**CAPÍTULO 7**

# **IMPLEMENTACIÓN**

Ahora vamos a pasar a la fase de implementación del proyecto. En esta fase van a intervenir varias herramientas, las cuales son: Visual Studio (como se ha mencionado anteriormente), SQL Manager y Postman. Gracias a SQL Manager puede visualizar el estado de las tablas de la base de datos y mediante Postman puedo comprobar la viabilidad de las llamadas y peticiones a la API.

Para la implementación se ha dividido el proyecto en dos partes: la parte de Front-End y la parte de Back-End. Cada una de estas partes tiene un lenguaje determinado. La parte del Back-End estará escrita en C# y para la parte del Front-End, se utilizará el Framework React, el cual está basado en Type-Script, que es un lenguaje similar a Java Script, pero con la imposición de tipos en las variables. La arquitectura que se ha seguido para el desarrollo de la aplicación se ve en (3 pág. 64) para desarrollar una arquitectura en N capas.

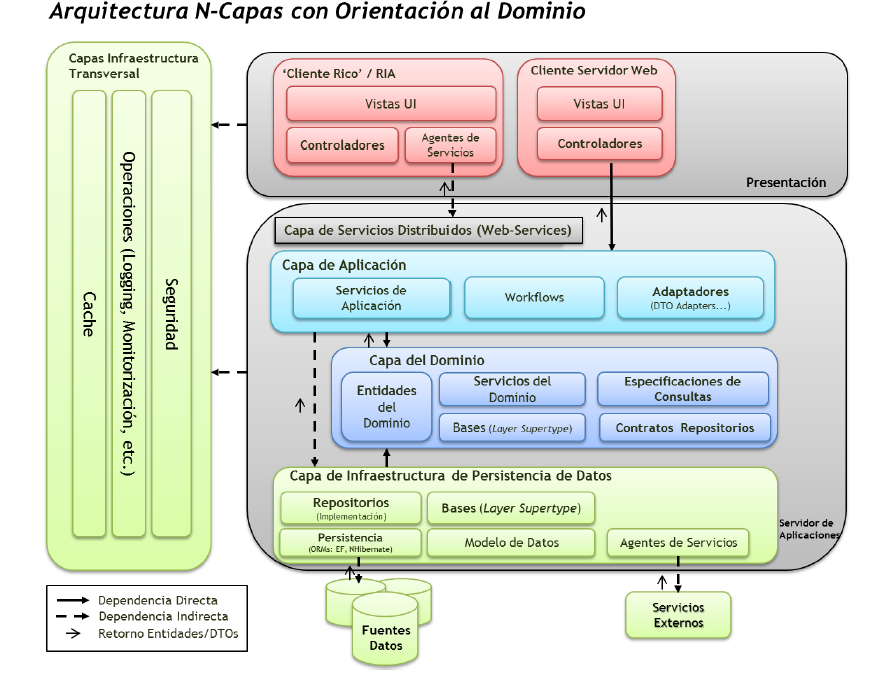


Ilustración 16. Arquitectura en N-Capas con orientación al Dominio, pág. 64.

Este esquema una vez traducido a nuestra arquitectura dentro de Visual Studio quedaría tal y como se ve en la siguiente ilustración.

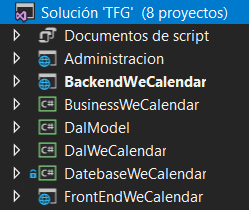


Ilustración 17. Ficheros del proyecto en Visual Studio.

Quedando las capas “Administración” y “Front-End” como clientes. La parte “Back-End” como capa de aplicación, “Business” como la capa de negocio y “DAL” como la capa de acceso a datos. Finalmente tenemos la “DalModel” en la que encontraremos las clases en C# las clases asociadas a la base de datos.

## **7.1 Back-End**

En este caso, nuestro Back-End constará de una capa DAL (Data Access Layer), la cual se encargará exclusivamente de realizar las consultas en la base de datos. Una capa Business, la cual se encargará de llevar a cabo la lógica en el servidor. Finalmente, tendremos una API Rest, la cual se comunicará con la capa de negocio para realizar los cambios en la base de datos. A esta API se accederá mediante las URLs desde las distintas pestañas del Front-End.

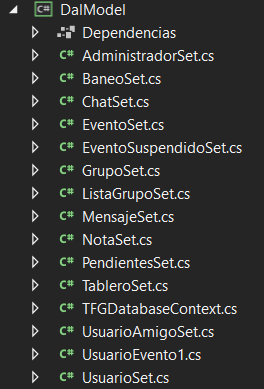
Anteriormente se ha mostrado un diagrama de UML de la que será la base de datos del sistema. Este diagrama puede transcribirse dentro de Visual Studio para que posteriormente sea la aplicación la encargada de generar las tablas de la base de datos partiendo de este diagrama pudiendo editarse visualmente. A partir de la base de datos Visual Studio creará una estructura de clases asociada.

