CrowSoft C# Coding Standards

Contents

[Why Coding Standards? 2](#_Toc6213953)

[Naming Conventions 2](#_Toc6213954)

[Classes and Methods 2](#_Toc6213955)

[Local Variables 2](#_Toc6213956)

[Namespaces 2](#_Toc6213957)

[Comments 3](#_Toc6213958)

[Language Guidelines 3](#_Toc6213959)

[Best Practice MVC .Net Core Coding 4](#_Toc6213960)

[Models 4](#_Toc6213961)

[Controllers 5](#_Toc6213962)

[Views 5](#_Toc6213963)

[Auto Generate Controllers and Views 6](#_Toc6213964)

[Bibliography 7](#_Toc6213965)

## Why Coding Standards?

* To create consistency how the code is developed
* Easy for developers to read and understand the code for maintenance purposes
* Establish a best practise for C# coding and MVC

## Naming Conventions

**Please note, these naming conventions are based on Microsoft.Net internal standards throughout its development IDE’s (Visual Studio IDE), by its intellisense formatter and tooltips.**

### Classes and Methods

Use PascalCasing for both Classes and Methods. Pascal casing is when the word starts with a Upper case and if more than one word is combined, each word will start with an upper case letter. E.g. **C**ustomer**A**ccount. Make sure the method name is clear and a proper word that can be easily understood.

public class Customer

{

public Customer()

{

}

public void CreateCustomer()

{

}

}

### Local Variables

Use camel casing (E.g. camelCasing). Variables passed as parameters will also use camel casing.

public void CreateCustomer(int customerId)

{

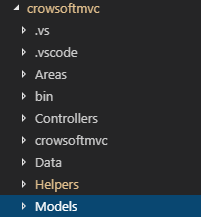
var newCustomerId = customerId;

}

### Namespaces

The application name is in lower case, and any folders use Pascal Casing

Folder example below



Namespace example below.

namespace crowsoftmvc.Models

### Comments

#### Commenting your class

Use 3 forward slashes into describe your classes and methods /// (**Note**, Using this method is a standard way to document when using a more advance IDE, such as Visual Studio Community 2017 edition. This will give the developer the ability run automated documentation against its code.)

/// <summary>

/// This class represent a Customer object

/// </summary>

public class Customer

{

#### Commenting complex code

Use 2 forward slashes, comment delimiter (//) to explain complex code in a short statement.

// Get the User Object from the database where email address is found

var user = await mockupDbContext.UserAccount.SingleOrDefaultAsync

(p => p.EmailAddress == userAccount.EmailAddress);

## Language Guidelines

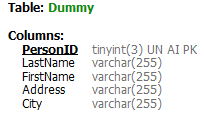
Please refer to Microsoft Documentation for language guidelines. C# Coding Conventions (C# Programming Guide) (Microsoft, 2015) <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions>

## Best Practice MVC .Net Core Coding

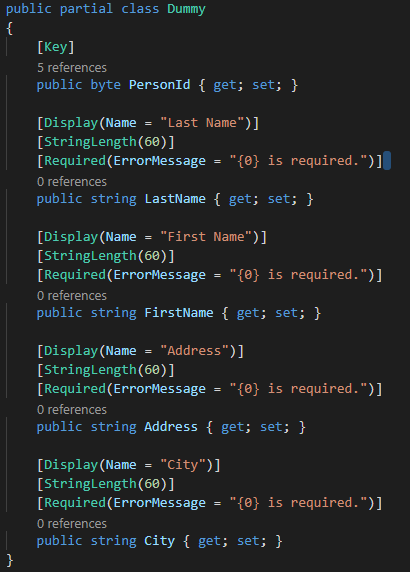
### Models

Your Model always represents your database table. In our example, we will use a table called Dummy. It is extremely important to add data annotations to your Model class. This will be used to auto-generate your Controllers and UI validation rules to the Views.

This is the table on MySQL



Below find the Model class in our code:



Note the following key aspects:

* PersonId is the primary key on the database table. This is auto-generated on the database. You need to specify the [Key] data annotation above the column declaration.
* Always use get; set; properties to declare a field in your Model.

public byte PersonId { get; set; }

* Add the DataAnnotations library to the using statements above the namespace of the model

using System.ComponentModel.DataAnnotations;

Note the data annotations below.

* [Display(Name = “”)] is how the name will be displayed on a list or screen on the Views
* StringLength is to limit the input characters on the UI.
* Required will validate that a user entered a value in this field before submitting a form.

[Display(Name = "Last Name")]

[StringLength(60)]

[Required(ErrorMessage = "{0} is required.")]

For more info on data annotation, go to this: <https://www.tutorialspoint.com/asp.net_mvc/asp.net_mvc_data_annotations.htm> (ASP.NET MVC - Data Annotations)

### Controllers

The Controller executes specific actions relating top the Model and called by the View. The controllers will action the CRUD; Create, Read, Update and Delete. The controller normally handles all the incoming requests from a View, using the Model class to return data to the View.

The ApplicationDbContext object needs to be passed as a parameter to the controller to establish a database connection.

public class UserAccountsController : Controller

{

private readonly ApplicationDbContext \_context;

private Counties myCounties = new Counties();

public UserAccountsController(ApplicationDbContext context)

{

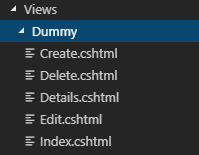
\_context = context;

}

### Views

Views are HTML files generating responses from the user interface. When you use auto generate

We use cshtml files in the new .Net Core MVC web application. It is called Razor Pages which are page focused. Find examples of the CRUD pages created by using the command line, refer to Heading:



### Auto Generate Controllers and Views

You can use the scaffolding tool to auto generate Controllers and razor pages (Views) from your Model.

First, Install the codegenerator using the following command

dotnet tool install --global dotnet-aspnet-codegenerator

Use the following command line to generate the controller and views automatically.

dotnet aspnet-codegenerator --project . controller -name DummyController -m Dummy -dc ApplicationDbContext

ASP.NET Core code generator parameters:

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| -m | The name of the model |
| -dc | The DbContext class to use. |
| -udl | Use the default layout |
| -outDir | The relative output folder path to create the views |
| --referenceScriptLibraries | Adds \_ValidationScriptsPartial to Edit and Create pages |

\*Reference URL: <https://docs.microsoft.com/en-us/aspnet/core/tutorials/razor-pages/model?view=aspnetcore-2.2&tabs=visual-studio-code>

# Bibliography

Microsoft. (2015, 07 20). *C# Coding Conventions (C# Programming Guide)*. Retrieved 04 14, 2019, from Microsoft.com: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions