

Modularización y DRY del Código JavaScript

Este módulo trata sobre escribir un JavaScript leíble, mantenible y modularizado para ser mucho más ordenados y para fomentar el BAM.

Veremos un poco de la separación de conceptos y limpiaremos algunas de las marañas de código que tenemos.

La modularización inicia por casa (Patrón Modular en JavaScript)

El primer error que tenemos es unas de las malas prácticas con las que iniciamos y es una regla de legibilidad, no se debe escribir JavaScript fuera de un archivo .js, no es correcto bajo ninguna justificación, excepto la de explicar conceptos didácticos y como en el día a día usted no anda explicando a estudiantes sino ganando dinero, siempre ponga sus JavaScript en un archivo Js.

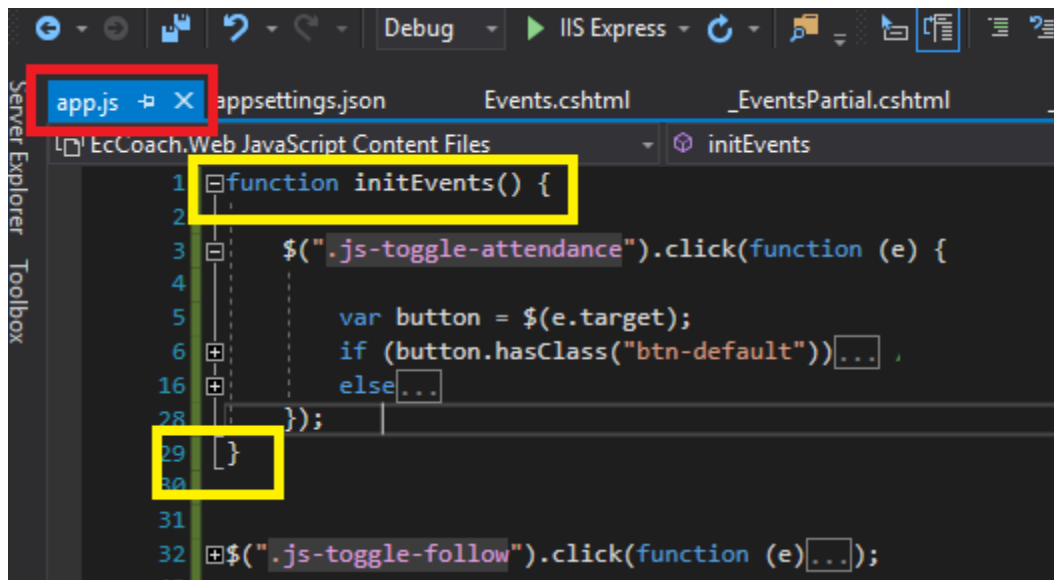
También un fallo y es que No tenemos separación de conceptos porque en nuestro script hay una parte que tiene que ver con trabajar sobre el DOM o manipular sus elementos y otra sobre trabajar con el server y tenemos todo junto, esto es lo que se conoce como código espagueti. El cual consiste en una maraña de cosas que hace muchísimas cosas dentro de la misma cosa. ¿entendiste algo de lo que acabo de escribir? No, ¿verdad? Eso mero pasa con el código espagueti, el que lo escribe lo entiende perfectamente, pero solo Dios y el y al tiempo solo Dios, porque él

para que no se le olvida porque rayos tiró ese código todo mezclado. El código espagueti hace difícil el mantenimiento futuro.

Primero hay que separar el js y moverlo a un archivo, para esto creamos una nueva carpeta llamada app, esto en el folder wwwroot, en de la carpeta js, la razón de ponerlo en un folder llamado app, es para separar de otros módulos que nosotros vayamos a tener que sirvan para otra cosa, es un tema de organización, pero ya ahí no es mala práctica ponerlo directo en el folder js.

