COMP 2139
How to Transfer Data from Controllers

Agenda

- Describe some important interfaces and classes of the ActionResult hierarchy
- Distinguish between the ViewBag and ViewData properties
- Describe the use of a view model to transfer data from a controller to a view
- Describe how one action method can redirect to another action method
- Decribe the use of the PRG (Post-Redirect-Get) pattern to prevent resubmission of POST data
- Disintguish between the ViewData and TempData properties
- Describe the purpose of the Keep() and Peek() methods of the TempData class.

The ActionResult Hierarchy

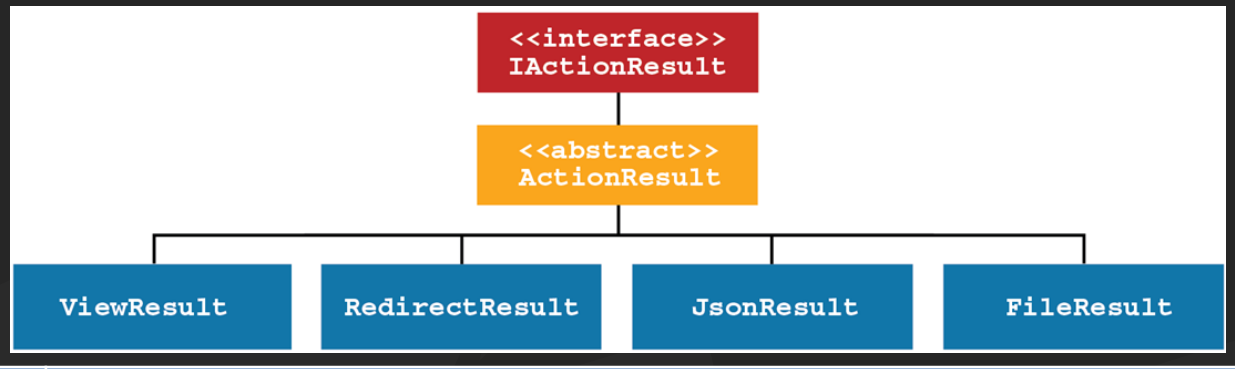
ActionResult

A URL for a full list of ActionResult subtypes

https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.actionresult

- Within a controller, an action method can return any type of ActionResult object. The ActionResult class is an abstract class that implements the IActionResult interface.
- Since the ActionResult class has many subtypes, an action method can return many different types of result objects.

The ActionResult Hierarchy



Class	Description
ViewResult	Renders a specified view as HTML and sends it to the browser
RedirectResult	Performs an HTTP redirection to the specified Url
RedirectToActionResult	Performs an HTTP redirection to a URL that's created by the routing system using the specified controller and action data
JsonResult	Serializes an object to JSON and sends the JSON to the browser
FileResult	Returns a file to the browser
StatusCodeResult	Sends an HTTP response with a status code to the browser
ContentResult	Returns plain text to the browser
EmptyResult	Returns an empty response to the browser

How to Return ActionResult objects

Some methods of the Controller class

Method	Description
View()	ViewResult object
Redirect()	RedirectResult object
RedirectToAction()	RedirectToActionResult object
File()	FileResult object
Json()	JsonResult object

Method	Description
View()	Renders the **default** view for that controller and action method
View(model)	Transfers a model object to the **default** view and renders that view
View(name)	Renders the specified view. This method searches for the specified view in the folder for the current controller. The it searches the the Views/Shared
View(name, model)	Transfers a model object to the specified view and renders that view.

Examples: How to Return ActionResult objects

An action method that returns a ViewResult Object

```
public ViewResult List() {
    var names = new List<string> { "Grace", "Ada", "Charles" };
    return View(names);
}
```

An action method that a RedirectToActionResult object

```
public RedirectToActionResult Index() =>
    RedirectToAction("List");
```

An action method that may return different types of result objects

```
[HttpPost]
public IActionResult Edit(string id) {
    if (ModelState.IsValid)
        return RedirectToAction("List");
    else
        return View((object)id);   // cast string model to object
}
```

