You can expand the locations where the view engine looks for views by implementing a view location expander. Here is some sample code to demonstrate the approach:

public class ViewLocationExpander: IViewLocationExpander {

/// <summary>

/// Used to specify the locations that the view engine should search to

/// locate views.

/// </summary>

/// <param name="context"></param>

/// <param name="viewLocations"></param>

/// <returns></returns>

public IEnumerable<string> ExpandViewLocations(ViewLocationExpanderContext context, IEnumerable<string> viewLocations) {

//{2} is area, {1} is controller,{0} is the action

string[] locations = new string[] { "/Views/{2}/{1}/{0}.cshtml"};

return locations.Union(viewLocations); //Add mvc default locations after ours

}

public void PopulateValues(ViewLocationExpanderContext context) {

context.Values["customviewlocation"] = nameof(ViewLocationExpander);

}

}

Then in the ConfigureServices(IServiceCollection services) method in the startup.cs file add the following code to register it with the IoC container. Do this right after services.AddMvc();

services.Configure<RazorViewEngineOptions>(options => {

options.ViewLocationExpanders.Add(new ViewLocationExpander());

});

Now you have a way to add any custom directory structure you want to the list of places the view engine looks for views, and partial views. Just add it to the locations string[]. Also, you can place a \_ViewImports.cshtml file in the same directory or any parent directory and it will be found and merged with your views located in this new directory structure.

**Update:**  
One nice thing about this approach is that it provides more flexibility then the approach later introduced in ASP.NET Core 2 (Thanks @BrianMacKay for documenting the new approach). So for example this ViewLocationExpander approach allows for not only specifying a hierarchy of paths to search for views and areas but also for layouts and view components. Also you have access to the full ActionContext to determine what an appropriate route might be. This provides alot of flexibility and power. So for example if you wanted to determine the appropriate view location by evaluating the path of the current request, you can get access to the path of the current request via context.ActionContext.HttpContext.Request.Path

This is a good solution however, this does not solve the problem of finding a view that has a route attribute on the action or controller. The View method still appears to use the name of the controller and not the route name to locate the view. – [Xipooo](https://stackoverflow.com/users/3207465/xipooo" \o "776 reputation) [Jun 14 '17 at 18:13](https://stackoverflow.com/questions/36747293/how-to-specify-the-view-location-in-asp-net-core-mvc-when-using-custom-locations#comment76094335_36772778)

* @Xipooo, good point. The example I provided is a good start but to use the route you can set locations array to include /Views + context.ActionContext.HttpContext.Request.Path + either index.cshtml or .cshtml. – [Ron C](https://stackoverflow.com/users/1415614/ron-c) [Jun 23 '17 at 14:49](https://stackoverflow.com/questions/36747293/how-to-specify-the-view-location-in-asp-net-core-mvc-when-using-custom-locations#comment76431014_36772778)
* This allowed me to use a view that was added to the bin folder via a .net standard 1.6. Thank you great solution. – [DeadlyChambers](https://stackoverflow.com/users/1248536/deadlychambers" \o "3,087 reputation)