Ilustración 18. Clases en Visual Studio pertenecientes a la base de datos.



Ilustración 19. Diagrama generado por Visual Studio.

Aquí se pueden apreciar las tablas y sus relaciones que posteriormente serán traducidas por la misma herramienta a las sentencias de SQL necesarias para generar la base de datos completa. Una vez que se ha generado la base de datos debemos de actualizar el modelo que tenemos en la capa “DalModel” del proyecto para que la estructura de las clases quede acorde con la estructura de la base de datos. Una vez actualizamos el modelo en la aplicación se genera la estructura vista en la *ilustración 18.*

API

En la capa del Back-End encontramos la API. Gracias a esto podemos comunicar las funciones del Back-End con el Front-End. En este caso tendremos la API dividida según su funcionalidad. Siendo estas divisiones denominadas “Controller”. En este caso veremos un “UsersController” para las operaciones de los usuarios, “EventsController” para las funciones y llamadas procedentes de los eventos, “NotasController” para las funciones derivadas de la administración de las notas y “ChatController” para todas las funciones referentes a los mensajes y a la sala de chat.

Ahora vamos a ver como se realizan las llamadas de las funciones más relevantes dentro de cada sección de la API.

**UsersController**

|  |  |  |
| --- | --- | --- |
| Operación | Argumentos | Respuesta |
| get | nombre | Datos referentes a usuario con el nombre especificado. |
| amigos | identificador | Devuelve la lista de amigos del usuario identificado. |
| listaBaneados |  | Devuelve la lista de los usuarios que se encuentren baneados en el sistema. |
| login | nombreUsuario, contraseña | Valida las credenciales de usuario y devuelve un booleano con la comprobación de los credenciales. |

**Ejemplo:**

llamada:

[**http://localhost:11111/api/Users/Afro**](http://localhost:11111/api/Users/Afro)

Respuesta:

{

"id": 1,

"nombreUsuario": "Afro",

"correo": "javier.fuentes78@gmail.com",

"notificacion": "YES",

"foto": "foto",

"createDate": "2018-07-03T10:02:32.503",

"token": "deea659a-c912-4f64-9f93-3ca477443b30",

}

**EventsController**

|  |  |  |
| --- | --- | --- |
| Operación | Argumentos | Respuesta |
| get | identificadorUsuario | Lista de eventos creados por el usuario identificado. |
| Invitación | identificadorUsuario | Lista de los eventos a los que ha sido invitado el usuario. |
| share | evento, usuario | Comparte el evento que se le ha suministrado como argumento al usuario que también se ha suministrado, devuelve un booleano en función de si ha podido o no compartir. |
| aceptarInvitacion | Usuario, evento | Crea el enlace entre el usuario y el evento, devuelve el booleano en función de si ha podido o no crearse el enlace. |

**Ejemplo:**

llamada:

**http://localhost:11111/api/events/1**

Respuesta:

[

{

"id": 99,

"nombre": "Conferencia de programación",

"descripcion": "Charla sobre desarrollo en C#",

"direccion": "ETSIIT",

"horaInicio": "13:30:00",

"horaFin": "14:00:00",

"fecha": "2018-08-04T",

"prioridad": 1,

"visibilidad": false,

"createDate": "2018-08-04T09:13:12.06",

"usuarioId": 1,

}, …

]

**NotasController**

|  |  |  |
| --- | --- | --- |
| Operación | Argumentos | Respuesta |
| get | identificador | Devuelve las notas asociadas al identificador del usuario suministrado como argumento. |
| grupo | grupo | Devuelve las notas asociadas al grupo suministrado como argumento. |
| post | título, cuerpo, fechaTope | Se crea una nueva nota con el contenido suministrado como parámetro y devuelve un booleano en que refleja la realización de la tarea. |

**Ejemplo:**

llamada:

**http://localhost:11111/api/Notas/grupo/1**

Respuesta:

[

{

"id": 22,

"titulo": "Demo",

"texto": "texto de prueba",

"fechaTope": "2018-08-03T15:58:12.407",

"createDate": "2018-08-03T15:58:12.407",

}, …

]

**ChatController**

|  |  |  |
| --- | --- | --- |
| Operación | Argumentos | Respuesta |
| get | usuario | Devuelve la lista de mensajes que han sido redactados por el usuario suministrado. |
| put | usuario, grupo, texto, hora | Añade un mensaje a la sala de chat del grupo especificado identificado con el usuario, con el texto y la hora suministrados. Devuelve un booleano que representa el éxito de la operación. |
| newUserGrup | usuario, grupo | Añade a un usuario al grupo suministrado por parámetro, devuelve un booleano que representa el éxito de la operación. |
| newGrup | nombre, usuario | Genera un nuevo grupo con el nombre especificado y dejando como creador al usuario suministrado por parámetro, devuelve un booleano con el éxito de la operación. |

**Ejemplo:**

llamada:

**http://localhost:11111/api/Chat/1**

Respuesta:

[

{

"id": 1,

"texto": "Hola",

"grupoId": 1,

"createDate": "26/08/2018 11:18:17",

"usuarioId": 1,

"chatId": 1,

}, …

]

## **7.2 Front-End**

Ahora vamos a abordar la parte del Front-End. En esta parte, como he mencionado anteriormente, utilizaremos el Framework React. He elegido este Framework, en primer lugar, por su integración con Visual Studio, y en segundo lugar por su tratamiento con Ajax, ya que, en este caso, React integra a Ajax de forma nativa. Con esto y el uso de las variables de estado de la plantilla, se consigue un efecto de página estática en la que se están produciendo llamadas a la API y continuas modificaciones en los datos sin que el usuario perciba que la página se recarga. Para las plantillas de estilo de la página se ha optado por la opción de Boostrap, debido a su sencillez visual y a su característica responsiva. Es muy importante esta característica porque necesitamos que esta página sea accesible por todos los dispositivos posibles (ordenadores, tabletas, teléfonos móviles…) y es necesario que la página se redimensione correctamente en función del dispositivo que la esté utilizando para garantizar la mayor accesibilidad posible.

Uno de los inconvenientes que se han encontrado a la hora de implementar es la necesidad de depuración en local de la aplicación. Gracias a la herramienta Visual Studio, no necesitamos hacer subidas constantes de la aplicación para probar las nuevas funcionalidades implementadas, a pesar de tener la base de datos en el servidor. Para ayudar a la depuración se han establecido puertos fijos a la ejecución de Back-End y el Front-End.

|  |  |
| --- | --- |
| Ilustración 20. Configuración del Back-End (loaclhost:11111) | Ilustración 21. Configuración del Front-End (localhost:44444) |

Gracias a esto podemos establecer las direcciones de la API en un fichero a parte y cambiar entre local host y la dirección de Azure una vez queramos depurar o hacer un despliegue.

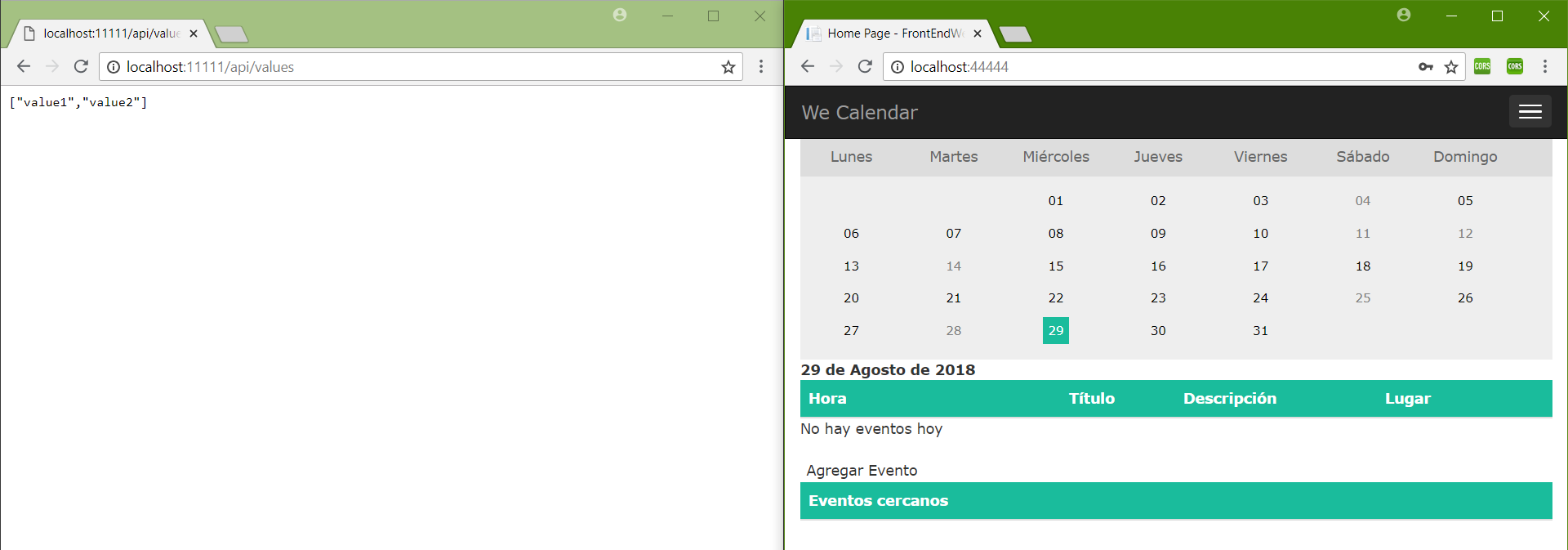


Ilustración 22. Ejecución del Back-End y el Front-End en local.

Se puede apreciar que en la página predominan los colores verde y azul esto es debido a lo que dice (4) *“el verde es el color que los ojos procesan mejor. Úsalo para crear un efecto relajante o de calma”* y también *“el azul suele asociarse con empresas grandes y bancos porque no es invasivo y se asocia con la seriedad”*. Con estos dos factores a tener en cuanta se ha decidido emplear una combinación de esos dos colores para transmitirle al usuario calma y organización ya que son dos atributos necesarios a la hora de realizar una correcta organización de los eventos.

En la depuración es donde encontramos un problema a la hora de hacer peticiones a la API. El problema es el control de acceso HTTP CORS (5), en este caso es necesario añadir una cabecera específica a nuestras rutas de la API ([EnableCors("AllowSpecificOrigin")]) para permitir que se realicen llamadas desde la aplicación en depuración. A parte de esta cabecera debemos de agregar las direcciones desde las que se accederá a la API al fichero *Startup.cs* que se encuentra dentro de esta.

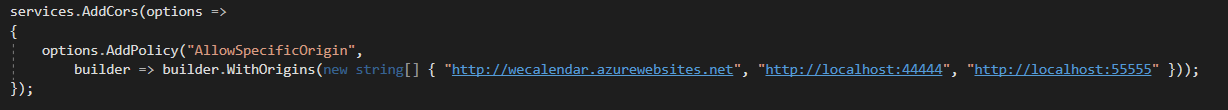


Ilustración 23. Código para permitir acceso en depuración a la API.

También es necesario agregar al navegador (Chrome en este caso ya que es el navegador que utiliza Visual Studio para la depuración) un complemento denominado Allow-Control-Allow-Origin para que el propio navegador también permita este acceso.

Para realizar las peticiones desde el Front-End a la API vamos a ayudarnos de la librería de Axios (6), librería compatible con los paquetes de React que nos ayudará a realizar peticiones a la API y recibir las respuestas con ayuda de las “Promesas”. Gracias a las promesas en las peticiones podemos especificar el formato en el que deseamos que se nos proporcione la respuesta. Esto hace que esa petición no acepte un formato distinto al que le proporcionemos, mitigando los errores en la transferencia de datos.

## **7.3 Envío de notificaciones**

También, en este apartado tenemos una funcionalidad que no depende de ninguna llamada por parte del Front-End. Esta funcionalidad se trata del envío de notificaciones, tal y como se indica en el caso de uso **CU-34**, todos los días deberá de enviarse una notificación a todos los usuarios que tengan las notificaciones activadas y que celebren un evento de prioridad alta ese mismo día. Esto se consigue gracias a la implantación de un demonio en la capa de negocio que, una vez lanzado repetirá periódicamente una vez al día las tareas de consulta de los usuarios mediante los criterios citados anteriormente y el envío de mensajes mediante correo electrónico gracias a una API externa de carácter comercial llamada Send Grid (7). La decisión de utilizar una API externa para el envío de mensajes viene de las limitaciones de los servidores de Azure de correos semanales, todo ello con la finalidad de evitar el SPAM.

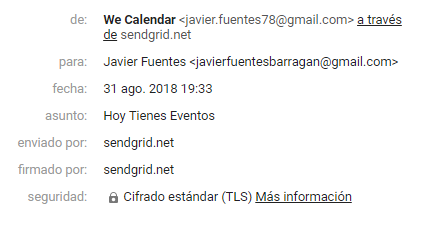


Ilustración 24. Correo Enviado desde Send Grid.

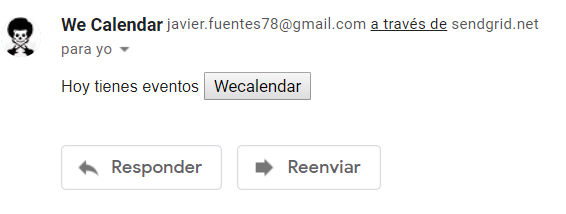


Ilustración 25. Contenido del correo.

Como se puede ver en el mensaje que se envíe con esta aplicación también podemos insertar código HTML para permitir hacer redirecciones a la página de una manera más limpia y sencilla para el usuario.

Gracias a esta aplicación garantizo no solo la seguridad en el envío de mensajes, sino también una posible ampliación del flujo de mensajes semanales en el caso en que esta aplicación creciera en número de usuarios, únicamente mediante el cambio de suscripción en Send Grid.

## **7.4 SEGURIDAD**

Los factores que se han tenido en cuenta en este apartado solos factores relacionados con los datos de seguridad del usuario. A continuación, hablaremos del uso de las claves cifradas para guardar las contraseñas, las consultas mediante Token para garantizar anonimato y el hecho de que el código del Back-End ha sido compilado en el servidor, por lo que lo único que encontraremos al acceder a él, serán los archivos ejecutables del mismo.

CIFRADO DE CONTRASEÑAS

En este apartado hablaremos de la necesidad del cifrado de las contraseñas. Todas las contraseñas de los usuarios guardadas en la base de datos están cifradas de manera simétrica, estableciéndole a todas las mismas dimensiones de caracteres con un cifrado de 16bits, haciendo muy difícil las violaciones de seguridad mediante ataques de fuerza bruta.

TOKEN

Todos los usuarios, como se ha visto anteriormente, tienen almacenados una variable llamada Token, la cual nos servirá para realizar consultas. Esta variable es única y gracias a una función de C# (8) se puede generar de una forma sencilla.

La dinámica del Token consiste en que cada vez que el usuario inicie sesión, el sistema tras validar sus credenciales asignará un nuevo Token distinto al usuario, haciendo que cada vez que éste inicie sesión, el campo de búsqueda cambie evitando así posibles ataques de suplantación de identidad.

COMPILACIÓN DEL BACK-END

Aunque esta característica se ha planteado antes como una desventaja a la hora de elegir C# como lenguaje, hay que destacar que es una protección contra posibles hackeos al servidor, ya que, si una persona lograse entrar en nuestro servidor de Azure, lo único que encontraría sería un ejecutable, dificultando así la obtención de datos del código fuente.

## **7.5 USABILIDAD**

Al tratarse este proyecto como uno principalmente orientado al usuario final, es necesario pensar y orientar todas las acciones que se realizan en la página para proporcionarle al usuario una experiencia fácil y cómoda.

Para ello se ha tratado de reducir lo máximo posible el número de pulsaciones que un usuario debe realizar en la página para terminar la acción deseada. Se han implementado buscadores para filtrar y reducir lo máximo posible el desplazamiento por la pantalla, agilizando las búsquedas.

Otra ayuda que se le suministra al usuario para ayudar al uso es la disposición de la pestaña de los eventos, la cual le proporcionará una vista general de toda su organización mensual con un solo vistazo, centrando su vista en el eje principal de la aplicación, que es el calendario, y minimizando las búsquedas de los eventos.

## **7.6 PERSISTENCIA**

En este caso el servidor de Azure que estamos utilizando para almacenar el Back-End y el Front-End se trata de una arquitectura *Serverless* (7) , gracias a este modelo podemos abstraernos de los servidores, sin preocuparnos del sistema operativo. Este modelo se basa en la reacción a los eventos en tiempo real en la nube. Las facturas derivadas de este servicio están relacionadas al tiempo que el código se encuentra en ejecución.

Gracias a este modelo se crea una estructura mucho más robusta, resistente a las amenazas y con una disponibilidad muy alta. A parte también tiene muchísima flexibilidad a la hora de realizar escalados. *“Un servicio Serverless puede empezar de nada hasta abarcar decenas de miles de funciones concurrentes casi al instante”*.

## **7.7 MANUAL DE USUARIO**

A continuación, se va a disponer un pequeño manual de usuario para poder entender mejor como se establecen las transiciones dentro de la página y como se realizarán correctamente todas las funciones.

**Inicio y registro**

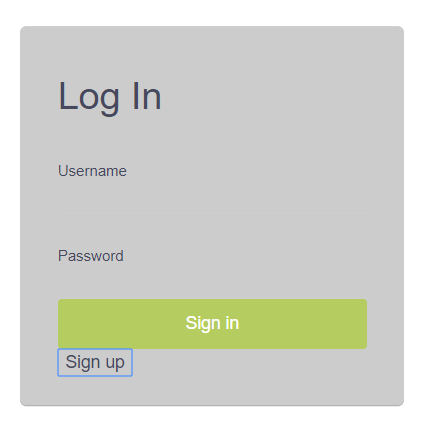
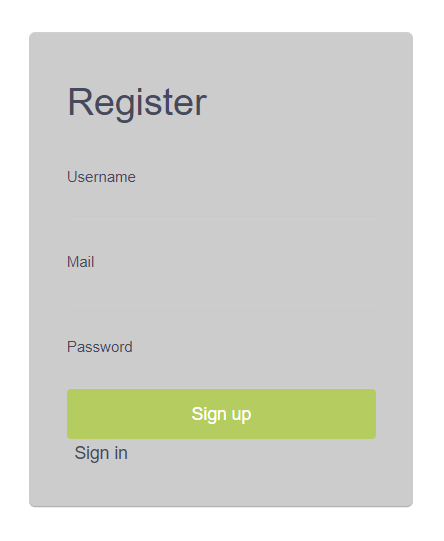


Ilustración 26. Pantalla de registro de usuario.

Ilustración 27. Pantalla de inicio de sesión.

Estas dos pantallas será las primeras con las que los usuarios tendrán contacto, tanto si el usuario es nuevo en el sistema y desea registrarse, como si el usuario ya se encuentra registrado en el sistema y lo que desea es acceder a él. El usuario una vez que se encuentre en cualquiera de estas dos plantillas podrá alternar entre estas mediante los botones *sign up* y *sign in*.

**Home**

Ilustración 28. Pantalla de Home.

Nada más iniciar sesión en la aplicación vamos a poder ver una pestaña con un menú lateral con todas las demás opciones de la aplicación.