How to work with View Models

View Models

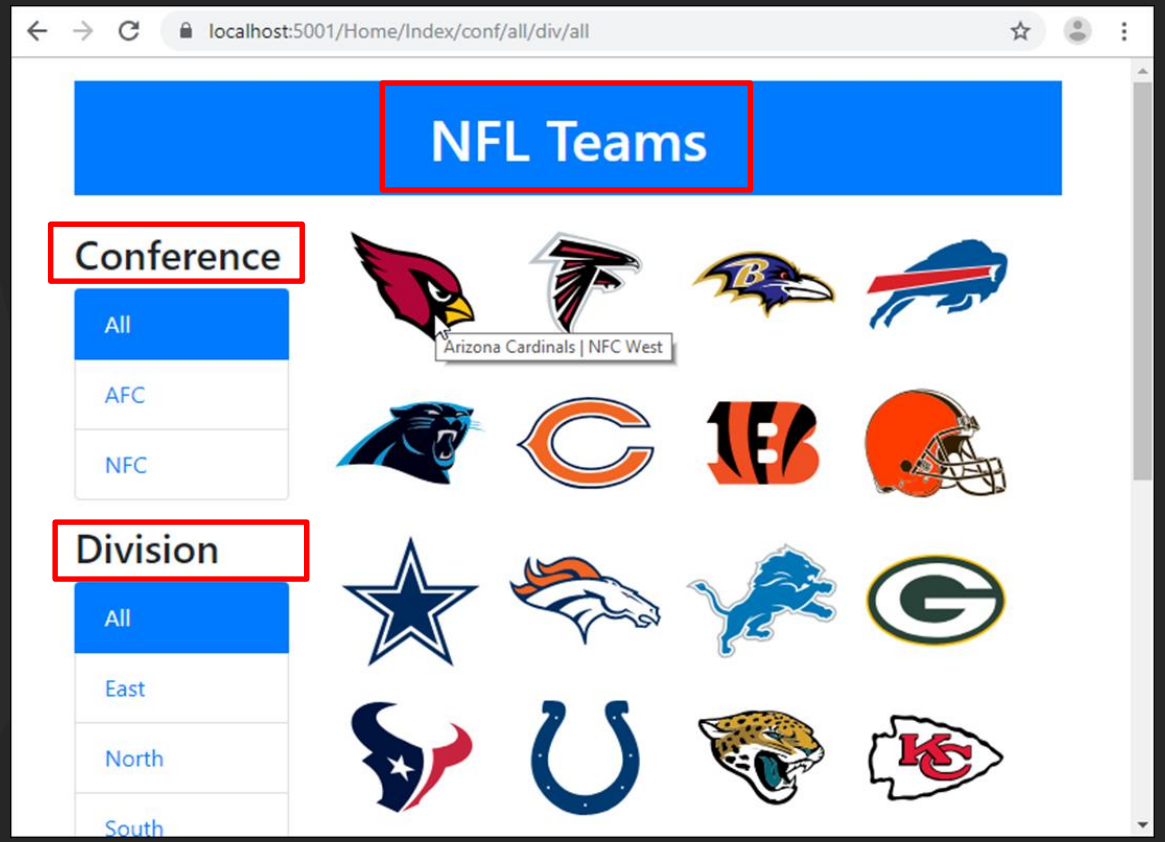
How to work with View Models

- The Applications that we have worked with so far, have used the View() method to transfer a single entity object or collection of entity objects to a view.
- **However, many times, the data needed by a view doesn't match the data in an entity model.**
- To remedy this problem, we create **View Models**
- A **View Model** is a regular C# class that defines a model of the data that's needed by a view
- By conventions, the name of a view model class ends with a suffix of "**ViewModel**" but this isn't required
- Most view models only provide data. However, a view model can also contain simple methods that help the view display that data.
- **Its generally considered a best pratice to use a view model to transfer data to a view.**

NFL Teams Application - REVISITED

Using View Models

- The model object for the Index view in the NFL teams application, is a collections of **Team** objects.
- However the view also requires a collection of **Conference** objects and **Division** objects
- It also requires the IDs of the active Conference and Division.



NFL Teams Application Revisited
(*Ch08bNFLTeams*)

TeamListViewModel

ViewModel is C# class that holds all the data that a specific view requires.

```
public class TeamViewModel
{
    7 references
    public Team Team { get; set; }
    7 references
    public string ActiveConf { get; set; } = "all";
    7 references
    public string ActiveDiv { get; set; } = "all";
}
```

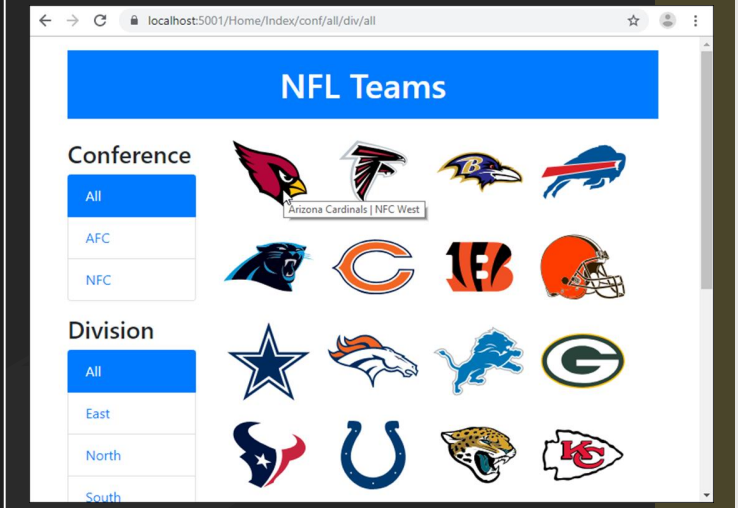
```
public class TeamListViewModel : TeamViewModel
{
    2 references
    public List<Team> Teams { get; set; }

    // use full properties for Conferences and Divisions
    // so can add 'All' item at beginning
    private List<Conference> conferences;
    2 references
    public List<Conference> Conferences {
        get => conferences;
        set {
            conferences = value;
            conferences.Insert(0,
                new Conference { ConferenceID = "all", Name = "All" });
        }
    }

    private List<Division> divisions;
    2 references
    public List<Division> Divisions {
        get => divisions;
        set {
            divisions = value;
            divisions.Insert(0,
                new Division { DivisionID = "all", Name = "All" });
        }
    }

    // methods to help view determine active link
    1 reference
    public string CheckActiveConf(string c) =>
        c.ToLower() == ActiveConf.ToLower() ? "active" : "";

    1 reference
    public string CheckActiveDiv(string d) =>
        d.ToLower() == ActiveDiv.ToLower() ? "active" : "";
}
```



