Eliminando la Participación en un Evento

Nosotros tenemos implementado que si yo (usuario) no voy a asistir a un evento puedo hacer clic en la opción “Going” y cambia el botón, además de que este siempre carga en el estado correcto, pero no tenemos la parte inversa, para eliminar una asistencia así que vamos a resolver ese impase en la brevedad.

En el script que tenemos en la vista “Events.cshtml” nos damos cuenta que tenemos siempre la asunción de que tenemos que crear, pero no siempre es correcto, por lo que tenemos que validar que si tiene una clase default haga el créate, pero, en caso contrario llame una nueva API que en este caso sería de eliminar la asistencia al evento.

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

    var button = $(e.target);
    if (button.hasClass("btn-default")) {

      $.post("/api/attendances/", { eventId: button.attr("data-event-id") })
              .done(function () {
                  button.removeClass("btn-default").addClass("btn-info").text('Attending')
              })
              .fail(function () {
                  alert("Someting fail");
              }); //Faltaba este punto y coma :’(
     }//No me dejes cuando hagas copy :D
});
```

En el caso contrario entonces significa que tiene la clase info, por lo que no tenemos que preguntar y podemos ir directamente a un else donde copypasteamos el método del apartado anterior y hacemos un par de modificaciones.

La primera es que en lugar de un post vamos a hacer una llamada ajax, donde el método va a ser DELETE, luego, en caso de que sea satisfactorio en lugar de remover la clase default la que removemos es la info, le añadimos la clase default y cambiamos el texto por la pregunta “Going?”

```
else {
      $.ajax({
          url: "/api/attendances/" + button.attr("data-event-id"),
          method: "DELETE"
      })
        .done(function () {
           button.removeClass("btn-info").addClass("btn-default").text('Going?')
        })
        .fail(function () {
           alert("Someting fail");
        });
      }
```

Este código esta grande y feo, pero tranquilos que después lo vamos a arreglar, por ahora concentrémonos en que funcione.

Creamos una acción en AttendancesController llamada DeleteAttendance que recibe un id como parámetro, la marcamos con un Request de Delete, para que funcione como tal, obtenemos el usuario logueado, localizamos la asistencia de este, si la asistencia es nula retornamos NotFound, en caso contrario la borramos, guaramos y retornamos ok con el id que eliminamos. Quedando algo como esto.

```
[HttpDelete]
public IActionResult DeleteAttendance(int id)
{
    var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);
    var attendance = _context.Attendances
        .SingleOrDefault(a => a.AttendeeId == userId && a.EventId == id);
    if (attendance == null)
        return NotFound();
    _context.Attendances.Remove(attendance);
    _context.SaveChanges();
    return Ok(id);
}
```

Probamos y verificamos que todo esté funcionando como se supone que deba funcionar. Ya tenemos la aplicación funcionando como debería ahora es momento de refactorizar para que digan que nosotros sabemos programar, aunque sea mentira :D.

Resumen

En este módulo terminamos la implementación de los casos de uso restantes, conocimos como evitar los Magic String dentro de nuestro controlador, echamos un vistazo a la potencia de los LookUps y sentamos las bases para los procesos venideros de remasterización, nos vemos en el próximo modulo.