

MVC

- [Table of contents](#)
- [General](#)
- [Naming Conventions](#)
- [Model/ Domain Model](#)
- [ViewModel](#)
- [View](#)
- [Controller](#)
- [Routing](#)
- [Attributes](#)
- [Extensions and Tools](#)
- [API Conventions](#)

Table of contents

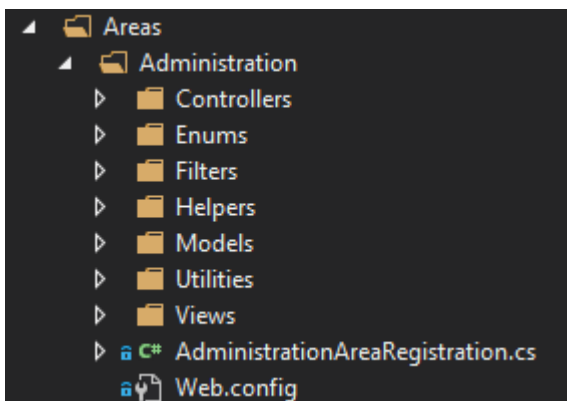
1. General
2. Naming Conventions
3. Models / Domain Models
4. View Models
5. View
6. Controllers
7. Routing
8. Attributes
9. Extensions and Tools

General

For a general guide on ASP.NET MVC, click on the following image



- Each functional domain being implemented must be added to its own Area e.g Administration
- The folder structure to use when creating a new MVC area is



- Each area must have the folders Controllers, Models and Views, the other folders are optional.
- The function of each folder is:
 - **Controller:** An interface between Model and View components to process all the business logic and incoming requests, manipulate data using the Model component and interact with the Views to render the final output.
 - **Views:** The View component is used for all the UI logic of the application.
 - **Models:** The Model component corresponds to all the data-related logic that the user works with. This can represent either the data that is being transferred between the View and Controller components or any other business logic-related data.
 - **Filters:** The action filters that are attributes that can apply to a controller action or an entire controller that modifies the way in which the action is executed.

- **Utilities:** The classes that receives a object and must do certain data modification or calculation custom to your domain are added here. These classes do not have access to the data domain and must be able to complete its action with the data provided e.g. Calculating the strength of a password
- **Helpers:** Similar to utilities, helpers perform certain actions form or on data received. Helpers do have access to the data domain and can fetch additional data or perform data modification depending on the results of its calculation
- **Enums:** Any enums only used in this domain are added here. It can be used for drop down population etc.
- Each area must have a web.config
 - It must define the namespace to include in the areas views
 - It must define the handlers and other configuration specific to that area

TODO : what happens in web.config? Add Example zip folder to General page

<https://signature.signifyhr.co.za/books/tools/page/v8---mvc-development>

Naming Conventions

The following naming convention must be followed when creating any of the MVC pattern sections. All names must be created using Pascal-Case.

Model

- The name is always a singularized representation of the database entity e.g. for the data base entity prsEmployees it must be prsEmployee
- When a model is required that is made up of SQL procedures for a specific domain of data, the name must represent the domain of data e.g. BadgesAndPoints is a model of different SQL objects returning data regarding badges and/ or points and nothing else.

Controller

- Every controller must be suffixed with the word Controller e.g HomeController
- Every controller must be placed within the ~/Controllers folder of its area

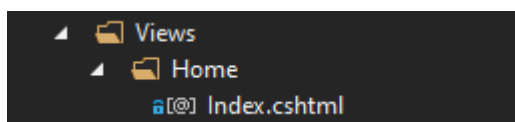
Attributes

All custom filter attributes must be named according to its function e.g. **BypassSession** Authorisation

View

- The view's name must represent the type of data it displays e.g. CreateEdit.cshtml
- The partialview's name must represent the type of data it displays and must always start with a underscore e.g. _OrderDetail.cshtml
- Every view must be placed within the ~/Views folder of its area matching the controllers name

e.g. for the HomeController



- When the view is used by multiple controller it must be placed within the ~/Views/Shared folder of that area

