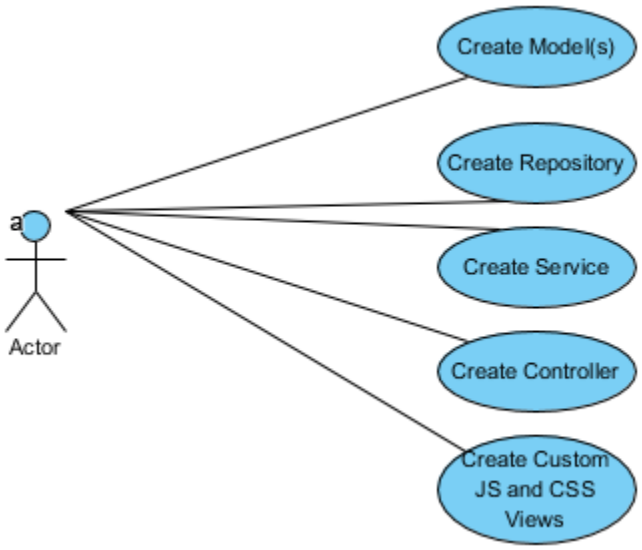

Figure 4: Developer Standard Use Case

The requirements in the below sections should be fulfilled by the final NuGet package and instructions. These requirements are divided into different modules by their separation of concerns. They are – Model Requirements, Structural Requirements, View Requirements, Extensibility Requirements, Performance Requirements.

3.1 Model Requirements

The requirements in this section outline the existence, necessary properties and features for framework models that will allow standardized layouts and views for framework features. The requirements are numbered as MRI X, where MRI stands for Model Requirement Item.

3.1.1 **MRI 1** [Critical Requirement]

The framework shall have a model that will represent the cell of a table. This is a critical requirement so that the framework can render the HTML element that represents a cell in a table.

- **Build Release Applicability**: Demo 2, Final Release

3.1.2 MRI 2 [Critical Requirement]

The framework shall have a model that will represent the row of table. This is a critical requirement so that the framework can render the HTML element that represents a row in a table.

- **Build Release Applicability**: Demo 2, Final Release

3.1.3 MRI 3 [Critical Requirement]

The framework shall have a model that will represent the column of table. This is a critical requirement so that the framework can render the HTML element that represents a column in a table.

- **Build Release Applicability**: Demo 2, Final Release

3.1.4 MRI 4 [Critical Requirement]

The framework shall have a model that will represent the table. This is a critical requirement so that the framework can render the HTML element that represents the table.

- **Build Release Applicability**: Demo 2, Final Release

3.1.5 MRI 5

The framework shall have an underlying model that can be used by the framework and the end developer to indicate if an error has occurred.

- **Build Release Applicability**: Demo 2, Final Release

3.1.6 MRI 6 [Critical Requirement]

The models that represents the columns and cells of a table should have properties and functions to enable filtering by the reporting portion of the framework. This is a critical requirement so that the framework can provide a standard desired feature of reporting applications.

- **Build Release Applicability**: Demo 2, Final Release

3.2 Structural Requirements

The requirements in this section outline the existence, necessary properties and features for the structural components of the framework that provide the base standardized classes. The requirements are numbered as SRI X, where SRI stands for Structural Requirement Item.

3.2.1 **SRI 1** [Critical Requirement]

The framework shall have a base repository that will provide standard functions to assist with model binding as well as standard processing for standardized views. This is a critical requirement as the repository is a critical layer/component of the MVC architecture.

- **Build Release Applicability**: Demo 1, Demo 2, Final Release

3.2.2 **SRI 2** [Critical Requirement]

The framework shall have a base service that will provide standard functions to assist with processing for requests as well as establishing the connection to the repository for the service. This is a critical requirement as the service layer provides for validation and integration between the controller and repository layers of the MVC architecture.