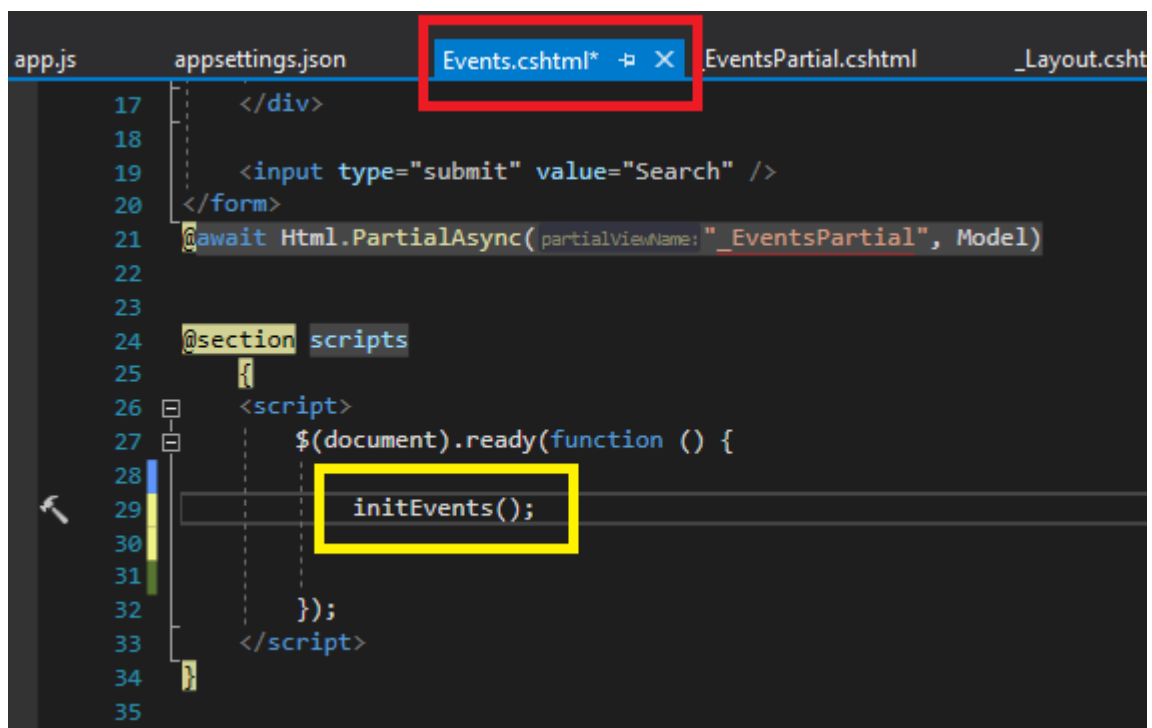
En nuestra carpeta app, creamos un archivo de JavaScript llamado "app.js" donde lo primero que vamos a hacer es poner aquí todo los js (que están dentro del ready) y después hacemos algo más limpio dividiéndolo como Carlos manda.

Una vez tenemos el js en nuestro archivo, debemos iniciar con la refactorización y lo primero que vamos a hacer es provocar que el llamado y ligadura de los elementos a los js, se hagan de la forma correcta, por lo que vamos a crear una función que llamaremos "initEvents" y dentro voy a pegar el código del "toggle attendance" y después iniciar esa función desde el ready de mi Events.cshtml



```
1 function initEvents() {
2
3     $(".js-toggle-attendance").click(function (e) {
4
5         var button = $(e.target);
6         if (button.hasClass("btn-default"))...
7     }
8 }
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32 $(".js-toggle-follow").click(function (e)...
```

No se nos puede olvidar cargar esa referencia en nuestro layout



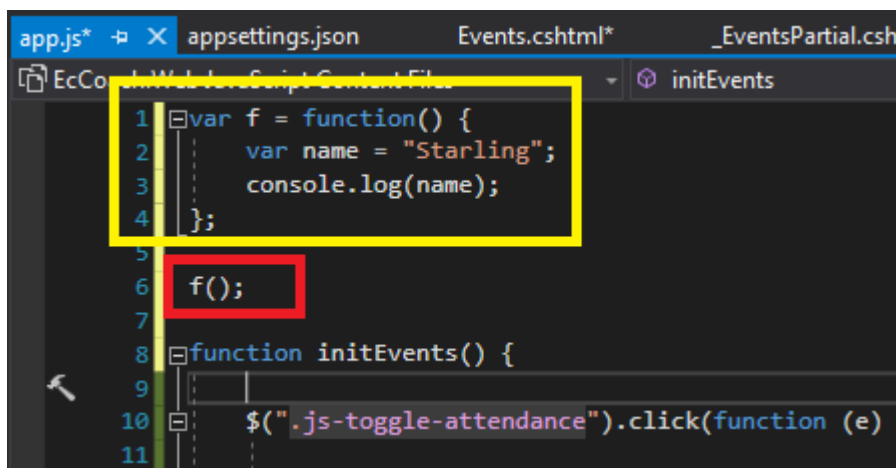
```
17 </div>
18
19 <input type="submit" value="Search" />
20 </form>
21 @await Html.PartialAsync(partialViewName: "_EventsPartial", Model)
22
23
24 @section scripts
25 {
26 <script>
27 $(document).ready(function () {
28
29     initEvents();
30
31 });
32 </script>
33
34
35
```

Perfecto. Definimos nuestra función, pero tenemos un problema y es que está en el NameSpace global de JavaScript y a medida que el código crezca la probabilidad de que dos funciones terminen llamándose igual y confligen (choquen) entre ellas, esto es lo que

se conoce como contaminación del NameSpace global, es también la razón de porque jQuery murió, desde que angular comenzó a chocar con él. Si vamos a escribir todo este código en el espacio global de la aplicación, entonces nosotros querríamos darle una estructura.

Queremos modularizar como en C# (o cualquier lenguaje moderno decente) que tenemos NameSpace, Clases y Métodos, en JavaScript no tenemos el concepto de clases como lo hacemos en C# pero hay una especie de sintaxis o un patrón si lo prefieres llamado Module Pattern.

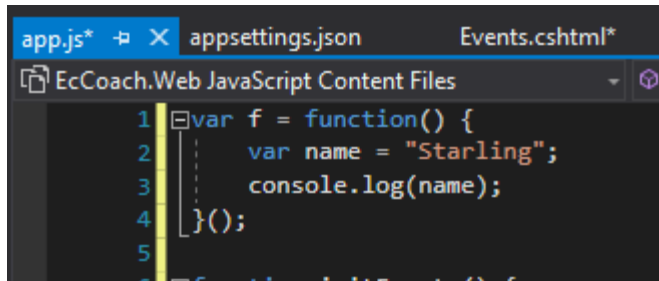
Supón lo siguiente, creo una variable como f y le asigno una función anónima donde dentro tengo una variable local que solo tiene vida dentro de la función anónima que crea esa variable, y luego para poder reflejar en valor que tiene esa variable tengo que invocarla como si fuera un método por ende invoco a f() y este se va a encargar de escribir en el log mi nombre; eso es algo que ya sabemos, f es un puntero a una función anónima.



```
app.js*  appsettings.json  Events.cshtml*  _EventsPartial.csh
EcCo  Web JavaScript Content File  initEvents
1  var f = function() {
2      var name = "Starling";
3      console.log(name);
4  };
5
6  f();
7
8  function initEvents() {
9
10     $(".js-toggle-attendance").click(function (e)
11
```

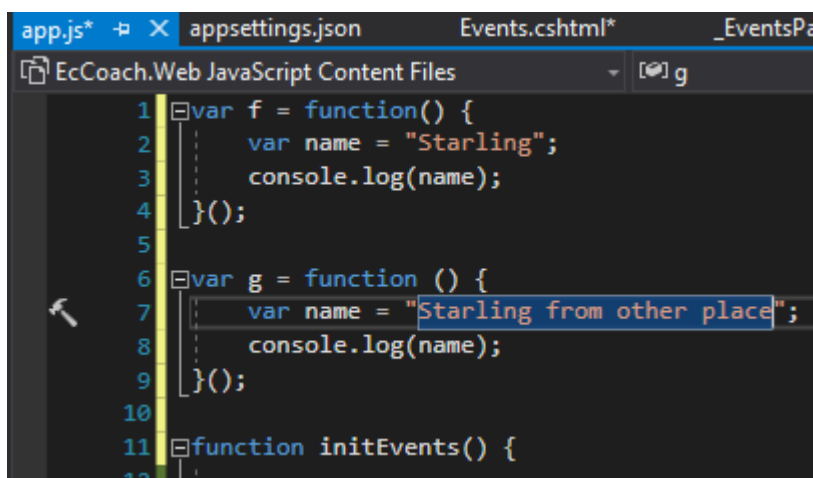
Puedo hacer este código más corto combinando la función y técnicamente convirtiendo la creación de mi variable en una

función. Creando algo llamado “**Immediately Invoked Function Expression (IIFE)**” expresión de función auto invocada o que se ejecuta así misma.



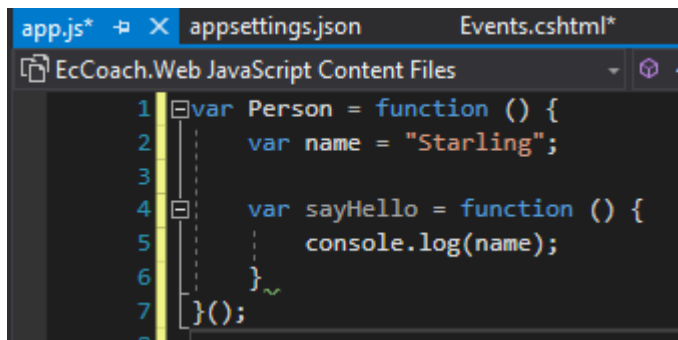
```
app.js* x appsettings.json Events.cshtml*
EcCoach.Web JavaScript Content Files
1 var f = function() {
2     var name = "Starling";
3     console.log(name);
4 }();
5
6 function initEvents() {
```

El beneficio de esta forma de trabajo es que podemos crear privacidad del código JavaScript. Y esto lo podemos usar para prevenir los conflictos del global NameSpace, en este ejemplo la variable “name” solo es accesible desde dentro de la función anónima, así es como creamos privacidad y podemos reusar nombres de variables a modo de ejemplo, si duplicamos el código podemos tener convivencia perfecta y reutilización de la variable “name” sin inconvenientes.



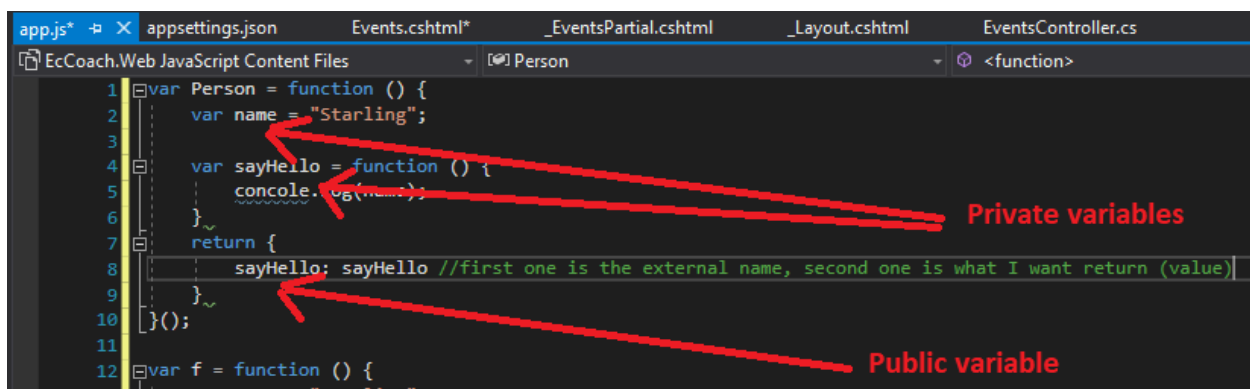
```
app.js* x appsettings.json Events.cshtml* _EventsPa
EcCoach.Web JavaScript Content Files [g]
1 var f = function() {
2     var name = "Starling";
3     console.log(name);
4 }();
5
6 var g = function () {
7     var name = "Starling from other place";
8     console.log(name);
9 }();
10
11 function initEvents() {
12
```

En JavaScript podemos crear una función dentro de la función, si, JavaScript es una locura.



```
1 var Person = function () {
2     var name = "Starling";
3
4     var sayHello = function () {
5         console.log(name);
6     }
7 }();
```

Esta función “sayHello” es interna de mi variable persona y solo es visible dentro de persona, si quiero exponerla fuera, entonces tengo que asignarle un Key (nombre) y ese key le puedo asignar lo que deseo que haga, ojo, el key no tiene que llamarse igual que la variable que deseamos retornar pero es bueno para inferencia de nombres.



```
1 var Person = function () {
2     var name = "Starling";
3
4     var sayHello = function () {
5         console.log(name);
6     }
7     return {
8         sayHello: sayHello //first one is the external name, second one is what I want return (value)
9     }
10 }();
11
12 var f = function () {
```

Entonces, todo esta explicación fue para poder entender lo que vamos a hacer ahora, voy a crear una especie de controlador en el cliente para empaquetar y proteger mis métodos, esto lo hago en el app.js.

```
var EventsController = function() {
}();

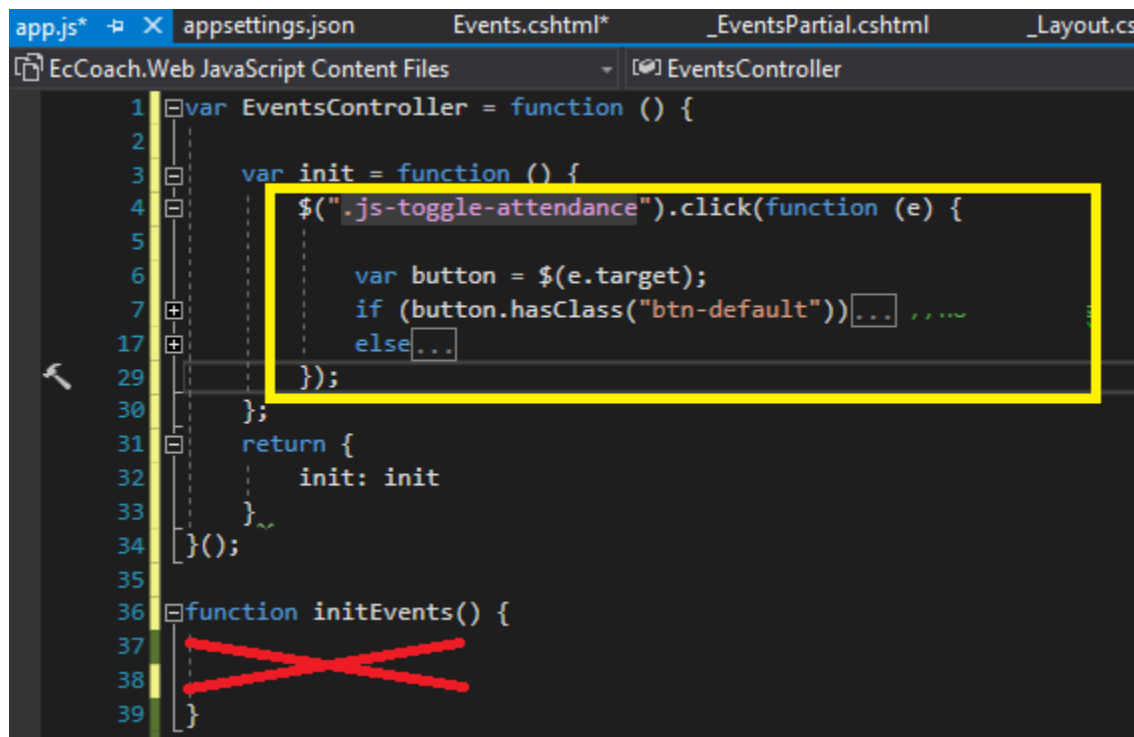
function initEvents() {

    $(".js-toggle-attendance").click(function (e) {
```

Dentro creamos mi método privado de inicialización y el valor que vamos a exponer (que vamos a llamar igual)

```
var EventsController = function () {  
    var init = function () {  
    };  
    return {  
        init: init  
    }  
}();
```

Cortamos el código de initEvents y lo movemos dentro de nuestro init que está en nuestro controlador, ya cuando lo tenemos así de organizado y bien trabajado podemos comenzar a romperlo. Ah y que no se nos olvide llamarlo desde el ready de la forma correcta.



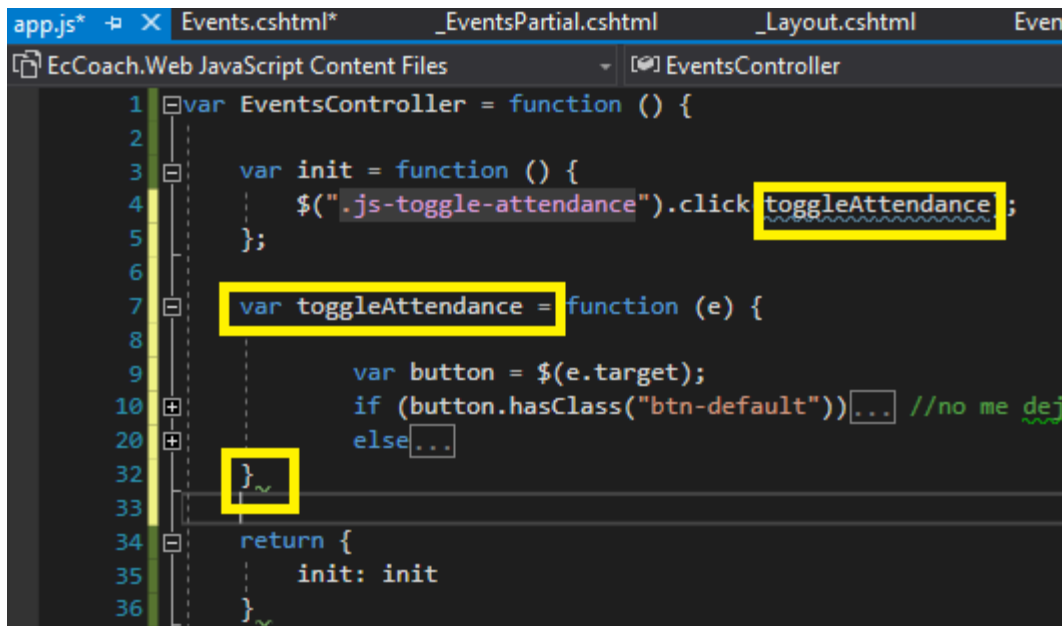
```
Events.cshtml*  _EventsPartial.cshtml  _Layout.cshtml  Events
17  </div>
18
19  <input type="submit" value="Search" />
20 </form>
21 @await Html.PartialAsync(partialViewName: "_EventsPartial", Model)
22
23
24 @section scripts
25 {
26 <script>
27     $(document).ready(function () {
28
29         initEvents();
30
31         EventsController.init();
32
33     });
34 </script>
35
36
```

Ahora bien, ya estamos avanzando hacia convertirnos en unos mejores desarrolladores, pero, este método tiene muchas cosas.

Por un tema de separación de conceptos, mi Js debería lucir algo como esto.