View Models

- The name must always end with the postfix **ViewModel**
- The name can start with the action it is called from e.g. LoginViewModel
- The name can start with a descriptive name defining the type of data it encapsulates e.g. EmployeeDetailViewModel

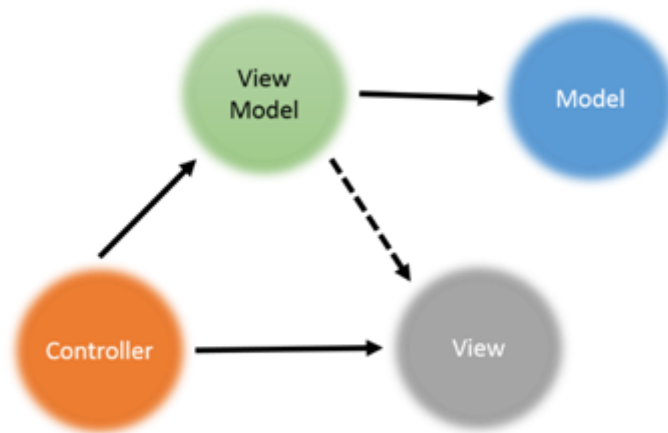
Routing

The registrations file for a areas route must be named {AreaName}AreaRegistration e.g LearningStoreRegistration.cs

Model/ Domain Model

[Go To Naming Conventions](#)

For information regarding models, click the images below

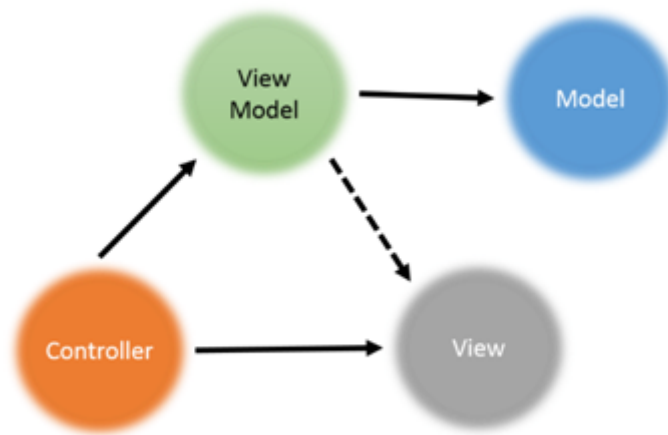


- Contain properties to represent specific data e.g EF Entities or SQL objects
- Contain business logic (e.g. validation rules) to ensure the represented data fulfills the design requirements
- May contain code for manipulating data. For example, a `SearchForm` model, besides representing the search input data, may contain a `search` method to implement the actual search. e.g. [Domain Conventions](#)
- The model may not contain any logic that is strictly required by the front end. This type of logic must rather be implemented in the [View Model](#) e.g. *build an html control based on business rules should be done the ViewModel instead of the Model.*
- Should not use `$_GET`, `$_POST`, or other similar variables that are directly tied to the end-user request.
- Should avoid embedding HTML or other presentational code.

ViewModel

[Go To Naming Conventions](#)

For information regarding view models, click the images below



- The class to group one or more models.
- It applies business logic on the results set from the models

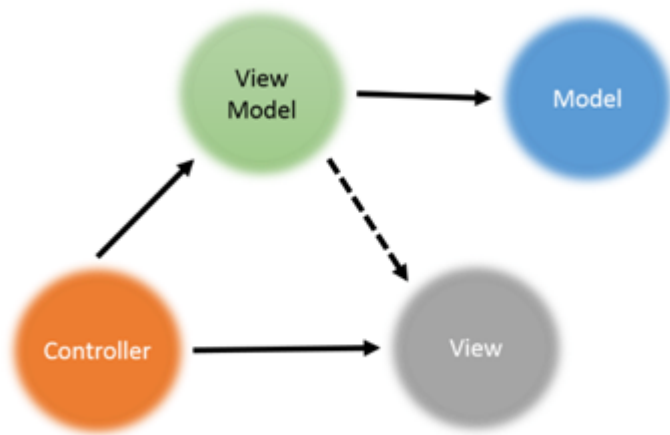
TODO : properties defined and properties inherited, private set properties, constructors

Example PathwayViewModel

View

[Go To Naming Conventions](#)

For information regarding razor views, click the images below



Responsible for presenting viewmodels, viewbags and other data structures in the format of the end user requirements

- Should mainly contain presentational code, such as HTML, to format and render data
- No code that performs explicit DB queries may be added to the view and must rather be placed in the view model.
- Should avoid direct access to `$_GET`, `$_POST`, or other similar variables that represent the end user request.
- May access properties and methods of controllers and view models directly only for presentational reasons

Reusing Views

Layouts

Common presentational areas (e.g. page header, footer) can be put in a layout view.

