LookUp

Bien, ahora vamos a corregir un error o más bien un feature que tenemos, si nos vamos al Feed donde salen los eventos y yo hago clic en “Going”, el cambia a azul y marca que voy a ir, como debe de ser, este es el funcionamiento correcto, pero si refresco la página se va, aunque si verifico en la base de datos puedo ver que está bien, entonces, tengo que corregir el estado de inicio del evento.

Lo que haremos es ir al controlador home, en la acción Index y una vez aquí, justo antes de inicializar el ViewModel, voy a loguear todas las posibles asistencias futuras del usuario.

Nosotros vamos a necesitar convertir la lista que vamos a manipular a una estructura de datos que me permita buscar rápido dentro de la lista en cuestión una asistencia; porque cuando rendericemos cada evento, necesitamos marcar si yo (el usuario logueado) voy a asistir o no al evento que se está renderizando, para eso hacemos uso de un LookUp.

Un LookUp es como un diccionario, internamente usa una tabla hash para rápidamente buscar y localizar objetos por eso es tan potente y veloz el LookUp recibe una expresión lambda que nos va a servir para indicar la llave y después de eso instanciamos el resultado en el ViewModel.

```
g.Venue.Contains(query));
        }
```

```
            var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);
            var attendances = _context.Attendances
                .Where(a => a.AttendeeId == userId && a.Event.DateTime > DateTime.Now)
                .ToList()
                .ToLookup(a => a.EventId);

            var vm = new EventsViewModel
            {
                UpcomingEvents = upcomingEvents,
                ShowActions = User.Identity.IsAuthenticated,
                Heading = "Upcoming Events ", //My upcoming events on the other screen
                SearchTerm = query,
                Attendances = attendances
            };
```

Obviamente la propiedad Attendances no la tenemos en el ViewModel por lo que tenemos que crearla y lo podemos hacer con el Intellisense quedando de la siguiente forma.

```
public class EventsViewModel
{
    public IEnumerable<Event> UpcomingEvents { get; set; }
    public bool ShowActions { get; set; }
    public string Heading { get; set; }
    public string SearchTerm { get; set; }
    public ILookup<int, Attendance> Attendances { get; internal set; }
}
```

El ILookUp es una interfaz genérica que recibe dos parámetros el primero es el tipo de la llave, y el segundo es el tipo de elemento que deseo almacenar en este LookUp.

Ahora vamos a renderizar en la vista que es nuestro Partial “_EventsPartial.cshtml”, sin embargo, en algún momento de la vida que no recuerdo el código de la vista se la habían hecho ligeras modificaciones quedando de la forma en que se describe debajo, así que solo lo que está en amarillo pertenece a la explicación, el resto es arreglando el diseño.

```
<div class="details">
```

```
<div class="details">
 <span class="coach">
   <a asp-controller="Events" asp-action="Details" asp-route-id="@item.Id">
      @item.Coach.Name
   </a>
   @if (item.IsCanceled)
   {
    <span class="label label-warning">Canceled</span>
   }
   @if (Model.ShowActions)
   {
    <button data-user-id="@item.CoachId"
            asp-route-id="@item.CoachId"
            class="btn btn-default btn-sm pull-right js-toggle-follow">
     Follow
    </button>
   }
 </span>
 <span class="type">
            @item.Type.Name
 </span>
    @if (Model.ShowActions && !item.IsCanceled)
    {
      <button data-event-id="@item.Id"
              asp-route-id="@item.Id"
              class="btn btn-default btn-sm pull-right js-toggle-attendance">
      Going?
      </button>
      <button data-user-id="@item.CoachId"
              asp-route-id="@item.CoachId"
              class="btn btn-default btn-sm pull-right js-toggle-follow">
       Follow
      </button>
    }
   </div>
```

