salesianos

ATOCHA

Trabajo de Fin de Grado

Proyecto de geolocalización.

Pablo Ramírez San Miguel

Pablo López Muñiz-Alique

C.F.G.S. Administración de Sistemas Informáticos en Red

Contenido

1. Introducción y objetivos 3
   1.1. ¿Qué es la geolocalización y para qué sirve? 3
   1.2. Objetivos. 4
2. ¿A quién va dirigido? ¿Por qué es necesario su uso? 5
3. Lenguajes utilizados. 6
   3.1. HTML 6
   3.2. JavaScript 6
   3.3. CSS 6
   3.4. Bootstrap 7
4. Aprendizaje y herramientas o servicios utilizados. 8
   4.1. Firebase. 8
      4.1.1. Introducción. ¿Qué es y para qué sirve? 8
      4.1.2. Productos usados. 9
         4.1.2.1. Cloud Firestore. 9
            4.1.2.1.1. Colecciones. 9
            4.1.2.1.2. Documentos. 10
            4.1.2.1.3. Datos. 10
         4.1.2.2. Authentication. 11
         4.1.2.3. Hosting. 11
            4.1.2.3.1. ¿Por qué hemos escogido Firebase Hosting? 12
         4.1.2.4. Analytics. 13
   4.2. Conectividad con la aplicación. 14
   4.3. Google API's 15
      4.3.1. Geolocation API 15
      4.3.2. Maps JavaScript API 15
5. ¿Qué es capaz de hacer WIMB? ¿Cómo lo hace? 16
   5.1. Conexión con Firebase, permisos de ubicación y mapa inicial. 16
   5.2. Apertura de la web e inicio de sesión. 17
      5.2.1. Apertura de la web e inicio de sesión. Código. 18
   5.3. Crear o unirse a una sala. 19
      5.3.1. Crear o unirse a una sala. Código. 20
   5.4. Sala privada. 22
      5.4.1. Sala privada. Código. 23
   5.5. Buscar. 28

5.5.1. Buscar. Código. 29
5.6. Buscar todos. 31
5.6.1. Buscar todos. Código. 32
5.7. Área de Administradores. 33
5.7.1. Área de administradores. Código. 35
6. Dificultades. 40
6.1. Android Studio y el compilador Gradle 40
6.2. Apache Cordova 40
6.3. Manejo de variables entre funciones: 40
7. Bibliografía. 41

1. Introducción y objetivos

1.1. ¿Qué es la geolocalización y para qué sirve?

La geolocalización nace a raíz de la necesidad de obtener la ubicación real de un objeto, un radar, un teléfono móvil o un ordenador conectados a Internet.

Este término está directamente ligado al uso de sistemas de posicionamiento, que no siempre está orientado a localizar o determinar la ubicación de un dispositivo, sino que también es usado a la hora de determinar una posición concreta, como podría ser una calle, y no solo por un conjunto de coordenadas geográficas.

Este proceso es posible gracias al uso de sistemas de información geográfica compuestos por un conjunto de hardware, software y datos geográficos, diseñados para capturar, almacenar, manipular y analizar dicha información.

Los datos de geolocalización se pueden obtener de diversas maneras:

- Mediante la navegación web a través de direccionamiento IP. Con esta práctica es como **WIMB** obtiene la latitud y la longitud de los dispositivos para, más tarde, poder tratar dichos datos.
- Teléfonos móviles.
- Dispositivos GPS (Sistema de Posicionamiento Global).
- Identificación de radiofrecuencias.
- Transacciones con tarjeta de crédito o débito.

En muchos de los casos, los dispositivos, sistemas o páginas de geolocalización nos ayudan a sentirnos más seguros y nos facilitan la posibilidad de saber dónde estamos, cual es la mejor ruta posible o a que distancia se encuentra el punto A del punto B con exactitud. Las empresas son capaces de realizar estadísticas basadas en la frecuencia con la que los usuarios frecuentan por sus tiendas o negocios, o en qué puntos del mapa hay más o menos densidad de población.

Al igual que utilizar este tipo de tecnologías es una gran ventaja y un gran avance para las empresas, también puede ser una práctica muy peligrosa. Una mala implementación de dichos sistemas o una configuración de seguridad poco eficaz, puede poner en riesgo la privacidad del individuo dejando expuestos aquellos datos que le sitúan en el mapa.

Es muy importante reforzar la seguridad en estos casos y utilizar puertos y protocolos seguros, como podría ser **https** en caso de la obtención y trata de datos mediante navegación web.

Como “Trabajo de Fin de Grado”, nuestra solución frente a las necesidades de los usuarios a la hora de poder utilizar un servicio fiable y útil para geolocalizar a sus amigos o familiares a tiempo real y con total exactitud recibe el nombre de **WIMB**.

