

Separación de Conceptos, ahora sí, hecho como se debe.

Hay algo que atormenta mi cabeza y no me deja dormir por las noches, en verdad no, pero tengo que meterle algo de drama, es el tema de la separación de conceptos que venimos violando hace un viaje de rato, nuestro “toggleAttendance” es responsable de trabajar con elementos del DOM pero también es responsable de acceso a datos, tenemos que él se encarga de realizar dos cosas que no le competen y eso significa que no tenemos buena separación de conceptos.

Por ende, creo que las cosas se verían un poco más decentes si se modificara un poco el código primero que nada modificando las llamadas al api y separándolas en funciones aparte.

```
9  var toggleAttendance = function (e) {
10     button = $(e.target);
11     if (button.hasClass("btn-default")) {
12         ...
13     }
14     else {
15         ...
16     }
17 };
18
19 var createAttendance = function () {
20     $.post("/api/attendances/", { eventId: button.attr("data-event-id") })
21         .done(done)
22         .fail(fail);
23 };
24
25 var deleteAttendance = function () {
26     $.ajax({
27         url: "/api/attendances/" + button.attr("data-event-id"),
28         method: "DELETE"
29     })
30         .done(done)
31         .fail(fail);
32 };
33
```

Luego dentro de ese if y else puedo poner la llamada a mis sentencias o funciones, como es solo una línea no necesito las llaves.

```
var init = function () {
    $(".js-toggle-attendance").click(toggleAttendance);
};

var toggleAttendance = function (e) {
    button = $(e.target);
    if (button.hasClass("btn-default"))
        createAttendance();
    else
        deleteAttendance();
};

var createAttendance = function () {
    $.post("/api/attendances/", { eventId: button.attr("data-event-id") })
        .done(done)
        .fail(fail);
};

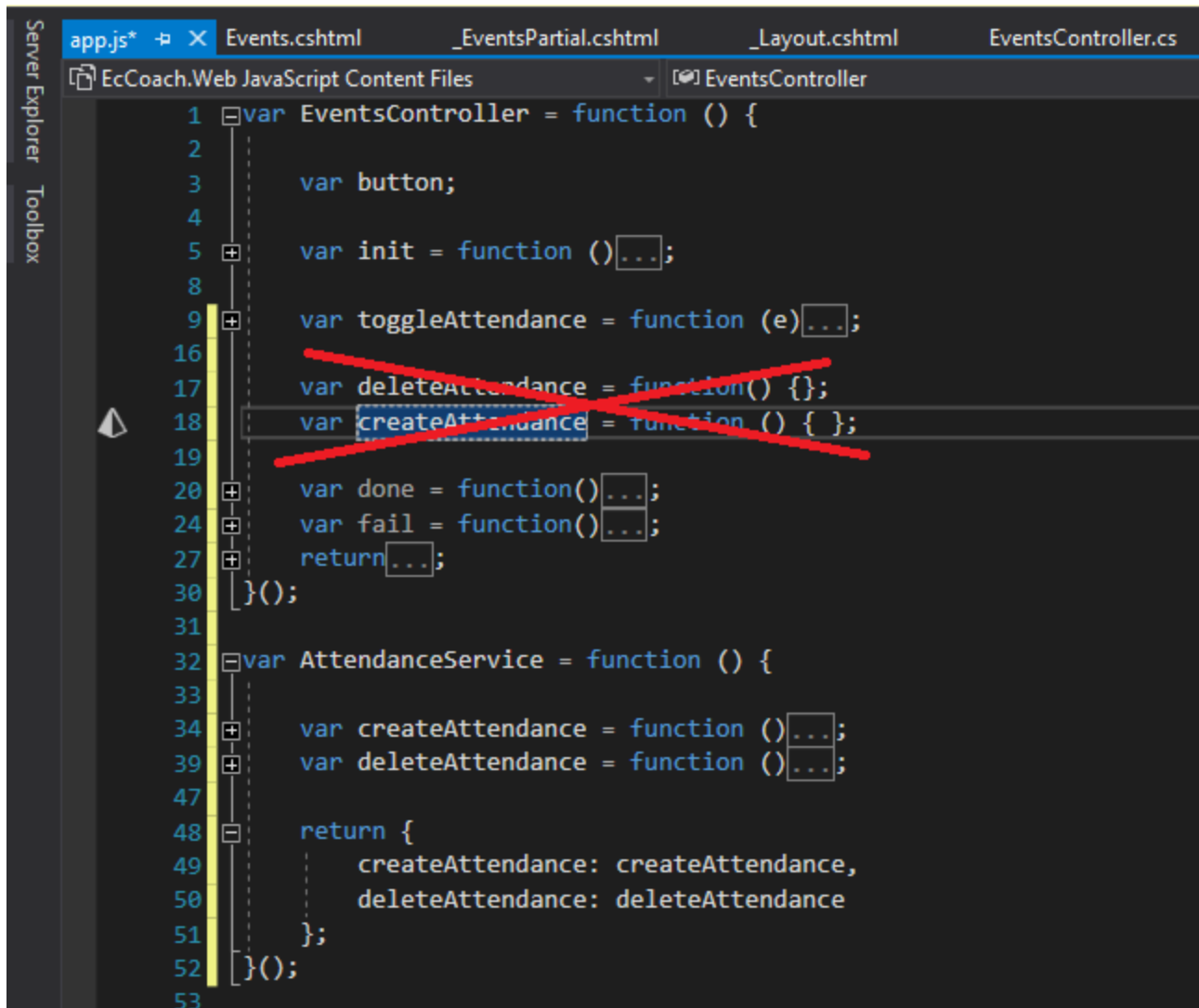
var deleteAttendance = function () {
    $.ajax({
        url: "/api/attendances/" + button.attr("data-event-id"),
        method: "DELETE"
    })
        .done(done)
        .fail(fail);
};

var done = function () {
    var textToDisplay = (button.text() == "Going") ? "Going?" : "Going";
    button.toggleClass("btn-info").toggleClass("btn-default").text(textToDisplay);
}
```