- **Build Release Applicability**: Demo 1, Demo 2, Final Release

3.2.3 **SRI 3** [Critical Requirement]

The framework shall have a base controller that will provide standard functions to assist with processing view models into framework supported views. As well as, establishing the connection to the service for the controller. This is a critical requirement as the controller is most critical component of the MVC architecture as it provides the HTML for the consuming web application.

- **Build Release Applicability**: Demo 1, Demo 2, Final Release

3.3 View Requirements

The requirements in this section outline the existence, necessary properties and features for the view components of the framework that provide the base standardized layouts and partials. The requirements are numbered as VRI X, where VRI stands for View Requirement Item.

3.3.1 **VRI 1** [Critical Requirement]

The framework shall have a base layout that has sections for CSS, JavaScript, parameters, copyright information. This is a critical requirement as the base layout is used by every view for the appearance of the overall website.

- **Build Release Applicability**: Demo 2, Final Release

3.3.2 **VRI 2** [Critical Requirement]

The framework shall have a layout for a DataTable report. This is a critical requirement as the layout for the DataTable report provides the locations for the components of the final DataTable report view.

- **Build Release Applicability**: Demo 2, Final Release

3.3.3 **VRI 3** [Critical Requirement]

The framework shall have a view for a DataTable report that is definable by the models that represents the table and its components. This is a critical requirement as a view is required to render the model as HTML to the browser or web application.

- **Build Release Applicability**: Demo 2, Final Release

3.3.4 **VRI 4** [Critical Requirement]

The framework shall have partials for the HTML, CSS, and JavaScript needed for filtering, column selection, and column ordering for the DataTable. This requirement is critical these functions are standard features of reporting.

- **Build Release Applicability**: Final Release

3.4 Extensibility Requirements

The requirements in this section outline the existence of, necessary properties, features, classes, and extensions for the framework that will allow the end developer to customize and override the components of the framework. The requirements are numbered as ERI X, where ERI stands for Extension Requirement Item.

3.4.1 **ERI 1**

The framework shall have a service provider that can be used to allow the end developer to create additional web projects that can utilize, extend, override components of the MVC Reporting Framework.

- **Build Release Applicability**: Final Release

3.4.2 **ERI 2**

The framework shall have extension classes that can be used to customize standard options for whether features are enabled or disabled by default.

- **Build Release Applicability**: Final Release

3.4.3 **ERI 3**

The end developer shall have the ability to use multiple ASP.NET MVC projects that reference the MVC Reporting Framework NuGet package to create a fully extensible website.

- **Build Release Applicability**: Final Release

3.5 Performance Requirements

The requirements in this section outline the necessary performance and latency expectations for the site features and reporting. The requirements are numbered as PRI X, where PRI stands for Performance Requirement Item.

3.5.1 **PRI 1** [Critical Requirement]

The framework should be able to use post loading to render and be able to search and filter at least 80,000 rows with 20 columns in less than 30 seconds from the beginning of DataTable initialization. This is a critical requirement to demonstrate that the reporting portion of the framework is viable for enterprise level reporting applications.

- **Build Release Applicability**: Final Release

4. Assumptions

- The developer will use the latest version of ASP.NET MVC.
- The developer(s) will use Visual Studio (preferably 2017) when developing using the framework.
- The developer will use Razer for the view component language.
- The framework will not be concerned with how data is loaded into the repository.
- If the developer overrides a portion of the framework they are responsible for ensuring that necessary functions remain consistent.

5. Constraints

- The framework's abilities in relation to its standard reporting features are constrained by the jQuery DataTables API.
- The project and resource extensibility of the framework is provided by the Embedded Resource Virtual Path Provider NuGet package delivered through NuGet built into Visual Studio.

6. Environment

- Visual Studio 2017 will be used to develop the MVC Reporting Framework.
- .NET Framework 4.6.1 will be used and any project that uses the MVC Reporting Framework should utilize .NET Framework 4.6.1.
- Version control will be handled through Visual Studio Team Services.
- This project and all utilizing projects will be written using C#.