1.2. Objetivos.

El objetivo de nuestro servicio es poder proporcionar a los usuarios una web, orientada a dispositivos móviles, capad de crear lo que llamamos “salas” donde todo aquel que se encuentre en la misma comparta su ubicación a tiempo real con el resto de los usuarios, siempre y cuando haya permitido previamente la sustracción de dicha información.

A su vez, es posible geolocalizar a todos los usuarios que se encuentren en la misma sala mediante marcadores situados en un mapa al cual todos tienen acceso.

Aquel usuario que ha creado dicha sala, automáticamente cumplirá el rol de administrador y será capaz de acceder al “Área de administradores”, donde puede crear nuevos administradores, expulsar usuarios, cerrar e inhabilitar la sala o dejarla accesible al público.

Haciendo uso de los servicios proporcionados por “Firebase”, podemos garantizar un servicio duradero, eficaz y seguro.

Gracias a la herramienta de “Firebase Authentication”, la cual utilizamos para tener un sistema de autenticación completo mediante correo electrónico y verificación en dos pasos, podemos asegurar que las personas que usan nuestra web son quien dicen ser o por lo menos tienen limitaciones a la hora de registrarse y utilizar nuestros servicios.

Otra de las herramientas proporcionadas por “Firebase” y que ha jugado un papel importante a la hora de crear **WIMB**, recibe el nombre de “Firestore Database”.

“Firestore Database” es la solución que propone “Firebase” para el uso de una base de datos no relacional (NoSQL). Gracias a dicha herramienta, hemos sido capaces de gestionar todo tipo de información y clasificarla dependiendo de las necesidades.

Por último, pero no menos importante, dentro de dichas herramientas encontramos el “Hosting” que proporciona “Firebase”. Una solución fácil y eficaz para poder subir tu web a producción y así ser accesible para el resto de usuarios.

Proporcionando certificados de seguridad SSL y HTTP2 de forma automática y gratuita para los dominios, haciendo la navegación más segura.

Por otra parte, también han sido contratados servicios de Google con el objetivo de implementar APIs como son “Geolocation API” y “Maps JavaScript API” para el uso del mapa, la obtención de los datos de geolocalización (latitud y longitud) y la posibilidad de gestionarlo con lenguaje JavaScript.

2. ¿A quién va dirigido? ¿Por qué es necesario su uso?

La aplicación está enfocada a cualquier tipo de usuario con cualquier tipo de necesidad.

Poniendo ejemplos prácticos, podríamos hablar de la importancia que tendría su uso a la hora de tratar con padres que quieren saber si sus hijos están en el colegio o simplemente poder controlar el camino hasta el mismo. También podría estar enfocado para gente con cierta edad o que tiene algún tipo de enfermedad/discapacidad que la incapacita a la hora de hacer su día a día y, de esta forma, poder saber dónde está en caso de que se pierda o se desvíe de su rumbo.

Por otra parte, también podría estar enfocada para grupos que están de excursión, en algún campamento o en una yincana. Estando todos en la misma sala, el cuidador podría saber fácilmente dónde está cada uno de los integrantes si alguien se pierde.

Otro ejemplo que podemos destacar y que realmente tiene más relevancia de la que parece, sería enfocar nuestra web sobre grupos de adolescentes a la hora de salir a locales nocturnos o viajar por el mundo. Dado que existen ciertos riesgos tanto en la vida nocturna como el explorar nuevos países, WIMB ayuda a tener a un grupo de personas en constante vigilancia unos con otros. De esta manera, sería mucho más fácil actuar con rapidez ante una situación de riesgo.

Da igual donde te encuentres, la situación en la que estés o quién seas, con WIMB puedes localizar a tu grupo de familiares, amigos o alumnos en cualquier momento y con una exactitud casi perfecta.

¡WIMB actualiza de forma automática tu posición cada 5 segundos!

De esta manera, en caso de que a cualquier persona que esté utilizando nuestra página le ocurriese un accidente, quedaría registrado hasta el último segundo en el que se compartió su localización a toda la sala y, como es evidente, las coordenadas precisas.

¿No te gusta que una web obtenga tu localización? Lo entendemos, por eso WIMB se asegura de que nadie más que aquellas personas que se encuentren en tu misma sala (entendemos que darás acceso a gente de confianza) pueda acceder a tus datos. Directamente nadie sabrá ni que estas utilizando nuestros servicios.