Ahora tenemos algo más limpio, más ordenado, tenemos separación de conceptos pues tenemos pequeños métodos privados que se encargan de hacer una única cosa, lo que quiere decir que respetamos a su vez el principio de única responsabilidad, tenemos un módulo bastante limpio, pero aún hay algo que está de sobra, y es el hecho de tener el createAttendanse y el deleteAttendanse en el mismo JavaScript que se encarga de manipular eventos, estos métodos son de acceso a datos.

El controlador o más bien el módulo js que tenemos solo debe estar encargado de manipular los eventos, cargarlo a la vista y actualizarla.

Hacer esas cosas como actualizar datos, hacer delete, hacer post y eso es una responsabilidad diferente, además si en el futuro quiero crear otro controlador que actualice la asistencia, tendría que crear estos métodos allá, por lo cual vamos a extraer estos métodos a un nuevo módulo llamado attendanceService. Nosotros como buenos programadores debemos trabajar de una manera en que sea responsabilidad de servicios brindarnos interacción con el server creando así la puerta a poder reusar los servicios desde otros controladores (módulos) y vistas.



```
1 var EventsController = function () {
2
3     var button;
4
5     var init = function ()...;
6
7
8
9     var toggleAttendance = function (e)...;
10
11
12
13
14
15
16     var deleteAttendance = function () {};
17
18     var createAttendance = function () { };
19
20     var done = function ()...;
21
22
23
24     var fail = function ()...;
25
26
27     return...;
28
29 }();
30
31
32 var AttendanceService = function () {
33
34     var createAttendance = function ()...;
35
36
37
38     var deleteAttendance = function ()...;
39
40
41
42
43
44
45
46
47
48     return {
49         createAttendance: createAttendance,
50         deleteAttendance: deleteAttendance
51     };
52 }();
53
```

```
var AttendanceService = function () {

    var createAttendance = function () {
        $.post("/api/attendances/", { eventId: button.attr("data-event-id") })
            .done(done)
            .fail(fail);
    };

    var deleteAttendance = function () {
        $.ajax({
            url: "/api/attendances/" + button.attr("data-event-id"),
            method: "DELETE"
        })
            .done(done)
            .fail(fail);
    };

    return {
        createAttendance: createAttendance,
        deleteAttendance: deleteAttendance
    };
};
```

```
}());
```