TODO - add link to Layout page in The Standards.

Partial Views

Views without a layout that reuse fragments of presentational code e.g. used in modals.

- Must be used if only parts of a page with data must be refreshed e.g. List with paging
- Represents code that can be used by multiple views in the Area.
- A partial must always start with an **underscore**.
- If the partial is only used inside the specific controller it must be in that specific folder. E.g. Details for LeaveGroupController will be in the LeaveGroup folder with the name: **_Details.cshtml**.
- A partial that shows items linked to the specific entity, must have the name of the collection linked to the entity, and must be in the folder of the entity. E.g. Leave Types linked to Leave Groups will be: **_LeaveTypes.cshtml** and it will be in the LeaveGroup folder.
- A partial that contains a list of items that can be linked to the entity must follow the convention in 3. and must follow these naming conventions:
 1. If it is a multiple link list (with the delink, link and in use buttons): **_LeaveTypesLinkMultiple.cshtml**
 2. If it is a picker list that populates a textbox on the parent page (you can only link one item at a time): **_LeaveTypesSelectSingle.cshtml**
- A partial with a list of items that is generic and shared over multiple controllers must be in the Shared folder.
 1. If it is shared only in the current Area, it must be in the Area's shared folder: **App/Areas/Module/Views/Shared**
 2. If it is shared over multiple areas it should be in the global shared folder: **App/Views/Shared**
- A partial that is used for navigation should follow these naming conventions: **_LeaveTypesSideNav.cshtml** is a partial for Leave Groups that is used to navigate between Leave Types linked to a Leave Group
- Exceptions:
 1. When using a partial in a one view only but for AJAX posting i.e. an action returns **PartialViewResult** in controller
- If the partial is shared between views, the javascript needs to reside on the partial itself.
- Javascript shared among multiple partials with the same parent view should reside on the parent view.

Helpers Classes

In views we often need some code snippets to do tiny tasks such as formatting data or generating HTML tags.

- Small segments can be placed directly in the view, but larger segments must be moved to a class file
- An example of a helper and its use in a view

The method in the helper class

```

@helper StepLink(string name, string icon, string actionText, string actionIcon, string onclick = null, string href = null, string target = null) {
    // Must have action text as well as an onclick or an href
    var clickable = !String.IsNullOrEmpty(actionText) && (!String.IsNullOrEmpty onclick) || !String.IsNullOrEmpty(href);

    <div class="step-link">
        <div class="step-link_name">
            @System.Web.Mvc.MvcHtmlString.Create(name)
        </div>

        <a class="step-link_icon" onclick="@onclick" href="@href" target="@target" @if (!clickable) {<text>style="cursor: default"</text>}>
            <i class="fa fa-@icon"></i>
        </a>

        @if (clickable)
        {
            <a class="step-link_action primary-bg" onclick="@onclick" href="@href" target="@target">
                <i class="fa fa-@actionIcon"></i> <span>@actionText</span>
            </a>
        }
    </div>
}

```

The implementation of the class and method in the view

```

<div class="text-center">
    @PathwayHelpers.StepLink(Model.QuestionMarkAssessment.AssessmentId, "question-circle", "Launch", "paper-plane", Model.Url)
</div>

```

TODO: add examples from v8 mvc page

Ensure list data bound during post (use for instead of for each)