¿Aún te preocupa que WIMB almacene tu última ubicación? No te preocupes, para ello existe una opción, una vez te encuentres en la sala, llamada “Abandonar sala”. Te estarás preguntando, ¿y qué tiene de importante? Bien, esta opción borra todo tipo de registro o rastro que haya podido dejar tu usuario, es decir, nombre de usuario, nombre de la sala, última actualización, latitud, longitud… ¡Desapareces de la base de datos!

De esta forma, ni siquiera las personas que se encontraban en tu misma sala pueden tener acceso a aquellos datos que hace un instante sí tenían.

3. Lenguajes utilizados.

3.1. HTML

Es un lenguaje formado por etiquetas que se utiliza para describir la estructura y el contenido de una página web, dichas etiquetas pueden definir los textos, las imágenes que conforman la web, las listas o videos entre otros elementos. El HTML que hemos usado, el dinámico, engloba un conjunto de técnicas con dos objetivos principales: ofrecer un control íntegro al diseñador de páginas web y, por otro lado, acabar con el carácter monótono de estos documentos, es decir, nos permite implementar el esqueleto de nuestra página, ajustándose a nuestras preferencias.

Al principio, este lenguaje no estaba pensado para ser utilizado en webs con carácter multimedia, por eso se fue actualizando poco a poco. Aun así, no termina de satisfacer todas las necesidades para el desarrollo total de una página web, por ejemplo, en el apartado del diseño, es por ello que se han creado otros lenguajes para complementar aquellas partes que HTML no puede abarcar.

3.2. JavaScript

JavaScript (JS) es un lenguaje de programación que fue desarrollado a partir del lenguaje Java, cuya principal diferencia es que JavaScript solo funciona dentro de una página HTML. JS presenta una característica especial: sus programas, llamados scripts, se encuentran en las páginas HTML y se ejecutan en el navegador. Estos scripts consisten en unas funciones que son llamadas desde el propio HTML cuando algún evento sucede. Así, creamos una página web dinámica que incorpora efectos como: texto que aparece y desaparece, animaciones, acciones que se activan al pulsar botones o ventanas con mensajes de aviso al usuario.

3.3. CSS

CSS se trata de un lenguaje más completo que HTML en el apartado del diseño de la aplicación, pues permite realizar acciones de cambio: del tipo de letra, de los colores o añadir márgenes y editar el estilo de los botones o enlaces. Para que el CSS funcione hay que agregar clases o identificadores en el HTML y en el fichero CSS poner el nombre de la clase o incluir la etiqueta. También ayuda mucho a la hora de proporcionar estilo rápidamente a los elementos que se repiten con frecuencia en el propio HTML, ya que, al incorporar una misma clase a los elementos, la hoja de estilos afectaría a todos los elementos a la par, en lugar de tener que ir implementando todos los elementos uno a uno.

3.4. Bootstrap

Es un framework que combina CSS y JavaScript que sirve para estilizar las páginas web HTML, permite mucho más que cambiar el color de botones o enlaces, esta herramienta permite dar interactividad en la página, ofrece una serie de componentes que facilitan la comunicación con el usuario, como barras de progreso, controles de páginas, menús de navegación entre otras muchas más opciones. Para asignar una característica a un elemento, simplemente tenemos que informar la clase correspondiente en la propiedad "class" del elemento que desea estilizar.

También permite hacer la página ajustable, permite poner diferentes tipos de alertas, con colores específicos, según la situación, también se pueden poner carruseles que es un recurso muy utilizado, que consiste en una presentación de imágenes, se pueden incluir efectos especiales para la transición de imágenes y controles de visualización, como por ejemplo poner indicaciones de "siguiente" y "anterior" o barras de navegación, que permite construcción de un sistema de navegación para desplazarse entre las distintas páginas de la aplicación, dichas barras es posible configurarlas para que se muestren en la posición deseada, crear menús que se puedan extender o contraer, también permite definir cómo mostrar los enlaces del menú, para facilitar la implementación de la navegación de la aplicación.

4. Aprendizaje y herramientas o servicios utilizados.

4.1. Firebase.

4.1.1. Introducción. ¿Qué es y para qué sirve?

Firebase de Google es una plataforma en la nube para el desarrollo de aplicaciones web y móvil. Está disponible para distintas plataformas (iOS, Android y web), haciendo más fácil su desarrollo.

Firebase dispone de varias herramientas que se gestionan desde una misma plataforma, estas se pueden dividir en: desarrollo, crecimiento, monetización y análisis.

Algunas de estas herramientas en el apartado del desarrollo son:

-Cloud Firestore: Trata de una base de datos en tiempo real alojada en la nube, almacena los datos en JSON y es no SQL.