Pero ahora hay un problema y es que no tenemos referencia al botón que desencadena el evento acá adentro de nuestro servicio y realmente no debemos de tenerla; nunca deberíamos tener referencia a un elemento del UI dentro de un servicio, por lo que lo correcto es recibir como parámetro el id y hago lo propio con el done y el fail, debo recibirlos como parámetros pues lo que va a pasar cuando yo (servicio) funcione o falle, no es mi responsabilidad, así que lo único que debo de hacer es pedir como parámetro cuales son los métodos que voy a ejecutar si todo sale bien y cuales si todo male sale (sale mal)

```
31
32 var AttendanceService = function () {
33
34     var createAttendance = function ( eventId, done, fail ) {
35
36         $.post("/api/attendances/", { eventId: eventId, button.attr("id") = event-id })
37             .done(done)
38             .fail(fail);
39     };
40     var deleteAttendance = function ( eventId, done, fail ) {
41         $.ajax({
42             url: "/api/attendances/" + eventId, button.attr("id") = event-id,
43             method: "DELETE"
44         })
45             .done(done)
46             .fail(fail);
47     };
48
49     return {
50         createAttendance: createAttendance,
51         deleteAttendance: deleteAttendance
52     };
53 }();
54
```

Ahora tenemos un servicio más limpio que no sabe nada del UI y que puede ser reusado desde múltiples controladores (módulos)

Después de esto nos toca recibir, en el “EventsController”, como parámetro la variable que vamos a usar para invocar este servicio, es una especie de recibirlo por inyección de dependencia para que más o menos se entienda un poco el homologó, ¿recuerdan lo que significa IIFI?, es la Expresión de Función de Invocación Inmediata o de auto invocación, nuestro controlador posee una función y cuando esa función es invocada tenemos que especificar un valor para este parámetro, tenemos que pasar un argumento, ¿Dónde llamamos esto? Al final, donde llamamos la función, ¿Cuál debería ser el valor de ese parámetro? Una referencia al módulo AttendanceService.

Ojo y atención a la capitalización, se usa Pascal Case nombrar los módulos, donde la primera letra de cada palabra está en mayúscula, y los parámetros están en Cammel Notation, donde la primera letra de la primera palabra está en minúscula y la primera letra de cada palabra adicional está en mayúscula (Si quieres entender mejor el tema de los Pascal, Camel y su usabilidad visita el siguiente enlace (<https://medium.com/better-programming/string-case-styles-camel-pascal-snake-and-kebab-case-981407998841>)).

```
app.js*  Events.cshhtml  _EventsPartial.cshhtml  _Layout.cshhtml  EventsController.cs  Attendances
EcCoach.Web JavaScript Content Files  [EventsController]  <function>

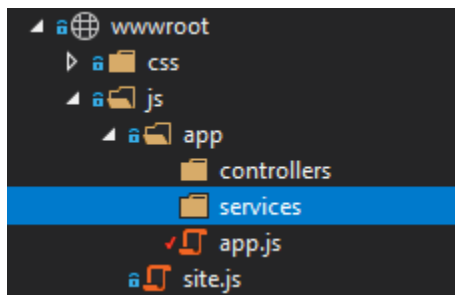
1
2 var EventsController = function ( attendanceService ) {
3
4     var button;
5
6     var init = function (...);
7
8
9
10    var toggleAttendance = function (e) {
11        button = $(e.target);
12
13        var eventId = button.attr("data-event-id");
14
15        if (button.hasClass("btn-default"))
16
17            attendanceService.createAttendance( eventId, done, fail );
18
19        else
20
21            attendanceService.deleteAttendance( eventId, done, fail );
22    };
23
24
25
26
27    var done = function (...);
28    var fail = function (...);
29    return ...;
30
31    }( AttendanceService );
```

Otra forma seria en lugar de pasar estos tres elementos, pasar un solo objeto con los tres elementos dentro, pero por ahora este no está mal, vamos a dejarlo así.