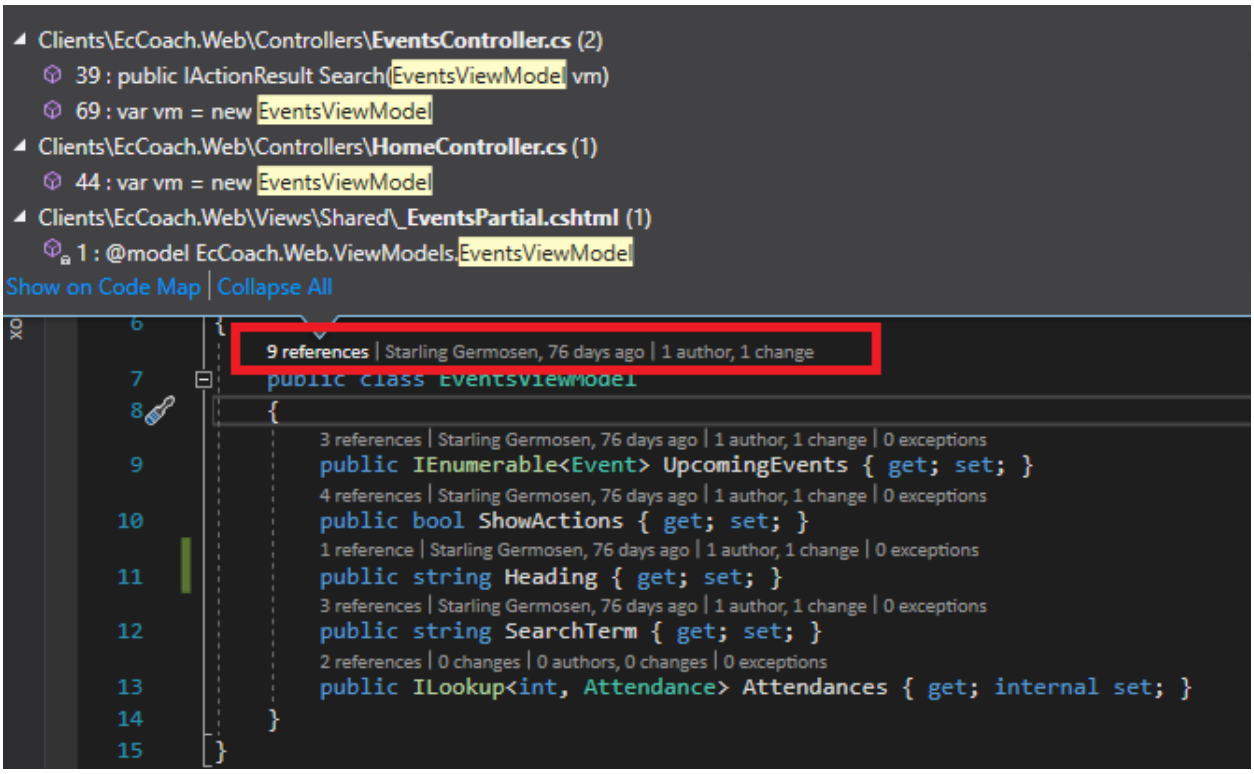
Donde lo que habíamos hecho era mover el botón de seguir al coach justo al lado del nombre de su nombre y con la propiedad ShowActions que habíamos programado en el controlador, validamos si debemos mostrar o no la acción de seguir, lo propio para el indicador de si vamos a ir al evento o no, donde validamos también que el evento no esté cancelado para poder mostrar la opción.

Bueno ahora, nos valemos de la técnica de Razor y código C# para cargar de manera dinámica una clase btn info u otra; dentro de cualquier elemento del Dom, yo puedo hacer uso del @ para

indicar que voy a iniciar una instrucción de C#, y hago uso del paréntesis para indicar que es “In Line Code” en caso contrario tendría que abrir y cerrar llaves y poner el código dentro. Si tenemos un resultado de asistencia, renderizamos una clase btn-info en caso contrario btn-default. Quedaría de esta forma.

```
<button data-event-id="@item.Id"
         asp-route-id="@item.Id"
         class="btn btn-default
                @(Model.Attendances.Contains(item.Id) ? "btn-info" : "btn-default")
                btn-sm pull-right js-toggle-attendance">
         Going?
</button>
```

Si probamos, veremos que todo está nítido, carga de manera inicial el color que debe cargar y cambia cuando hago la operación. pero ahora surge un inconveniente, nosotros tenemos una nueva propiedad, que es el look up, la cual si no la inicializamos nos puede arrojar excepciones por referencia nula, por ende, tenemos que buscar donde quiera que hayamos implementado el ViewModel de Events y en todas sus referencias me aseguro de que al inicializar el EventsViewModel, le pase valores al LookUp si lo amerita, para esto sencillamente nos situamos arriba de la clase y podemos ver al lado de la última persona que modificó, la cantidad de referencias, desde aquí podemos verlas e ir una por una a tomar los correctivos de lugar las de vistas no nos interesan, solo las de clases.



Podemos ver que tenemos una referencia en EventsController, en la parte del Search, solo hacemos una redirección por lo que no tenemos que preocuparnos, esa la podemos dejar así.

Pero en la otra aquí en la acción Attending, tenemos una referencia a EventsViewModel, donde retornamos para mostrar los eventos a los que estoy atendiendo, así que vamos al controlador home, copiamos el código para marcar si está asistiendo, porque no queremos un Null Reference Exeption, pero aquí vienen a crucificarme los jinetes del Apocalipsis y con razón, porque estoy violando el principio DRY, bueno, eso es algo que vamos a resolver más tarde, esto lo hacemos de hecho para darle más sentido al patrón repositorio, así que tranquilos, no me maten, al menos no por ahora, vamos a copypastear y después metemos mano.

```
var events = _context.Attendances
.Where(a => a.AttendeeId == userId)
.Select(a => a.Event)
.Include(p => p.Type)
.Include(p => p.Coach)
.ToList();

var attendances = _context.Attendances
  .Where(a => a.AttendeeId == userId && a.Event.DateTime > DateTime.Now)
  .ToList()
  .ToLookup(a => a.EventId);

var vm = new EventsViewModel
{
    UpcomingEvents = events,
    ShowActions = User.Identity.IsAuthenticated,
    Attendances= attendances
};
```

Para memorizar: siempre que creamos una nueva propiedad, debemos revisar todas las referencias que se tienen hacia ese objeto (la clase donde creamos esa propiedad) y validar que necesitemos o no inicializar esos nuevos elementos. Seguimos con nuestro proyecto.