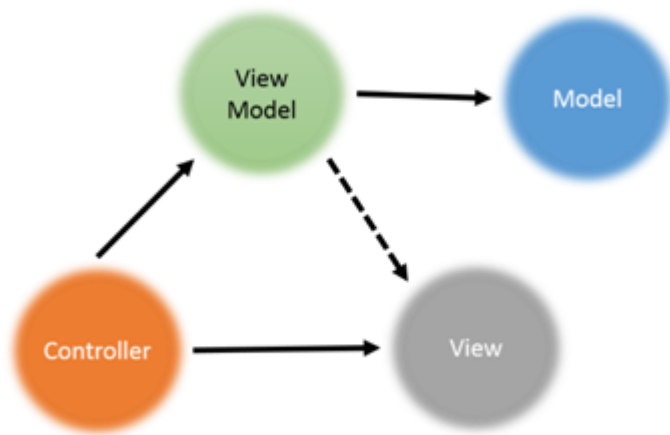
Difference between Html.Form and Ajax.Form

Use of hidden field variables and where they should be placed in a view

Controller

[Go To Naming Conventions](#)

For information regarding Controllers, click the images below



- Each controller must implement the IController interface directly or indirectly, this must be done using the BaseController

```
namespace SignifyHR.Areas.Portal.Controllers
{
    public class HomeController : BaseController
    {
        [UserTranslationFilter]
        public ActionResult Index(PortalTabs? activeTab)
        {
            var model = new HomeViewModel(SessionHandler, activeTab)
            {
                Title = TranslationResources.PageHeadingsLmsPortal,
            };

            return View(model);
        }
    }
}
```

- TODO from v8 - MVC Examples

<https://signature.signifyhr.co.za/books/tools/page/v8---mvc-development/edit>

When to use ActionResult and JsonResult

Link to exception

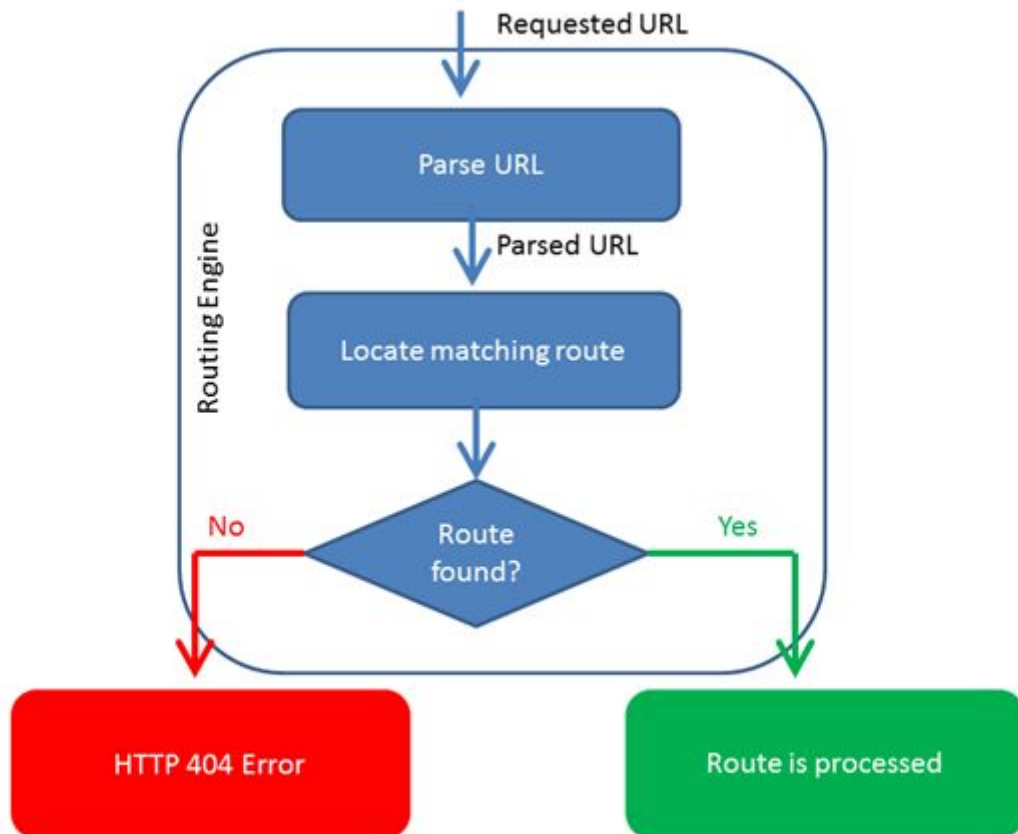
AllowGet

Follow C# standards

Routing

[Go To Naming Conventions](#)