Ahora si podemos decir que todo está relativamente bien, pero tenemos un código que debemos mover a otro lugar pues independientemente de que ahora todo está más organizado, esto

puede crecer mucho y hacer imposible el proceso de búsqueda y mantenimiento.

Creamos dos nuevas carpetas en nuestra carpeta app “controllers” y “services”



Dentro de “services” creamos el archivo “attendanceService.js” donde cortamos el código del AttendanceService y lo pegamos aquí. Para luego continuar creando en la carteta “controllers” el archivo “eventsController” donde vamos a pegar el controlador de Events que cortaremos del archivo app.js

Ahora tenemos vacío el app.js, pero podemos usarlo después para tener configuraciones globales (si, yo se que tenemos el toggle-follow rodando por ahí, pero hacerle el proceso les toca a ustedes.

Añadimos estos dos nuevos js al _layout para que puedan ser ejecutados cuando se necesiten y haya referencia a ellos, aunque en este caso no aplica, pero puede haber casos donde en la carga inicial de un elemento necesitemos un orden específico porque un elemento depende de que ya exista otro previamente en este caso añadimos el servicio primero y después el controller. Organizar el javascript de forma manual sirve para aplicaciones pequeñas, pero a medida que tienes una app super robusta con cientos de

javascripts se torna difícil saber cuál es el orden de dependencia y es ahí donde tenemos que recurrir a librerías externas como RequiredJs, pero eso es un tema de otro programa de clases.

Este método tiene un ligero problema, y es el hecho de que solo está adherido a elementos que hayan sido creados con la primera carga de la página, no va a funcionar con elementos que sean cargados de forma dinámica, como por ejemplo si nosotros tenemos un lazy loading o una paginación o un load more.

Y el otro problema es que voy a tener una implementación en memoria de .js-toggle-attendance por cada elemento que tenga esta clase, ósea, cada elemento evento que se muestra en la pantalla va a tener un “Going” o un “Going?” por lo que si tenemos en pantalla 10 eventos, vamos a tener 10 implementaciones en memoria del toggle-attendance por lo que debo hacer uso del método On de jquery.

Vamos al EventsController Empezamos con un contenedor, al cual le vamos a indicar el on, el primer argumento es el tipo de evento que queremos capturar, el segundo argumento es el selector, y el tercer argumento es el manejador.