The Updated Index() action

```
0 references
public IActionResult Index(string activeConf = "all",
                           string activeDiv = "all")
{
    var data = new TeamListViewModel
    {
        ActiveConf = activeConf,
        ActiveDiv = activeDiv,
        Conferences = context.Conferences.ToList(),
        Divisions = context.Divisions.ToList()
    };

    IQueryable<Team> query = context.Teams;
    if (activeConf != "all")
        query = query.Where(
            t => t.Conference.ConferenceID.ToLower() == activeConf.ToLower());
    if (activeDiv != "all")
        query = query.Where(
            t => t.Division.DivisionID.ToLower() == activeDiv.ToLower());
    data.Teams = query.ToList();

    return View(data);
}
```

The Updated Home/Index view

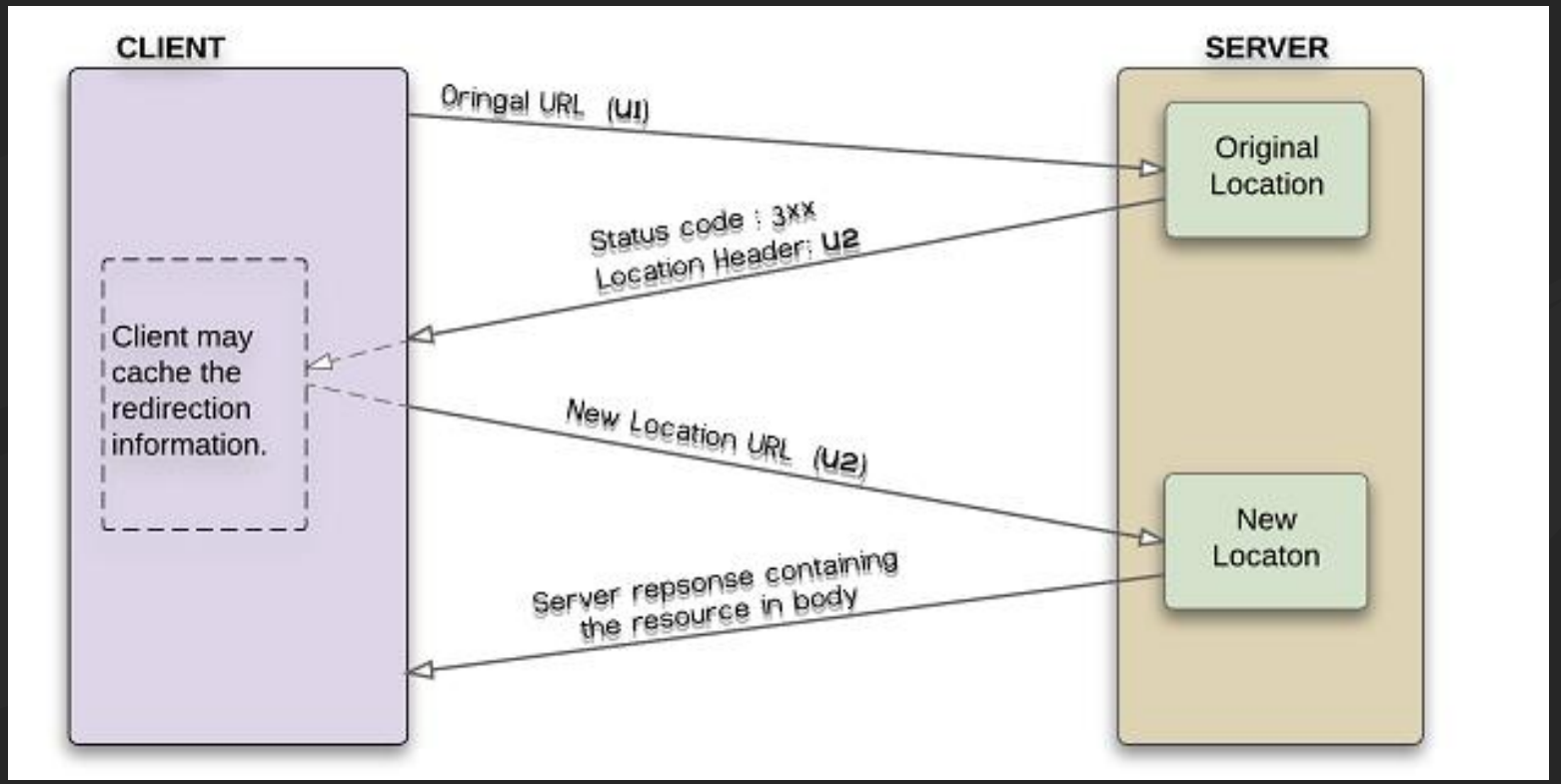
```
@model TeamListViewModel
@{
    ViewData["Title"] = "NFL Teams";
}
<div class="row">
    <div class="col-sm-3">
        <h3 class="mt-3">Conference</h3>
        <div class="list-group">
            @foreach (Conference conf in Model.Conferences) {
                <a asp-action="Index"
                    asp-route-activeConf="@conf.ConferenceID"
                    asp-route-activeDiv="@Model.ActiveDiv"
                    class="list-group-item
                        @Model.CheckActiveConf(conf.ConferenceID)">@conf.Name</a>
            }
        </div>
        <h3 class="mt-3">Division</h3>
        <div class="list-group">
            @foreach (Division div in Model.Divisions) {
                <a asp-action="Index"
                    asp-route-activeConf="@Model.ActiveConf"
                    asp-route-activeDiv="@div.DivisionID"
                    class="list-group-item
                        @Model.CheckActiveDiv(div.DivisionID)">@div.Name</a>
            }
        </div>
    </div>
```

```
    <div class="col-sm-9">
        <ul class="list-inline">
            @foreach (Team team in Model.Teams)
            {
                <li class="list-inline-item">
                    <form asp-action="Details" method="post">
                        <button type="submit" class="bg-white border-0">
                            <img src="~/images/@team.LogoImage" alt="@team.Name"
                                title="@team.Name | @team.Conference.Name @team.Division.Name" />
                        </button>

                        <input type="hidden" asp-for="@team.TeamID" />
                        <input type="hidden" asp-for="ActiveConf" />
                        <input type="hidden" asp-for="ActiveDiv" />
                    </form>
                </li>
            }
        </ul>
    </div>
</div>
```

HTTP Redirection

HTTP Redirect



HTTP Redirect…

Two of the HTTP Status codes for Redirection

302 Found → instruct the client browser to make a GET request to another URL

301 Moved Permanently → instruct the client browser to make a GET request to another URL for this and all future requests

The ActionResult return subtypes for redirection

Subtype	302 Found method	301 Moved Permanently method
RedirectResult	Redirect()	RedirectPermanent()
LocalRedirectResult	LocalRedirect()	LocalRedirectPermanent()
RedirectToActionResult	RedirectToAction()	RedirectToActionPermanent()
RedirectToRouteResult	RedirectToRoute()	RedirectToRoutePermanent()

How to know which return subtype to use for redirection

Subtype	Use when…
`RedirectResult`	Redirecting to an external URL, such as https://google.com.
`LocalRedirectResult`	Making sure you redirect to a URL within the current app.
`RedirectToActionResult`	Redirecting to an action method within the current app.
`RedirectToRouteResult`	Redirecting within the current app by using a named route.

Some of the overloads available for the **RedirectToAction**() method

Arguments	Redirect to…
`(a)`	The specified action method in the current controller.
`(a, c)`	The specified action method in the specified controller.
`(a, routes)`	The specified action method in the current controller with route parameters.
`(a, c, routes)`	The specified action method in the specified controller with route parameters.

Code that redirects to another method

The List() action method in the current controller

```
public RedirectToActionResult Index() => RedirectToAction("List");
```

The List() action method in the Team controller

```
public RedirectToActionResult Index() => RedirectToAction("List", "Team");
```

The Details() action method in the current controller with a parameter

```
public RedirectToActionResult Index(string id) =>
                                    RedirectToAction("Details", new { ID = id });
```

Code that redirects to another method…

Shortcut when variable name and route segment name match

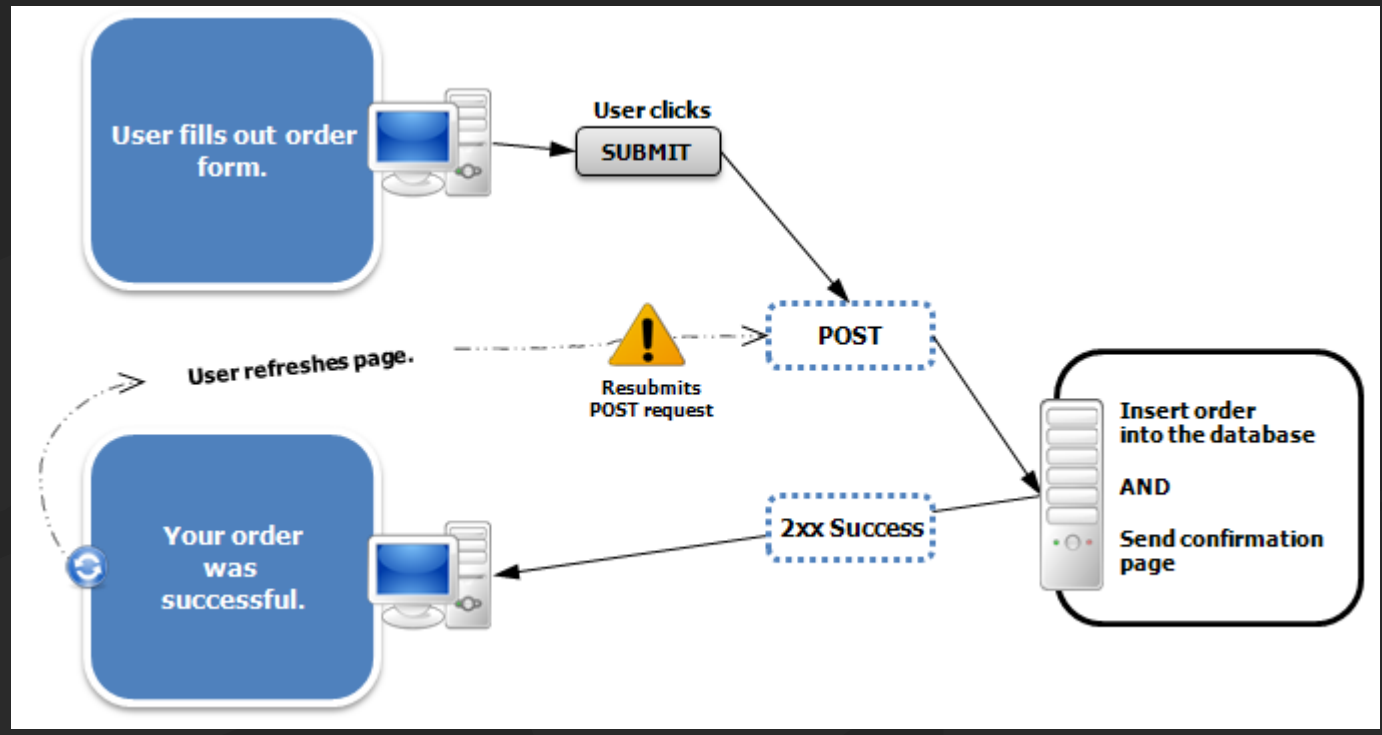
```
public RedirectToActionResult Index(string id) =>
                              RedirectToAction("Details", new { id });
```