Lo principal de esta pestaña es el calendario que aparece en el centro de la pestaña. Justo debajo del calendario se despliega una tabla con los eventos que tendrán lugar en el día seleccionado, que por defecto será el día actual. Se ha optado por este diseño del calendario para que el usuario tenga de un solo vistazo una visión global de la planificación del mes y a la vez pueda ver de una forma más específica cada uno de los eventos en el día.

En la tabla de los eventos podemos ver la información del evento correspondiente a la hora de inicio y fin del evento, su nombre y descripción y el lugar. Justo después tenemos dos botones para borrar o ver con más detalla el evento.

Justo debajo podemos ver también una pequeña tabla donde nos aparecerán los eventos más cercanos al día actual para tener presenta no solo lo que acontecerá ese día si no también, lo que pasará próximamente.

**Notas**

****

Ilustración 29. Pantalla de Notas.

Aquí podemos ver la pestaña de notas, se ha optado por un modelo de notas adhesivas para que se asemeje más a un tablero de notas casero. Dentro de esta pestaña se dispondrán todas las notas del usuario. Al pasar el ratón por encima de la nota esta aumentará ligeramente de tamaño y rotará ligeramente hasta enderezarse para que sea mucho más fácil de leer. La última nota que aparecerá en esta pestaña será el formulario para generar nuevas notas.

Se puede apreciar que hay notas con un color distinto de las demás, eso significa que esa nota ha sido compartida en algún grupo como podrá verse más adelante.

**Usuario**

****

Ilustración 30. Pantalla de Usuario.

En esta pantalla se puede ver la información referente al usuario que se encuentra registrado en el sistema en este momento. A parte de este perfil en el menú aparecerá el nombre del usuario. En esta pestaña se ve una imagen de perfil de usuario, su nombre y correo y la posibilidad de recibir las notificaciones, en esta última existe la opción de silenciarla si queremos para que el usuario ya no reciba más notificaciones al correo.

**Chat**



Ilustración 31. Pantalla de Chat.

En esta pestaña podemos ver la sala de chat que nos permitirá comunicarnos con el resto de los usuarios del sistema. Lo primero que nos encontramos son los mensajes de un grupo seleccionado. Estos mensajes disponen de una barra de scroll para poder ver todos los mensajes que se encuentren. Debajo de los mensajes tenemos la barra para escribir el mensaje el cual una vez enviado aparecerá con el nombre y la fecha en la que se envió.

A la derecha se puede ver la lista de los grupos, aquí se mostrarán todos los grupos en los que el usuario participa. Cuando el usuario pincha en alguno de los grupos se cambia el contenido del chat para así poder ver los mensajes correspondientes a dicho grupo.

Debajo de la barra del chat de puede apreciar otro tablero similar al tablero que había en la sección de notas. En este tablero se mostrarán las notas compartidas en el grupo seleccionado, las cuales hacen referencia a las notas azules de la pestaña de notas. También encontraremos un formulario para añadir una nota directamente al grupo seleccionado.

**Eventos**



Ilustración 32. Pantalla de Eventos.

Como se ha comentado anteriormente en el caso de uso **CU-20** los usuarios registrados en el sistema tendrán la posibilidad de ver los eventos públicos, los cuales aún no hayan terminado, y tras visualizarlos el usuario podrá marcar que desea asistir a ellos y este evento será añadido a su calendario. En este caso nos encontramos con una pestaña en la cual se dispondrán los eventos por orden mostrando los datos tales como el nombre, la descripción y el lugar y finalmente un botón para asistir. Se puede apreciar también, que hay una barra de búsqueda, la cual está justo encima de los eventos, esto hace referencia al **CU-21**.



Ilustración 33. Ejemplo de buscador.

Aquí les proporcionamos a los usuarios una barra de búsqueda para que puedan filtrar los eventos cuando la lista de los eventos públicos crezca mucho. Este filtro se activa al introducir cualquier parámetro de búsqueda en la barra del buscador inmediatamente filtrando las coincidencias como se ve en la *ilustración 32.*

**Gestión de eventos pendientes**

Todo lo mostrado anteriormente corresponde con el sistema en su estado normal en un momento en el que el usuario no haya recibido ninguna notificación de evento pendiente. Ahora vamos a ver las transiciones que un usuario debe hacer cuando ha sido invitado a un evento.

Primero, el usuario apreciará que en su barra de navegación aparece un elemento distinto. En este caso una alerta de que tiene un número de eventos pendientes, este número irá variando en función de la cantidad de eventos.

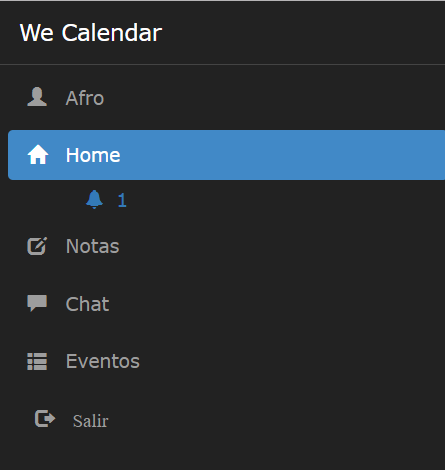


Ilustración 34. Notificación de evento pendiente.