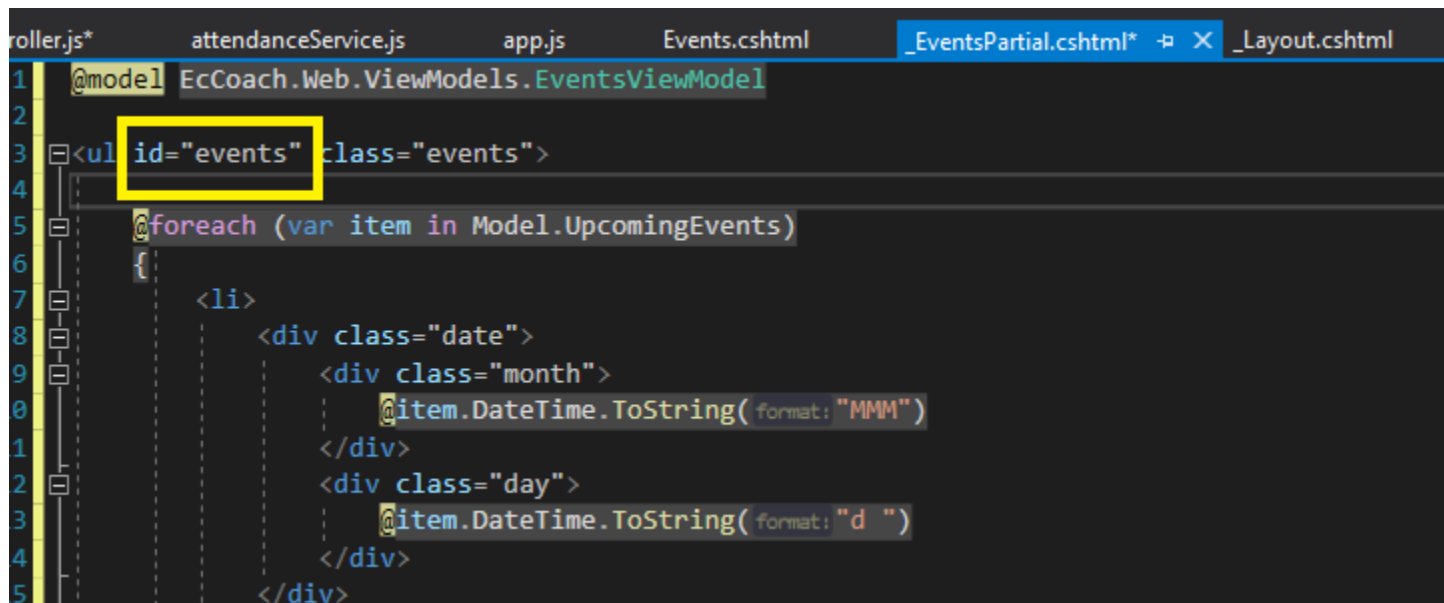
```
var init = function () {  
    $(container).on("click", ".js-toggle-attendance", toggleAttendance);  
    $(".js-toggle-attendance").click(toggleAttendance);  
};
```

Esta implementacion elimina los dos problemas que habia descrito previamente, solo tenemos una instancia de toggle-attendance en la memoria y si añadimos otro elemento a este contenedor que tenga la clase selectora “js-toggle-attendance” el metodo toggleAttendance sera llamado cuando haga click.

Bien, ahora container debe ser un elemento que sea recibido en el método init

```
var init = function (container) {  
    $(container).on("click", ".js-toggle-attendance", toggleAttendance);  
};
```

Ahora tenemos que pasarle ese contenedor, el cual es un selector que va a representar el evento (Presentación, Acontecimiento), así que vamos al “Events.cshtml” y donde llamamos el método Init debemos pasarle un selector y si nos vamos al “_EventsPartial.cshtml” podemos identificar el “ul” con la clase “events” es donde renderizamos cada evento por lo cual esto lo podemos volver nuestro contenedor dándole un id pues es el selector que debemos enviar



```
roller.js*  attendanceService.js  app.js  Events.cshtml  _EventsPartial.cshtml*  _Layout.cshtml  
1  @model EcCoach.Web.ViewModels.EventsViewModel  
2  
3  <ul id="events" class="events">  
4  
5      @foreach (var item in Model.UpcomingEvents)  
6      {  
7          <li>  
8              <div class="date">  
9                  <div class="month">  
10                     @item.DateTime.ToString(format: "MMM")  
11                 </div>  
12                 <div class="day">  
13                     @item.DateTime.ToString(format: "d ")  
14                 </div>  
15             </div>  
16         }  
17     </ul>
```

Y entonces cuando llamamos el método init, le pasamos el selector

```
sController.js*  attendanceService.js  app.js  Events.cshtml*  _EventsPartial.cshtml*
20  </form>
21  @await Html.PartialAsync(partialViewName: "_EventsPartial", Model)
22
23
24  @section scripts
25  {
26      <script>
27          $(document).ready(function () {
28
29              EventsController.init("#events");
30
31          });
32      </script>
33  }
```

Bien, ahora, como sé que ustedes no lo van a hacer, nos toca hacer la refactorización del following

Creamos con controlador llamado “EventDetailsController”

Es un nuevo módulo que tiene dependencia de un módulo llamado followingservice, la implementación es muy similar a lo que vimos previamente.