Use a string-string dictionary to supply a parameter

```
public RedirectToActionResult Index(string id) =>
                RedirectToAction("Details",
                                 new Dictionary<string, string>(){ { "ID", id } } );
```

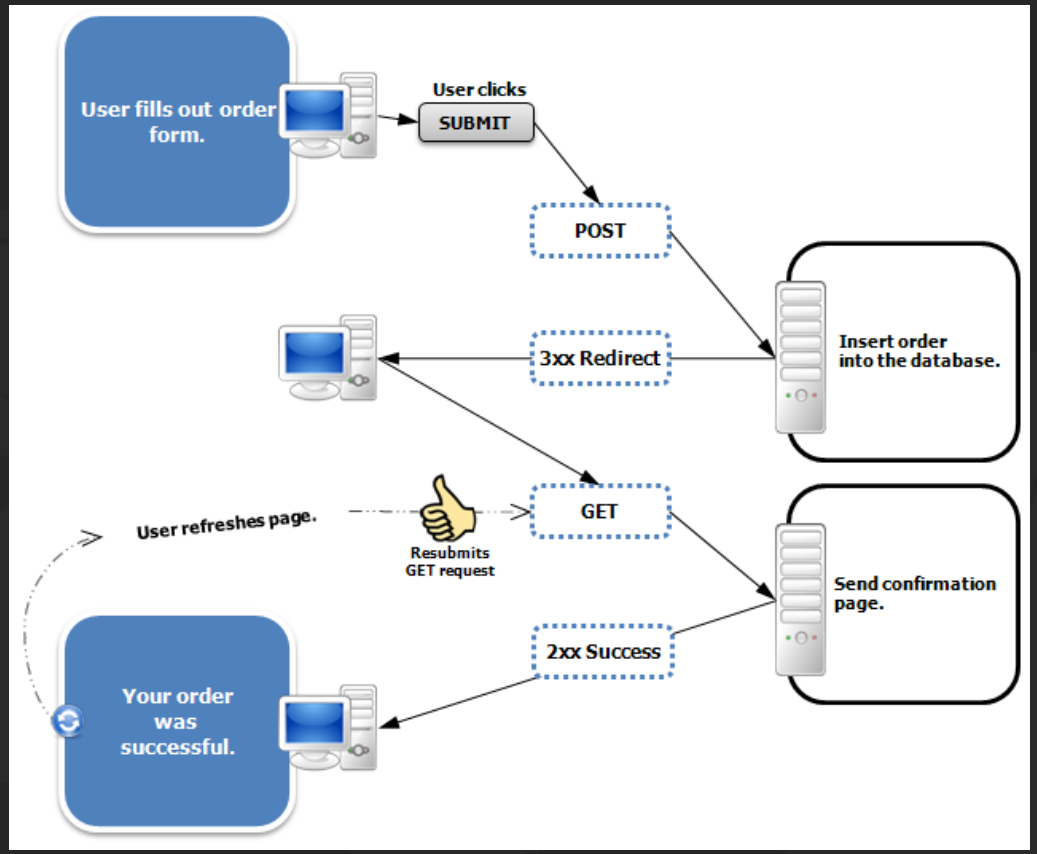
Post-Redirect-Get Pattern (PRG)

Diagram of “Double Post” Problem



When a web form is submitted to a server through an HTTP POST request, attempts to refresh the server response can cause the contents of the original POST to be resubmitted, possibly causing undesired results, such as a duplicate web purchase (“Double Post”)

Double Post Problem Solved with PRG



To avoid this problem, many web developers use the PRG pattern, instead of returning a web page directly, the POST returns a redirect. The HTTP 303 response code is used to ensure that in this situation, browsers can safely refresh the server response without causing the initial POST request to be resubmitted.