For a guide regarding routing, click on the image below



Area specific routing is implemented in the Area Registration cs file and can have custom routing as required by the area e.g. LearningStoreAreaRegistration.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace SignifyHR.Areas.LearningStore
{
    public class LearningStoreAreaRegistration : AreaRegistration
```

```
{
    public override string AreaName
    {
        get
        {
            return "LearningStore";
        }
    }

    public override void RegisterArea(AreaRegistrationContext context)
    {
        context.MapRoute(
            name: "LearningStore.Pages",
            url: "LearningStore/{id}/{slug}",
            defaults: new { controller = "Page", action = "Index", slug = UrlParameter.Optional },
            constraints: new { id = @"\d+" },
            namespaces: new[] { "SignifyHR.Areas.LearningStore.Controllers" }
        );

        context.MapRoute(
            name: "LearningStore.DEFAULT",
            url: "LearningStore/{controller}/{action}",
            defaults: new { controller = "Home", action = "Index" },
            namespaces: new[] { "SignifyHR.Areas.LearningStore.Controllers" }
        );
    }
}
```

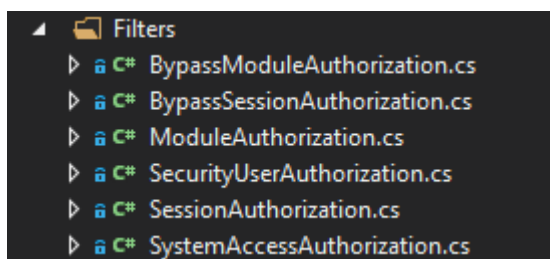
Attributes

[Go To Naming Conventions](#)

The ASP.NET MVC framework supports four different types of filters executed in the following order:

1. **Authorisation filters** – Implements the `IAuthorizationFilter` attribute.
2. **Action filters** – Implements the `IActionFilter` attribute.
3. **Result filters** – Implements the `IResultFilter` attribute.
4. **Exception filters** – Implements the `IExceptionFilter` attribute.

The common authorisation filters that is used in Signify are



- **BypassModuleAuthorisation:** The MVC authorization filter that enables one to by-pass module authentication.
- **BypassSessionAuthorisation:** The MVC authorization filter that enables one to by-pass session authentication.
- **ModuleAuthorisation:** Extended authorisation that determines if the user consists of a session.
- **SecurityUserAuthorisation:** Authorize the user against roles.
- **Session Authorisation:** Extended authorisation that determines if the user consists of a session, implemented in the base controller
- **SystemAccessAuthorisation:** Authorize the user on access to the module and the active status of the module.

Example of use

Applying a filter on a single method in the controller


```

[AllowAnonymous]
0 references | 0 changes | 0 authors, 0 changes
public ActionResult Confirmation(Guid token)
{
    try
    {
        var decryptedToken = sysAccessRequest.TokenDataWrapper.Decrypt(token);

        if (decryptedToken != null)
        {
            return View(new AccessRequestConfirmationViewModel(decryptedToken.sysUserRegistrationId, decryptedToken.sysUserRegistrationId));
        }
        else
        {
            throw new Exception("The request token could not be deserialized.");
        }
    }
    catch (Exception ex)
    {
        ErrorUtilities.LogException(ex);
        throw new Exception("The request token could not be decrypted or deserialized.", ex.InnerException);
    }
}

```

Applying a filter on the controller as a whole

```

namespace SignifyHR.Areas.Security.Controllers
{
    [AllowAnonymous]
    [AllowAnonymous]
    0 references | Martinique Lourens, 62 days ago | 7 authors, 11 changes | 2 work items
    public class AccountController : SignifyHR.Controllers.BaseController
    {

```

Extensions and Tools

Logging

Using NLog