En el init subscribimos un evento click a la clase js-toggle-follow, luego tenemos una referencia al botón que hice click del cual obtengo el id que voy a enviar al servicio.

Verifico si tiene la clase default y si la tiene la envío a hacer el create en caso contrario la envío a hacer el delete, es similar a lo que vimos anteriormente por ende no requiere mucha explicación, acá el código.

```

var EventDetailsController = function (followingService) {

    var followButton;

    var init = function () {
        $(".js-toggle-follow").click(toggleFollowing);
    };
    var done = function () {
        var textToDisplay = (followButton.text() == "Follow") ? "Following" : "Follow";
        followButton.toggleClass("btn-info").toggleClass("btn-
default").text(textToDisplay);
    };
    var fail = function () {
        alert("Someting fail");
    };

    var toggleFollowing = function (e) {
        followButton = $(e.target);

        var followeeId = toggleFollowing.attr("data-user-id");

        if (followButton.hasClass("btn-default"))
            followingService.createFollowing(followeeId, done, fail);
        else
            followingService.deleteFollowing(followeeId, done, fail);
    };

    return {
        init: init
    };
}(FollowingService);

```

El servicio “followingService” tiene dos métodos y hacemos casi lo mismo que habíamos hecho en el anterior

```

var FollowingService = function () {

    var createFollowing = function (followeeId, done, fail) {

        $.post("/api/followings/", { followeeId: followeeId })
            .done(done)
            .fail(fail);
    };
    var deleteFollowing = function (followeeId, done, fail) {
        $.ajax({
            url: "/api/followings/" + followeeId,
            method: "DELETE"
        })
            .done(done)
            .fail(fail);
    };

    return {

```

```

        createFollowing: createFollowing,
        deleteFollowing: deleteFollowing
    };
}();

```

Creamos su end point para el delete o “Unfollow” dentro del Api “FollowingsController”

```

[HttpDelete]
public ActionResult Unfollow(string id)
{
    var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

    var following = _context.Followings
        .SingleOrDefault(a => a.FollowerId == userId && a.FollowerId == id);

    if (following == null)
        return NotFound();

    _context.Followings.Remove(following);
    _context.SaveChanges();
    return Ok(id);
}

```

Y finalmente tenemos end point que hace la eliminación. En el código anterior de “follow” había un error porque estábamos buscando por el id incorrecto.

```

var exists = _context.Followings.Any(a => a.FollowerId == userId && a.FolloweeId ==
dto.FolloweeId);