Example Action methods that use the PRG pattern

A Delete action method for a POST request

```
[HttpPost]                                                    // Post
public IActionResult Delete(Movie movie)
{
    context.Movies.Remove(movie);
    context.SaveChanges();
    return RedirectToAction("Index", "Home");                 // Redirect
}


[HttpGet]                                                     // Get
public IActionResult Index()
{
    var movies = context.Movies
        .Include(m => m.Genre)
        .OrderBy(m => m.Name)
        .ToList();
    return View(movies);
}
```

How to use the TempData Property

- Often used to transfer data from one controller to anther controller
- TempData is a property of the controller class that actually let you transfer data from controller **or** view
- Data in TempData persists across multiple requests until it is read. By contrast data in ViewBag or ViewData only persists until the end of the current request
- TempData is often used with the PRG pattern because that pattern takes place across two requests (the POST request and the subsequent GET Request).
- TempData can only store data that can be serialized such as permitive types.
- TempData is a dictionary (keys/values, contains() method to check for values)
- TempData is automatically enabled when you call **AddControllersWithViews()** method in **Startup.cs**

Example: TempData

An Action method that uses TempData with the PRG Pattern

```
[HttpPost]
public IActionResult Delete(Movie movie)
{
    context.Movies.Remove(movie);
    context.SaveChanges();
    TempData["message"] =
        $"{movie.Name} deleted from database.";
    return RedirectToAction("Index", "Home");
}
```

Example: Code that reads TempData value

```
...
<header class="jumbotron">
    <h1>My Movies</h1>
</header>...
@if (TempData.Keys.Contains("message"))
{
    <h4 class="bg-info text-center text-white p-2">
        @TempData["message"]
    </h4>
}
...
```

How to use methods of the TempData dictionary

Method	Description
Keep()	Marks all the values in the dictionary as unread, even if they've already been read
Keep(key)	Marks the value associated with the specified key as unread, even if it has already been read
Peek(key)	Reads the value associated with the specified key but does not mark it as read.

Details() action methods

```
[HttpPost]
0 references
public IActionResult Details(TeamViewModel model)
{
    Utility.LogTeamClick(model.Team.TeamID);

    TempData["ActiveConf"] = model.ActiveConf;
    TempData["ActiveDiv"] = model.ActiveDiv;
    return RedirectToAction("Details", new { ID = model.Team.TeamID });
}
```

```
[HttpGet]
0 references
public IActionResult Details(string id)
{
    var model = new TeamViewModel
    {
        Team = context.Teams
            .Include(t => t.Conference)
            .Include(t => t.Division)
            .FirstOrDefault(t => t.TeamID == id),
        ActiveDiv = TempData?["ActiveDiv"]?.ToString() ?? "all",
        ActiveConf = TempData?["ActiveConf"]?.ToString() ?? "all"
    };
    return View(model);
}
```

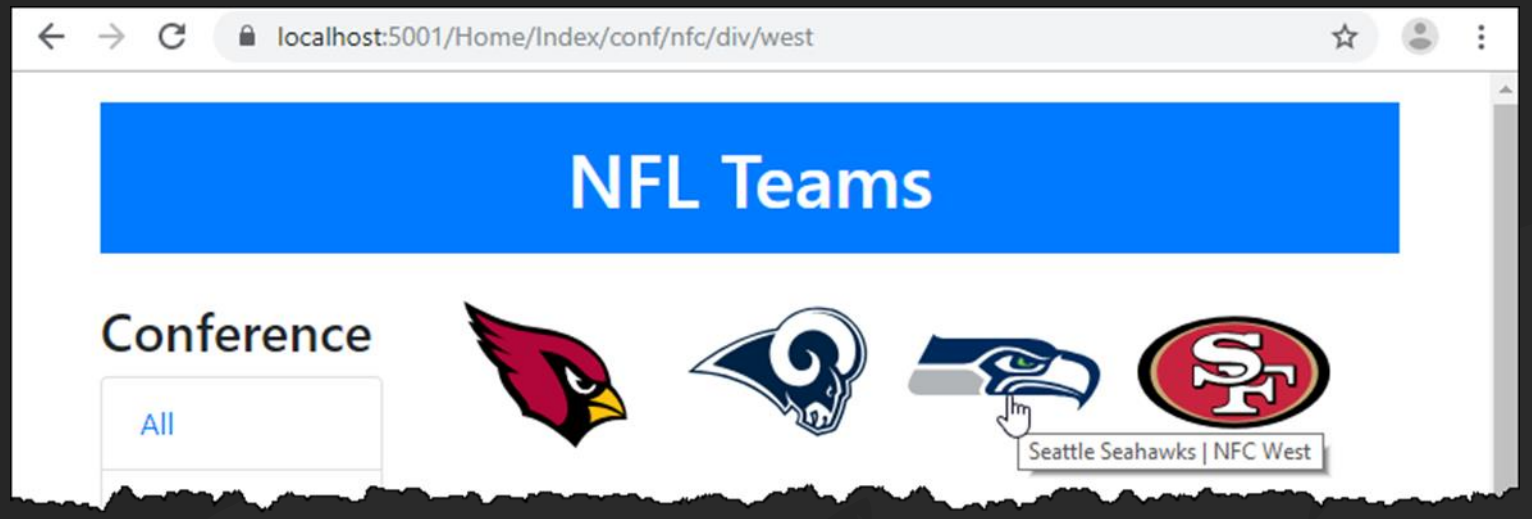
When to use the Keep() and Peek() methods

- Use **Peek()** when you know you want the value to stay marked as unread
- Use normal read and **Keep()** when you want to use a condition to determine whether to mark the value as unread.