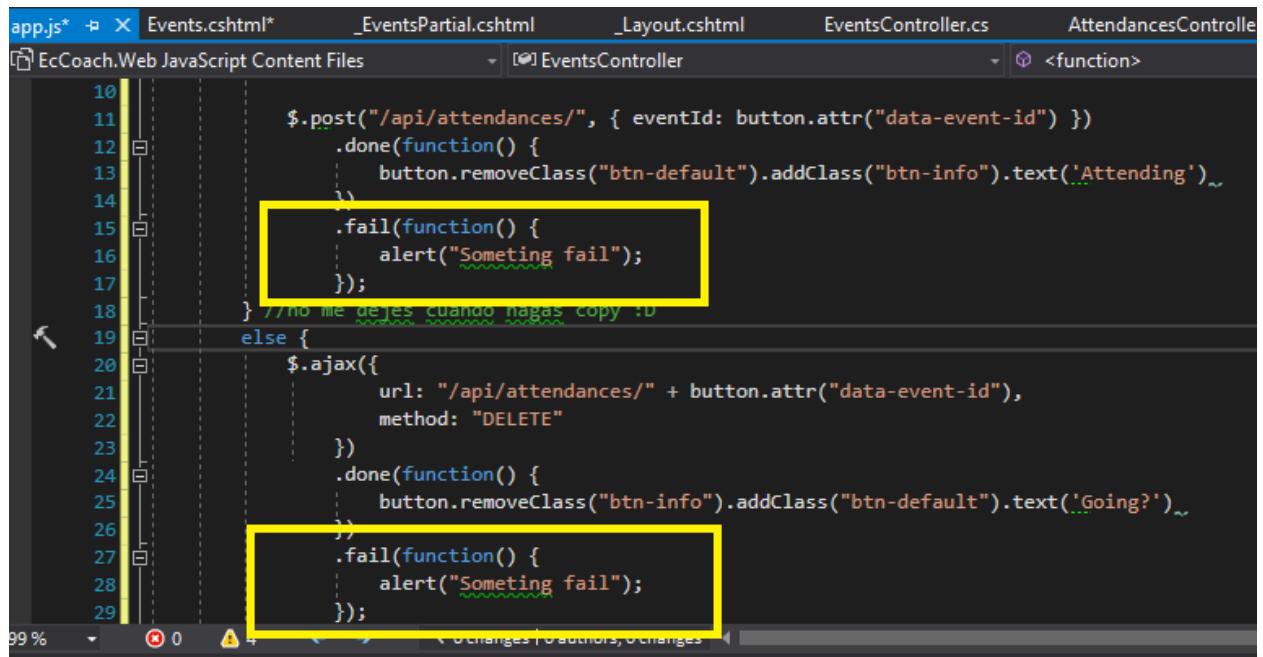
```
app.js*  Events.cshtml*  _EventsPartial.cshtml  _Layout.cshtml  EventsC
EcCoach.Web JavaScript Content Files  <global>
1  var EventsController = function () {
2
3      var init = function () {
4          $(".js-toggle-attendance").click(function (e){...});
5      };
6      return {
7          init: init
8      };
9  }();
10
11
12
```


Y ustedes dirán, ¿Por qué?, lo que sea que pase cuando yo haga clic en el elemento, debería ser irrelevante para el método específico, lo único que yo debo conocer es que estoy creando un Handler para un evento, así que creamos una función y reemplazamos.



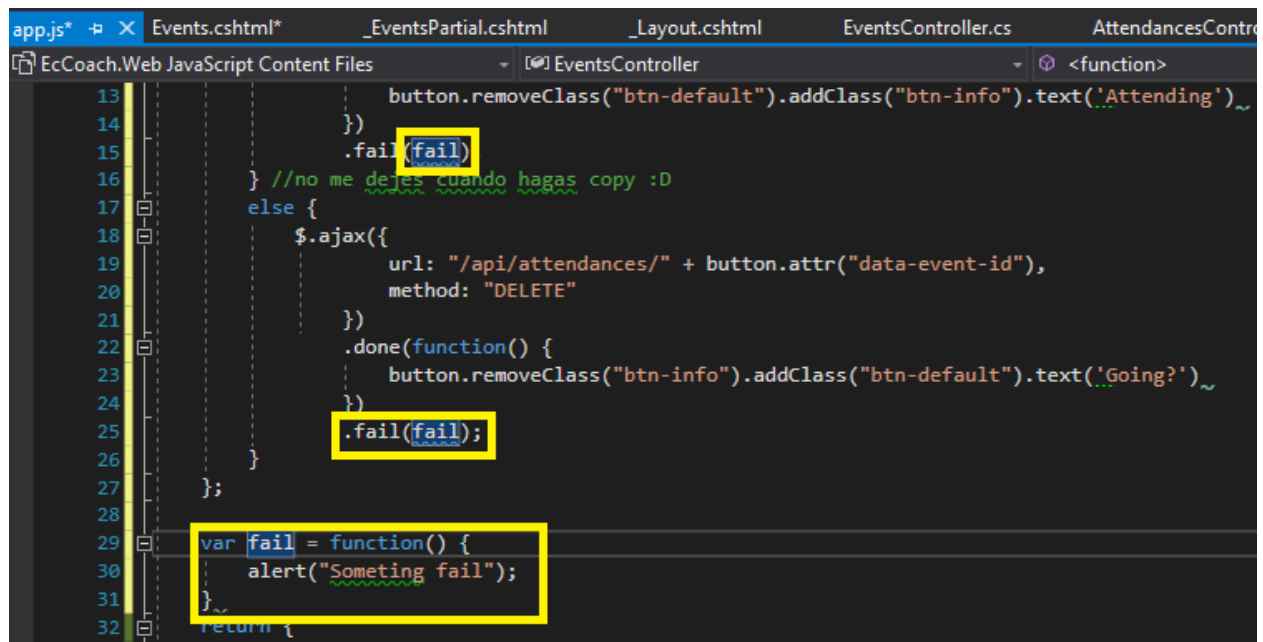
```
1 var EventsController = function () {
2
3   var init = function () {
4     $(".js-toggle-attendance").click(toggleAttendance);
5   };
6
7   var toggleAttendance = function (e) {
8
9     var button = $(e.target);
10    if (button.hasClass("btn-default"))... //no me dej
20    else...
32  }
33
34  return {
35    init: init
36  }
```