```

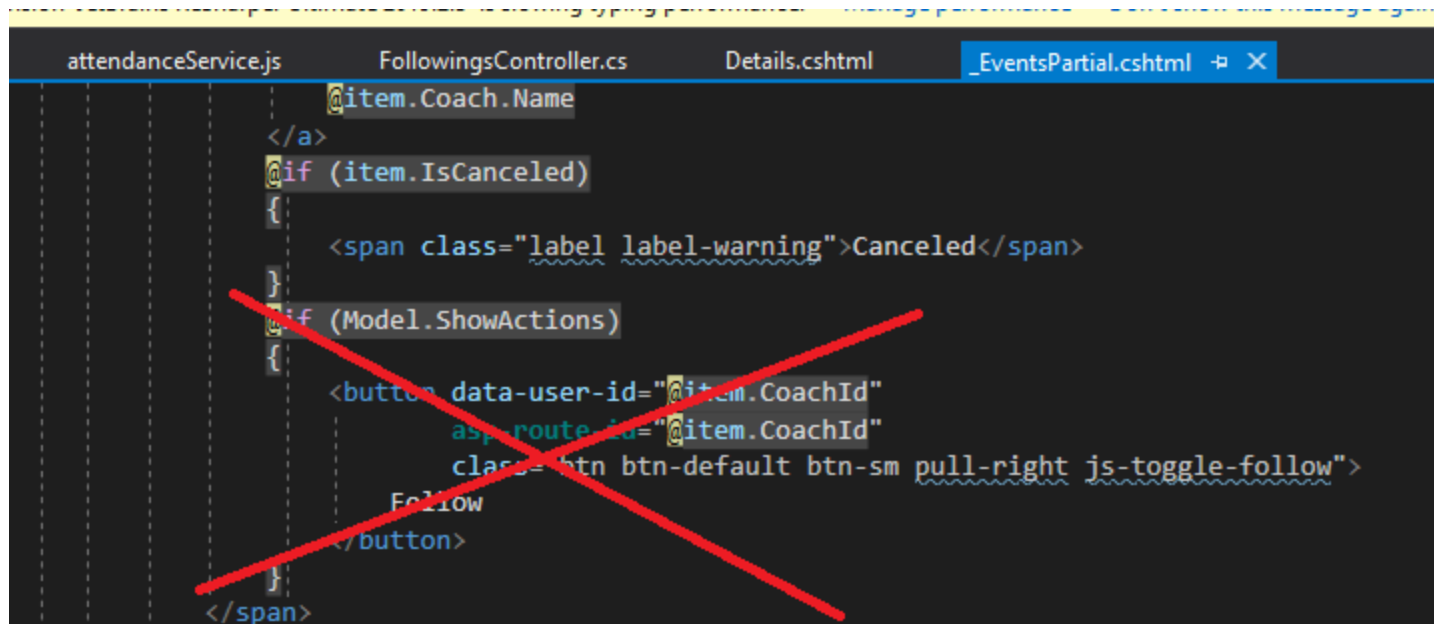
Podemos simplificar esto para ahorrarnos la variable de la siguiente manera

```

if (_context.Followings.Any(a => a.FollowerId == userId && a.FolloweeId ==
dto.FolloweeId))
    return BadRequest("The attendance already exists");

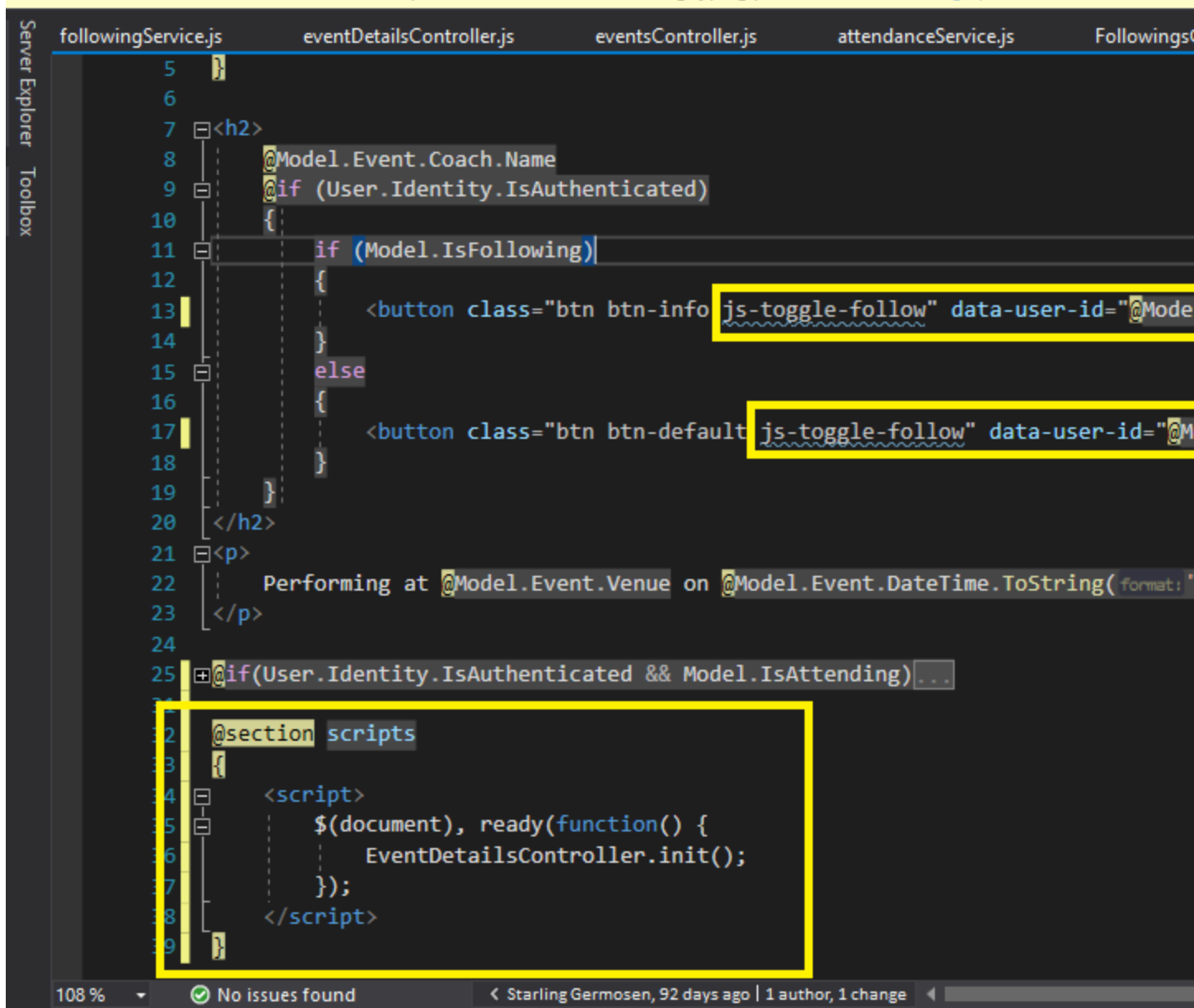
```