NFL Teams Application Revisited
(*Ch08bNFLTeams*)

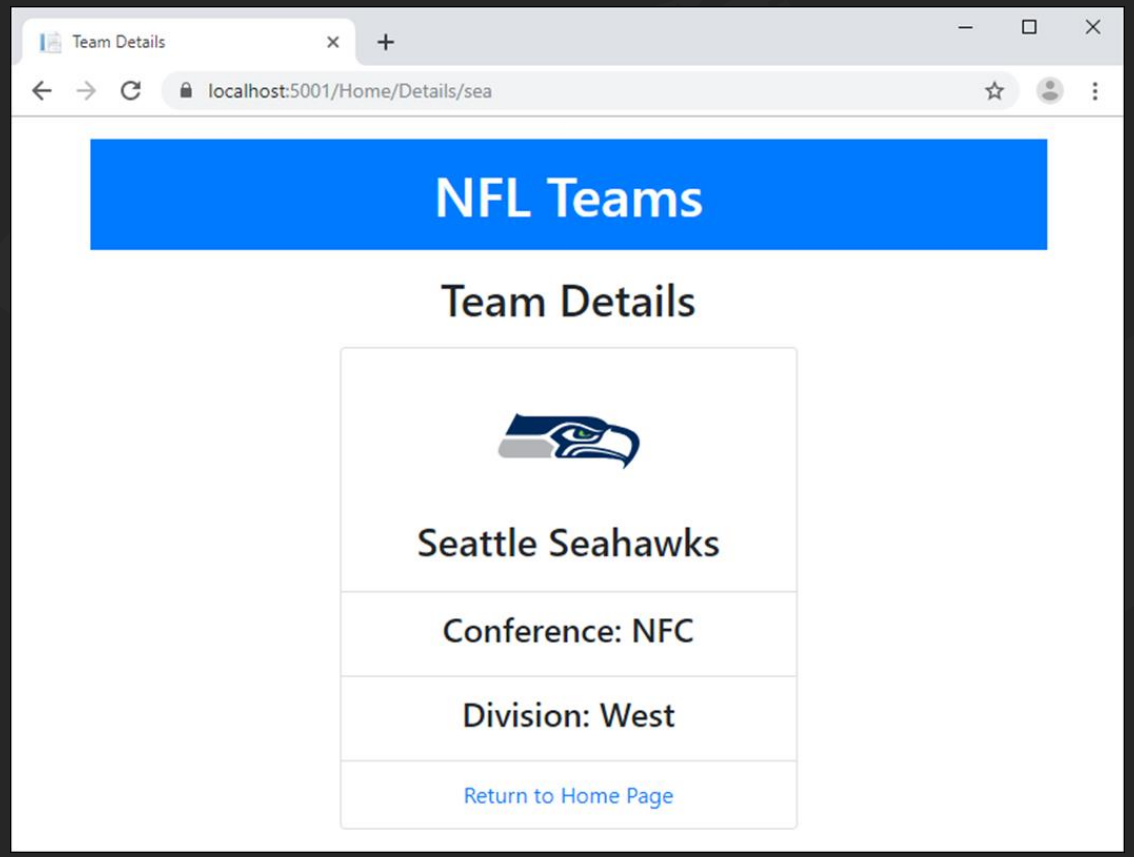
NFL Teams Application

The Home Page after Selecting a conference



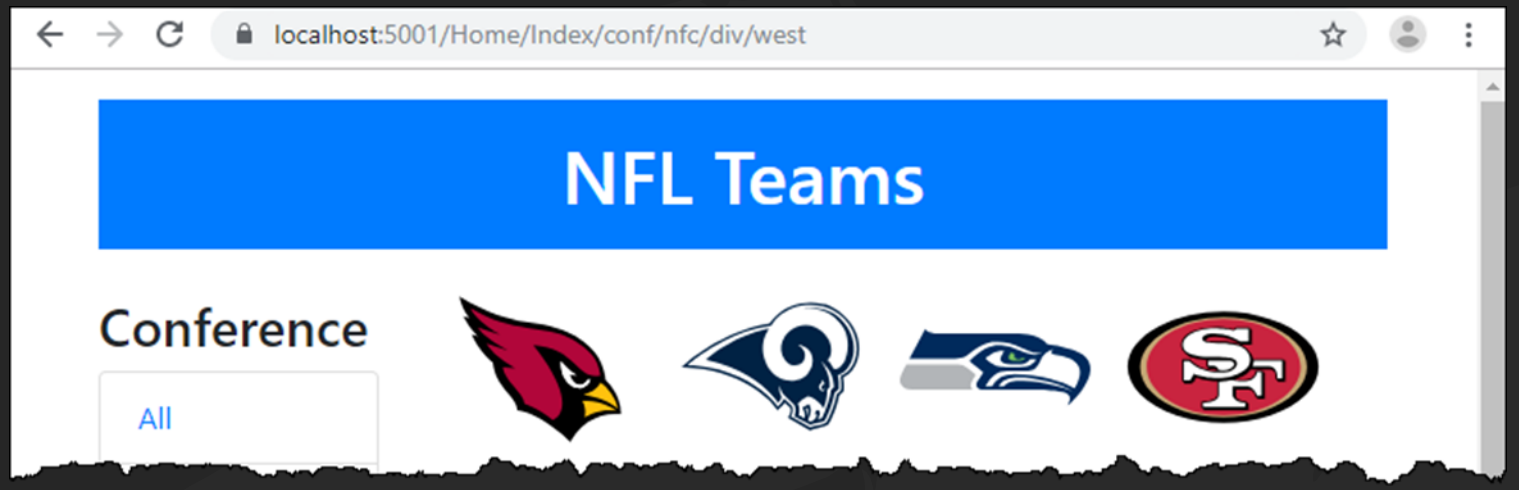
NFL Teams Application

The Details Page after clicking on a team



NFL Teams Application

The Home Page after clicking on the “Return to Home Page”



