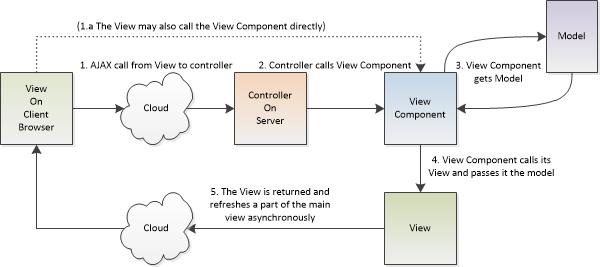
## Introduction

This document will provide a step by step work flow for working with View Components, those partial views that we work with in ASP.NET Core MVC. View Components allow us to refresh just a part of the page asynchronously.

They are developed in the same way a Controller and a View are developed together. A Controller calls a View and a View Component calls a View. Both the Controller and the View Component may get the Model to pass to the View before calling it.

A View Component may be called directly from View code or it may be called from a Controller method. In both cases data may be passed to the View Component such as an id value.



In this work flow, we will assume the following use case:

***“As a property renter, I want to select a property type from a drop-down list and view rental properties of that type so that I can find a place to rent more efficiently.”***

Technically, the whole page should not be refreshed and only the display of filtered rental properties needs to be refreshed. A View Component can also be reused on other pages (views) of the web application.

## THE MAIN VIEW

Using the same application created in the Entity Framework Core Code First document, we will create a method called Search in the *RentalsController* class that will return a View called Search. The code for the controller class and cshtml view file follows:

Add the following method to the Rentals controller class code if it doesn’t already exist:

public IActionResult Search()

{

//need to get the property types collection and assign to the ViewBag

//Working with a SelectListItem List collection so we can insert a

//SelectListItem for selecting all rentals based on an id of 0

var types = PropertyTypeManager.GetAsKeyValuePairs();

var styles = new SelectList(types, "Value", "Text");

//This list item is inserted at the first position in the collection and

//has a value of 0. This value of 0 is tested in the ViewComponent. If

//the property type id is 0 we want all rentals returned otherwise we will

//use the id in the query.

var list = styles.ToList();

list.Insert(0, new SelectListItem

{

Text = "All Styles",

Value = "0"

});

//Pass the SelectListItem collection to the view through the ViewBag

//ViewBag is strongly-typed

ViewBag.PropertyTypes = list;

//return the view

return View();

}

Add the Search view. It has two div elements. The first div contains a Select element populated with *SelectListItem* objects with the values and text coming from *PropertyType* objects. The View code follows:

<h2>Search Rental Properties by Property Type</h2>

<div>

<label>Select Property Type:</label>

<**select** id="uxPropertyTypes" **asp-items**="ViewBag.PropertyTypes"></**select**>

</div>

<div id="uxDisplay">

<!-- The View Component is invoked asynchronously with 0 as the id when

displayed for first time and this is step 1.a from the image above-->

@await Component.InvokeAsync("RentalsByType", 0)

</div>

@section Scripts{

<script>

$(document).ready(function () {

//When property type selection changes, we get the selected value

//to use in the asynchronous ajax call

$("#uxPropertyTypes").change(function () {

var propertyTypeId = $("#uxPropertyTypes").val();

//AJAX call is a GET to the controller method that calls the

//view component--the id comes from the property type selection.

//The done method handles the callback asynchronously. This is step 1.

$.ajax({

method: 'GET',

url: '/Rentals/GetPropertiesByType',

data: { id: propertyTypeId }

}).done(function (result, statusText, xhdr) {

$("#uxDisplay").html(result);

});

});

});

</script>

}

Note that the url in the ajax method above is calling the *GetPropertiesByType* method of the *Rentals* controller (This is #1 in the flow diagram on page 1).

## THE CONTROLLER CODE

public IActionResult GetPropertiesByType(int id)

{

return ViewComponent("RentalsByType", id);

}

This method requires the id of the Property Type. The Controller then calls the *ViewComponent* called *RentalsByType* and passes it the id value (this is #2 in the flow diagram).

## THE VIEW COMPONENT CODE

Add the View Component called *RentalsByType* to the MVC project (in Models folder or a new folder called Components). It inherits from *ViewComponent*. Use following code:

public class RentalsByType : ViewComponent

{

public async Task<IViewComponentResult> InvokeAsync(int id)

{

//Get the model asynchronously and pass to the view (ViewModel)

//Declare a local variable to store rental properties

List<RentalProperty> properties = null;

//OK, this is a "hack" because I want to test if id is 0 (select all)

if (id == 0)

{

//return all rentals if id is 0...

properties = RentalsManager.GetAll();

}

else

{

//...otherwise we will use the id in the query

properties = RentalsManager.GetAllByPropertyType(id);

}

//Now we can transform whatever rental collection we have to a

//collection of RentalsViewModel objects to pass to the view

var rentals = properties.

Select(rp => new RentalsViewModel {

Address = rp.Address,

City = rp.City,

Id = rp.Id,

OwnerName = rp.Owner.Name,

PostalCode = rp.PostalCode,

PropertyStyle = rp.PropertyType.Style,

Province = rp.Province,

RentAmount = rp.Rent.ToString()

}).ToList();

//model given to the view

return View(rentals);

}

}

**THE COMPONENT’S VIEW CODE**

Next we add the View called Default.cshtml to the project. Views for View Components have to be in a specific place in order to be found at runtime. There is only one view per View Component. One of the following folder structures must be used:

1. Views/Shared/Components/<View Component Name>/<View Name>
2. Views/<Controller>/Components/<View Component Name>/<View Name>

Our folder structure following option 1: Views/Shared/Components/RentalsByType/Default.cshtml

You can use the view list template with the RentalViewModel as the model to create the view and delete what you don’t need or use the code block below:

@model System.Collections.Generic.IEnumerable<CPRG214.Rentals.App.Models.RentalsViewModel>

@{

ViewData["Title"] = "Default";

}

<h3>Rental Properties</h3>

<!-- Table displaying the rentals by the selected property type-->

<table class="table">

<tr>

<th>Address</th>

<th>City</th>

<th>Province</th>

<th>Postal Code</th>

<th>Rental Amount</th>

<th>Property Style</th>

<th>Owner Name</th>

</tr>

@foreach (var rental in Model)

{

<tr>

<td>@rental.Address</td>

<td>@rental.City</td>

<td>@rental.Province</td>

<td>@rental.PostalCode</td>

<td>$@rental.RentAmount</td>

<td>@rental.PropertyStyle</td>

<td>@rental.OwnerName</td>

</tr>

}

</table>

<https://docs.microsoft.com/en-us/aspnet/core/mvc/views/view-components?view=aspnetcore-3.1>