Y finalmente en la vista detalle tenemos una sección de script que vamos a mover la funcionalidad del feed para el detalle del evento, pues se ve algo feo tener el botón follow y el Going en la misma ventana, es algo más de estética que otra cosa.



```
attendanceService.js | FollowingsController.cs | Details.cshtml | _EventsPartial.cshtml X
@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>
```

A los botones le agregamos la clase `js-toggle-follow` para que desencadene los eventos, pues es la clase que vamos a usar como selector en nuestro código jQuery, también seteamos el `data-attribute`



```
5 }
6
7 <h2>
8     @Model.Event.Coach.Name
9     @if (User.Identity.IsAuthenticated)
10     {
11         if (Model.IsFollowing)
12         {
13             <button class="btn btn-info js-toggle-follow" data-user-id="@Model.Event.Coach.Id">Following</button>
14         }
15         else
16         {
17             <button class="btn btn-default js-toggle-follow" data-user-id="@Model.Event.Coach.Id">Follow</button>
18         }
19     }
20 </h2>
21 <p>
22     Performing at @Model.Event.Venue on @Model.Event.DateTime.ToString("format:MM/dd/yyyy")
23 </p>
24
25 @if (User.Identity.IsAuthenticated && Model.IsAttending)
26 {
27     @section scripts
28     {
29         <script>
30             $(document).ready(function() {
31                 EventDetailsController.init();
32             });
33         </script>
34     }
35 }
```

Otro tema por cuestiones de estética que debemos realizar es poner el botón going debajo, pues por tema de usabilidad no me gusta el movimiento de los ojos para machear el evento con su respectivo botón, para eso realizamos un par de cosas de estilo.

Le añadimos el display en bloque y el margen



Resumen

En este módulo hicimos mucho énfasis en la separación de conceptos, iniciamos con un JavaScript espagueti escrito directo en nuestra view e hicimos una refactorización paso a paso moviéndolo a archivos separados y módulos separados, cada uno de esos módulos tiene métodos pequeños que son fáciles de entender y mantener, a medida que tus códigos crecen tienes que organizarlo y modularizarlo, nosotros hicimos uso de una técnica simple llamada “Revealing Module Pattern”, si bien hubiéramos podido usar alguna librería para eso como angular, backbone, amber para modularizar, pero eso sería agregar un extra de complejidad a este curso y no estamos en eso estamos en buscar el efectivo. En el siguiente modulo veremos la separación de conceptos en nuestro Backend.