TeamListViewModel

ViewModel is C# class that holds all the data that a specific view requires.

```
public class TeamViewModel
{
    7 references
    public Team Team { get; set; }
    7 references
    public string ActiveConf { get; set; } = "all";
    7 references
    public string ActiveDiv { get; set; } = "all";
}
```

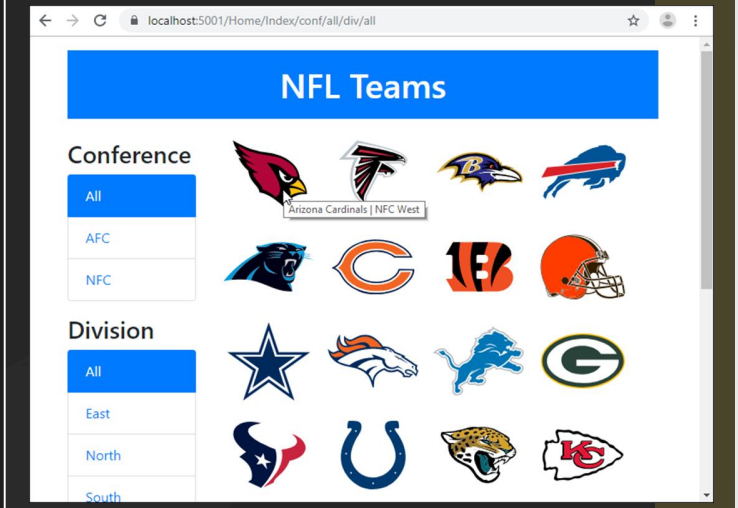
```
public class TeamListViewModel : TeamViewModel
{
    2 references
    public List<Team> Teams { get; set; }

    // use full properties for Conferences and Divisions
    // so can add 'All' item at beginning
    private List<Conference> conferences;
    2 references
    public List<Conference> Conferences {
        get => conferences;
        set {
            conferences = value;
            conferences.Insert(0,
                new Conference { ConferenceID = "all", Name = "All" });
        }
    }

    private List<Division> divisions;
    2 references
    public List<Division> Divisions {
        get => divisions;
        set {
            divisions = value;
            divisions.Insert(0,
                new Division { DivisionID = "all", Name = "All" });
        }
    }

    // methods to help view determine active link
    1 reference
    public string CheckActiveConf(string c) =>
        c.ToLower() == ActiveConf.ToLower() ? "active" : "";

    1 reference
    public string CheckActiveDiv(string d) =>
        d.ToLower() == ActiveDiv.ToLower() ? "active" : "";
}
```



The Updated Home/Index view

```
@model TeamListViewModel
@{
    ViewData["Title"] = "NFL Teams";
}
<div class="row">
    <div class="col-sm-3">
        <h3 class="mt-3">Conference</h3>
        <div class="list-group">
            @foreach (Conference conf in Model.Conferences) {
                <a asp-action="Index"
                    asp-route-activeConf="@conf.ConferenceID"
                    asp-route-activeDiv="@Model.ActiveDiv"
                    class="list-group-item
                        @Model.CheckActiveConf(conf.ConferenceID)">@conf.Name</a>
            }
        </div>
        <h3 class="mt-3">Division</h3>
        <div class="list-group">
            @foreach (Division div in Model.Divisions) {
                <a asp-action="Index"
                    asp-route-activeConf="@Model.ActiveConf"
                    asp-route-activeDiv="@div.DivisionID"
                    class="list-group-item
                        @Model.CheckActiveDiv(div.DivisionID)">@div.Name</a>
            }
        </div>
    </div>
```

```
    <div class="col-sm-9">
        <ul class="list-inline">
            @foreach (Team team in Model.Teams)
            {
                <li class="list-inline-item">
                    <form asp-action="Details" method="post">
                        <button type="submit" class="bg-white border-0">
                            <img src="~/images/@team.LogoImage" alt="@team.Name"
                                title="@team.Name | @team.Conference.Name @team.Division.Name" />
                        </button>

                        <input type="hidden" asp-for="@team.TeamID" />
                        <input type="hidden" asp-for="ActiveConf" />
                        <input type="hidden" asp-for="ActiveDiv" />
                    </form>
                </li>
            }
        </ul>
    </div>
</div>
```

NFL Teams Application

The Details View

```
@model TeamViewModel
@{
    ViewData["Title"] = "Team Details";
}

<h2 class="text-center p-2">Team Details</h2>

<div class="row">
    <div class="col-6 offset-3">
        <ul class="list-group text-center">
            <li class="list-group-item">
                <img src="~/images/@Model.Team.LogoImage" alt="" />
                <h3>@Model.Team.Name</h3>
            </li>
            <li class="list-group-item">
                <h4>Conference: @Model.Team.Conference.Name</h4>
            </li>
            <li class="list-group-item">
                <h4>Division: @Model.Team.Division.Name</h4>
            </li>
            <li class="list-group-item">
                <a asp-action="Index" asp-route-activeConf="@Model.ActiveConf"
                    asp-route-activeDiv="@Model.ActiveDiv">Return to Home Page</a>
            </li>
        </ul>
    </div>
</div>
```

Questions?