Una vez el usuario se percate de esta notificación podrá pulsar sobre esta para ser redirigido a la pestaña de los eventos pendientes. En esta pestaña se le dispondrá un calendario en el que estarán marcadas las fechas de los eventos que están pendientes de validación.



Ilustración 35. Pantalla de eventos pendientes.

Como se puede ver en la *ilustración 19* solamente nos aparece un evento en la pantalla marcado en naranja. En este caso se han elegido dos tipos de colores para los eventos, el naranja para el evento pendiente que, si puede agregarse sin problema al calendario, y el rojo para el evento que no puede agregarse o es necesario realizar algún cambio en la planificación para que el evento pueda agregarse a nuestro calendario.

Una vez un evento pasa a estado pendiente pueden pasar varias cosas. Primero, podemos no querer asistir al evento, en ese caso podemos simplemente rechazarlo y este quedaría fuera de nuestra planificación y borrado de la lista de pendientes. Si deseamos añadir el evento entonces tenemos dos posibilidades, la primera es que esté disponible el margen de horas que precisa el evento, en ese caso bastará con aceptarlo y el evento quedará añadido. En el caso de que deseemos añadir el evento, pero no podamos el sistema nos proporcionará las herramientas para editar los eventos creados por nosotros o cancelar la asistencia a los elementos que hemos sido invitados. Una vez hecho esto, podremos aceptar el evento tranquilamente.

**Administración**

En la pestaña de administración vamos a tener un menú con dos opciones, usuarios y eventos. Como se ha mencionado antes la función del administrador será la de ver y eliminar los eventos de los usuarios y ver y banear a los usuarios del sistema. A continuación, se mostrará la pantalla correspondiente a esas funciones.