```
public class HomeViewModel : PortalMasterModel
{
    private static readonly Logger _logger = LogManager.GetCurrentClassLogger();
    private static readonly Logger _lmsLogger = LogManager.GetLogger("LMS");

    public HomeViewModel()
    {
        // Using the class name as the name of the logger
        _logger.Info("Initializing HomeViewModel");

        // Using a new logger instance with a property automatically added
        _lmsLogger.WithProperty("ActiveTab", activeTab).Info("Initializing HomeViewModel");

        // Setting a property permanently on a logger
        _logger.SetProperty("Key", "Value");
        _logger.Info("Testing EventProperties");

        // Adding properties automatically in the message
        _logger.Info("Testing dynamic EventProperty Key={PropertyName}", "PropertyValue");

        // Logging an exception
        _logger.WarnException("Exception", new System.Exception("Example Exception"));
    }
}
```

Example of log levels

Level	Message	Logger	Properties	Callsite	Exception
-------	---------	--------	------------	----------	-----------

Info	Initializing HomeViewModel	SignifyHR.Areas.Portal.ViewModels.HomeViewModel		SignifyHR.Areas.Portal.ViewModels.HomeViewModel..ctor	
Info	Initializing HomeViewModel	LMS	ActiveTab=TrainingRequirements	SignifyHR.Areas.Portal.ViewModels.HomeViewModel..ctor	
Info	Testing EventProperties	SignifyHR.Areas.Portal.ViewModels.HomeViewModel	Key=Value	SignifyHR.Areas.Portal.ViewModels.HomeViewModel..ctor	
Info	Testing dynamic EventProperty Key="PropertyValue"	SignifyHR.Areas.Portal.ViewModels.HomeViewModel	PropertyName=PropertyValue Key=Value	SignifyHR.Areas.Portal.ViewModels.HomeViewModel..ctor	
Warn	Exception	SignifyHR.Areas.Portal.ViewModels.HomeViewModel	Key=Value	SignifyHR.Areas.Portal.ViewModels.HomeViewModel..ctor	System.Exception: Example Exception

Log Levels

Level	Typical Use
Fatal	Something bad happened; application is going down
Error	Something failed; application may or may not continue
Warn	Something unexpected; application will continue
Info	Normal behavior like mail sent, user updated profile etc.
Debug	For debugging; executed query, user authenticated, session expired
Trace	For trace debugging; begin method X, end method X

Rules

A rule is a logger element with the following attributes:

- name – logger name filter - may include wildcard characters (* and ?)
- minlevel – minimal level to log

- maxlevel – maximum level to log
- level – single level to log
- levels – comma separated list of levels to log
- writeTo – comma separated list of targets to write to
- final – no rules are processed after a final rule matches
- enabled – set to false to disable the rule without deleting it
- ruleName – rule identifier to allow rule lookup with Configuration.FindRuleByName and Configuration.RemoveRuleByName.

NLog.config

```
!--  
    Create a rule to only log entries from the LMS HomeViewModel logger,  
    using the logDatabase target, and applicable on log levels Info and above  
-->  
<logger name="SignifyHR.Areas.Portal.ViewModels.HomeViewModel" minlevel="Info" writeTo="logDatabase"  
enabled="true" final="true" />
```

API Conventions

Propagate errors. If an error occurred, return a 500. The status code should tell you what happened, not the result of the call itself. Follow the DefaultApiConventions, which state that the action should return void and produce a 200, 404, or 400 status. In MVC these rules don't have to be followed as strictly, but with the APIs it is non-negotiable for standardisation.

```
[ProducesResponseType(StatusCodes.Status200OK)]  
[ProducesResponseType(StatusCodes.Status404NotFound)]  
[ProducesResponseType(StatusCodes.Status400BadRequest)]  
[ProducesDefaultResponseType]  
[ApiConventionNameMatch(ApiConventionNameMatchBehavior.Prefix)]  
public static void Delete(  
[ApiConventionNameMatch(ApiConventionNameMatchBehavior.Suffix)]  
[ApiConventionTypeMatch(ApiConventionTypeMatchBehavior.Any)]  
object id) { }
```

<https://github.com/dotnet/aspnetcore/blob/master/src/Mvc/Mvc.Core/src/DefaultApiConventions.cs>