Ahora tenemos una función un poco más limpia, pero, hay cosas que me siguen molestando (en verdad no, solo finjo enojo porque tengo que enseñarles a enojarse de ver chapucerías), aquí en la imagen podemos ver dos alert que hacen exactamente lo mismo.



```
app.js* x Events.cshtml* _EventsPartial.cshtml _Layout.cshtml EventsController.cs AttendancesControlle
EcCoach.Web JavaScript Content Files [0] EventsController <function>
10
11 $.post("/api/attendances/", { eventId: button.attr("data-event-id") })
12 .done(function() {
13     button.removeClass("btn-default").addClass("btn-info").text('Attending')
14 })
15 .fail(function() {
16     alert("Someting fail");
17 });
18 } //no me dejes cuando hagas copy :D
19 else {
20     $.ajax({
21         url: "/api/attendances/" + button.attr("data-event-id"),
22         method: "DELETE"
23     })
24     .done(function() {
25         button.removeClass("btn-info").addClass("btn-default").text('Going?')
26     })
27     .fail(function() {
28         alert("Someting fail");
29     });
30 }
```

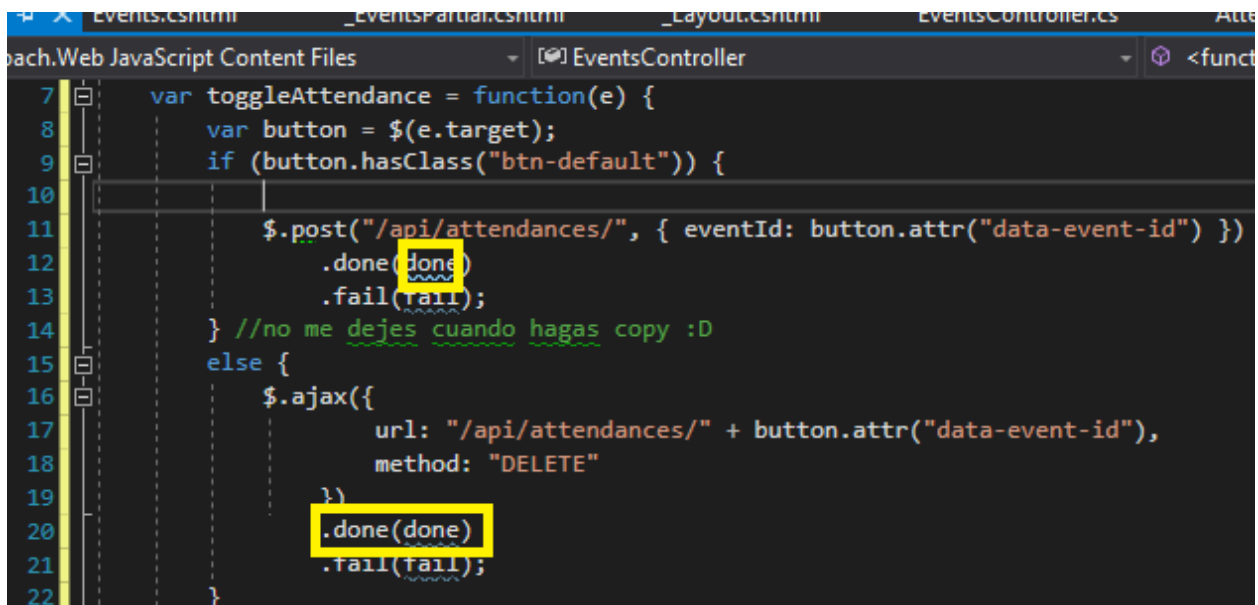
Por ende, como buenos desarrolladores que somos, entonces estamos en la obligación y tenemos el deber moral de corregir eso, en este caso lo haremos fácil, extrayendo esa funcionalidad en una función.



```
app.js* x Events.cshtml* _EventsPartial.cshtml _Layout.cshtml EventsController.cs AttendancesControlle
EcCoach.Web JavaScript Content Files [0] EventsController <function>
13     button.removeClass("btn-default").addClass("btn-info").text('Attending')
14 }
15 .fail(fail)
16 } //no me dejes cuando hagas copy :D
17 else {
18     $.ajax({
19         url: "/api/attendances/" + button.attr("data-event-id"),
20         method: "DELETE"
21     })
22     .done(function() {
23         button.removeClass("btn-info").addClass("btn-default").text('Going?')
24     })
25     .fail(fail);
26 }
27 };
28
29 var fail = function() {
30     alert("Someting fail");
31 }
32 return {
33     ...
34 }
```