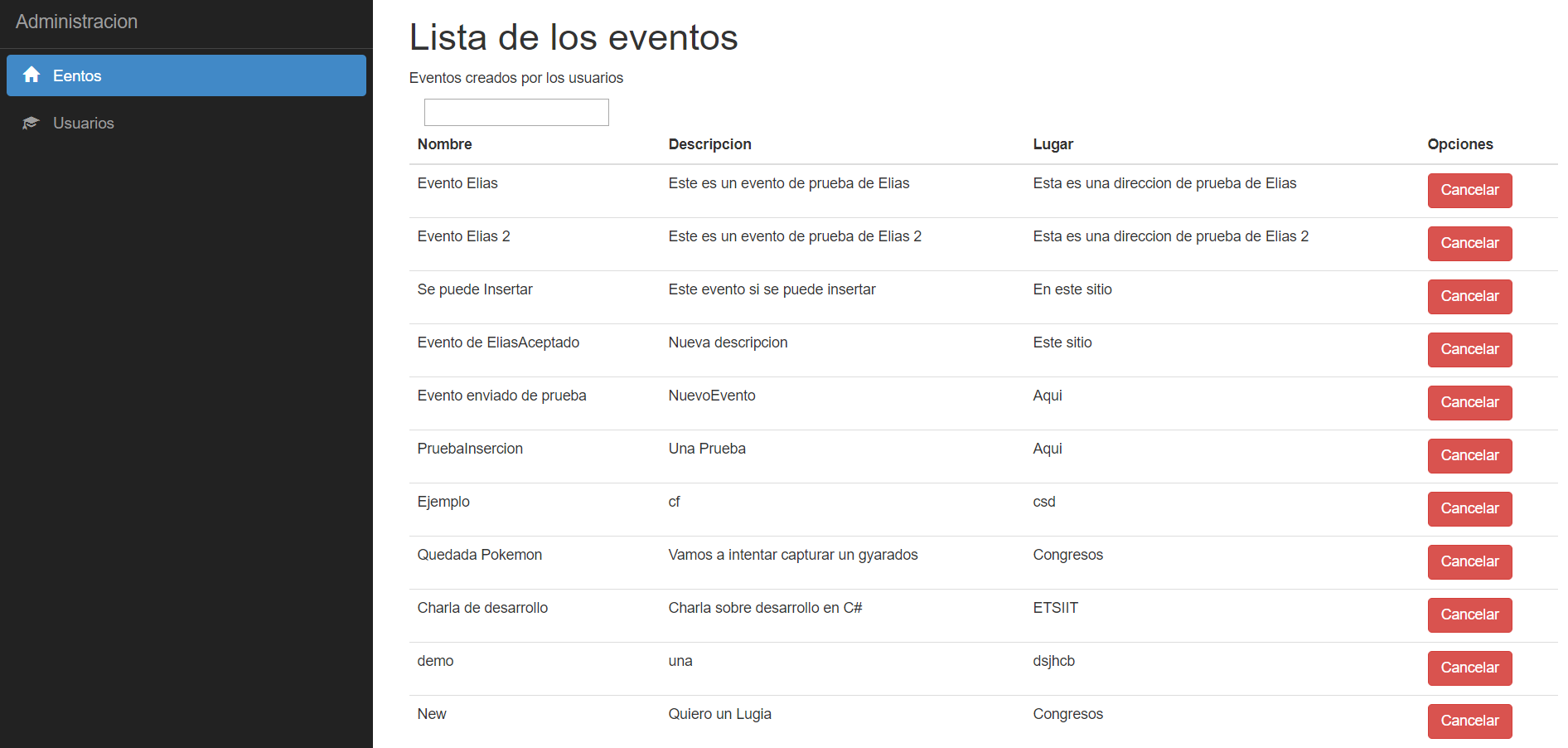


Ilustración 36. Pantalla de Administración de Eventos.

Aquí podemos ver la pantalla en la que los administradores podrán ver la lista de eventos del sistema. En la pantalla se observa una barra de búsqueda para facilitar la tarea del administrador y en cada uno de los eventos una opción de cancelar con la cual el evento quedará suspendido.

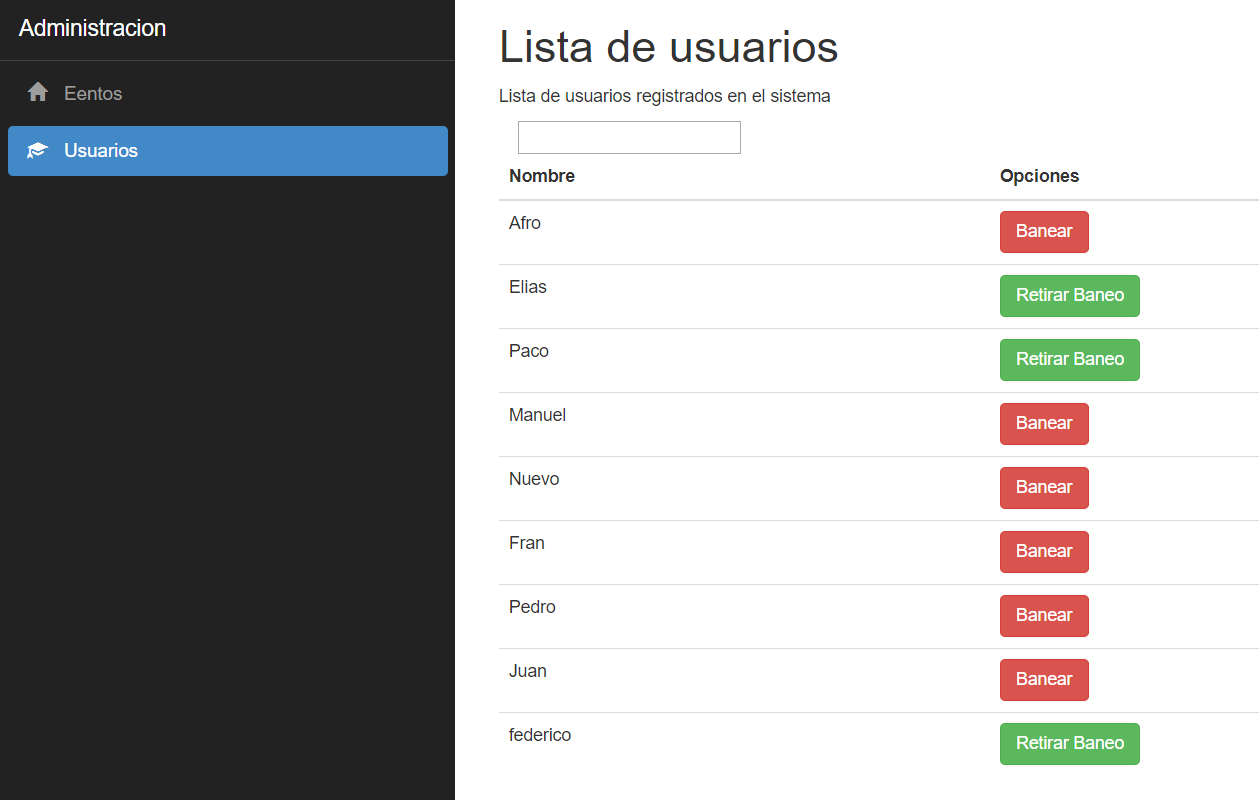


Ilustración 37. Pantalla de Administración de Usuarios.

En esta pantalla los administradores encontrarán una lista de los usuarios que se encuentren registrados en el sistema. Igual que en la pantalla anterior podemos ver una barra de búsqueda para filtrar los usuarios. Cada usuario tendrá la opción de baneo al lado de su nombre, una vez el usuario quede baneado se dispondrá de una opción de retirar baneo para devolverle el acceso al sistema.