Hacemos lo propio con el escenario del “done” pero como nosotros tenemos que ejecutar algo dependiendo de una condición, se

complica un poco las cosas, y lo “lógico” sería dejarlo aparte, pero no, nosotros somos buenos desarrolladores y vamos a hacer las cosas bien, identificamos que es lo único que diferencia un done del otro done y veremos que es solo una pinche condición, si el texto del botón que desencadena este evento tiene el texto “Going” entonces debe cambiar a “Going?” y en caso contrario significa que dice “Going?” por ende debe cambiar a “Going”, por consiguiente, tendremos algo como lo que sigue a continuación.




```
7 var toggleAttendance = function(e) {
8   var button = $(e.target);
9   if (button.hasClass("btn-default")) {
10
11     $.post("/api/attendances/", { eventId: button.attr("data-event-id") })
12       .done(done)
13       .fail(fail);
14     //no me dejes cuando hagas copy :D
15   } else {
16     $.ajax({
17       url: "/api/attendances/" + button.attr("data-event-id"),
18       method: "DELETE"
19     })
20       .done(done)
21       .fail(fail);
22   }
```

```
$.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);
}
var fail = function() {
```

Con esta implementación garantizamos que si tiene una clase se le asigne la otra, sin embargo, hay un fallo con esa implementación y es que nosotros no tenemos referencia al botón, una forma sería pasárselo como referencia.



```
};  
  
var toggleAttendance = function(e) {  
  var button = $(e.target);  
  if (button.hasClass("btn-default")) {  
    $.post("/api/attendances/", { eventId: button.attr("data-event-id") })  
      .done(done(button))  
      .fail(fail);  
  } //no me dejes cuando hagas copy :D  
  else {  
    $.ajax({  
      url: "/api/attendances/" + button.attr("data-event-id"),  
      method: "DELETE"  
    })  
      .done(done(button))  
      .fail(fail);  
  }  
};  
  
var done = function (button) {  
  var textToDisplay = (button.text() == "Going") ? "Going?" : "Going";  
  button.toggleClass("btn-info").toggleClass("btn-default").text(textToDisplay);  
}  
var fail = function() {
```

The image shows a code editor with two functions. The `toggleAttendance` function calls `done(button)` and `fail`. The `done` function is defined below and takes a `button` parameter. Blue arrows highlight the `button` variable in the `toggleAttendance` function and the `button` parameter in the `done` function, illustrating the flow of the reference.

Pero no es una buena práctica hacer esta salvajada, además si nosotros hacemos esto, no estaríamos pasando como referencia el botón al método, sino que estaríamos llamándolo a él mismo, es decir, el botón que tiene el método es quien se estaría llamando a sí mismo y no a una función genérica estándar que sea compatible con otros botones si así lo deseara y reutilizable, por lo cual la forma más “correcta” de implementarlo sería creando una función anónima a la cual le pasemos por referencia el botón algo así.

```
app.js*  Events.cshtml  _EventsPartial.cshtml  _Layout.cshtml  EventsController.cs  Attendance
EcCoach.Web JavaScript Content Files  EventsController <function>

7  var toggleAttendance = function(e) {
8      var button = $(e.target);
9      if (button.hasClass("btn-default")) {
10
11          $.post("/api/attendances/", { eventId: button.attr("data-event-id") })
12              .done(done(button))
13              .fail(fail);
14      } //no me dejes cuando hagas copy :D
15      else {
16          $.ajax({
17              url: "/api/attendances/" + button.attr("data-event-id"),
18              method: "DELETE",
19          })
20              .done(function() { done(button); })
21              .fail(fail);
22      }
23  };
24
25  var done = function ( button ) {
26      var textToDisplay = (button.text() == "Going") ? "Going?" : "Going";
27      button.toggleClass("btn-info").toggleClass("btn-default").text(textToDisplay);
28  }
```

Pero esto es una forma muy fea de hacer las cosas. La forma más limpia y orientada a objetos es mantener un track en todo momento del elemento y su estado dentro de esta vista y como estamos dentro de lo que viene a ser una especie de clase (modulo), todos los métodos privados pueden tener acceso a esa referencia y no vamos a tener problema con esto. Por lo que la variable local "button" la promovemos al global NameSpace al nivel del módulo, como si fuera un campo de una clase en C# una variable llamada botón y luego solo pasamos el valor en donde antes lo declarábamos.

```
app.js x Events.cshtml _EventsPartial.cshtml _Layout.cshtml EventsController.cs Attendance
EcCoach.Web JavaScript Content Files EventsController <function>

1 var EventsController = function () {
2
3     var button;
4
5     var init = function () {
6         $(".js-toggle-attendance").click(toggleAttendance);
7     };
8
9     var toggleAttendance = function (e) {
10         button = $(e.target);
11         if (button.hasClass("btn-default")) {
12             $.post("/api/attendances/", { eventId: button.attr("data-event-id") })
13                 .done(done(button))
14                 .fail(fail);
15             //no me dejes cuando hagas copy :D
16         } else {
17             $.ajax({
18                 url: "/api/attendances/" + button.attr("data-event-id"),
19                 method: "DELETE"
20             })
21                 .done(done)
22                 .fail(fail);
23         }
24     };
25
26     var done = function (x) {
27         var textToDisplay = (button.text() == "Going") ? "Going?" : "Going";
28         button.toggleClass("btn-info").toggleClass("btn-default").text(textToDisplay);
29     };
30 }
```

La movemos al principio de la clase porque es una buena práctica tener todas las variables al inicio y después las funciones.