-Autenticación de usuarios: Como en toda aplicación o sitio web se necesita de una identificación de los usuarios, para ello Firebase ofrece un sistema de autenticación que permite el registro mediante email y contraseña o el acceso utilizando las cuentas de otras plataformas como Twitter, Google o Facebook.

-Hosting: Otra herramienta es un servidor para alojar las aplicaciones de manera rápida y sencilla, este hosting proporciona certificados de seguridad SSL y HTTP2.

También ofrece herramientas para testear las aplicaciones, hacer configuraciones remotamente, un seguimiento de los errores para ayudar a solucionarlos y por último ofrece un almacenamiento en la nube para guardar los ficheros de las aplicaciones.

Por el lado del crecimiento ofrece un servicio de notificaciones para informar a los usuarios de eventos, que puede ser un mensaje o una información importante.

También dispone de un posicionamiento en el buscador de Google y de esta forma da a conocer el proyecto.

Por último, en el apartado del crecimiento ofrece la posibilidad de que los usuarios animen a sus contactos a utilizar la app o compartir contenido a través de e-mails o por SMS. Y Adwords te ofrece la posibilidad de realizar campañas de publicidad para dar a conocer la aplicación.

4.1.2. Productos usados.

Para la realización de nuestra web, hemos utilizado algunas de las herramientas nombradas anteriormente como "**Cloud Firestore**". La base de datos que utilizamos para almacenar los datos de las salas, sus usuarios, quién es el administrador, sus coordenadas, etc.

Otra herramienta que hemos usado es la de "**Authentication**", que nos sirve para registrar quién entra, exigiendo un acceso mediante credenciales y, evitar así, que entre cualquier persona.

También, hemos utilizado el servicio de "**Hosting**". Permite que se pueda acceder a la aplicación desde cualquier sitio ya que está alojada en la red y da seguridad a la aplicación.

Por último, hemos usado la herramienta de "**Analytics**" que permite que veamos ciertas estadísticas sobre el funcionamiento de la aplicación y nos pueda ayudar a mejorarla de cara al futuro.

4.1.2.1. Cloud Firestore.

Cloud Firestore es una base de datos NoSQL (no relacional) alojada en la nube a la que pueden acceder tus apps para iOS, Android y Web directamente desde los SDK nativos. Cloud Firestore también está disponible en los SDK nativos de muchos otros lenguajes (Node.js, Java, Python, Unity, C++ y Go, además de las API de REST y RPC).

A partir del modelo de datos NoSQL de Cloud Firestore, almacenan los datos en documentos que contienen campos que se asignan a valores. Estos documentos se almacenan en colecciones, que son contenedores para tus documentos y que puedes usar para organizar tus datos y compilar consultas.

Los documentos admiten varios tipos de datos diferentes, desde strings y números simples, hasta objetos anidados complejos. También puedes crear subcolecciones dentro de documentos y crear estructuras de datos jerárquicas que se ajustan a escala a medida que tu base de datos crece.

4.1.2.1.1. Colecciones.

Son el contenedor de los documentos almacenados en la base de datos, dentro de las colecciones solo pueden contener documentos y no puede contener campos con valores de manera directa, ni tampoco puede contener otras colecciones.

Al no hacer uso de esquemas, se denota cierta libertad sobre los campos que son introducidos en cada documento y los tipos de datos que se almacenan en esos campos.

Los documentos que se encuentran dentro de una colección pueden tener campos distintos o almacenar diferentes tipos de datos en sus campos, pero se recomienda usar los mismos campos y tipos de datos en los documentos de una colección. De esta manera, se facilita la consulta de los datos.

4.1.2.1.2. Documentos.

Las unidades de almacenamiento en Firestore son los documentos, estos tienen unos campos con valores que se le asignan y se identifican con un nombre. También existen los objetos complejos anidados que en un documento se llaman mapas, estos mapas se crean cuando estructuras un dato en varios apartados. Por ejemplo, si el dato “Nombre completo” lo divides en “Nombre”, “Primer apellido” y “Segundo apellido”. Todos estos documentos se pueden tratar como registros JSON puesto que son muy similares.

4.1.2.1.3. Datos.

Los datos son cada uno de los campos que tiene un documento, donde se guarda la información, por ejemplo, en el documento de un usuario sus datos pueden ser su nombre, apellido, edad o cualquier dato que sobre una misma persona.

Estos datos pueden agregarse como cadenas de texto, números, booleanos, fechas y horas, puntos geográficos, valores nulos o rutas a directorios entre otros.

4.1.2.2. Authentication.

La mayoría de las apps necesitan identificar a los usuarios. Conocer la identidad de un usuario permite que una app guarde sus datos en la nube de forma segura y proporcione la misma experiencia personalizada en todos los dispositivos del usuario.

Firebase Authentication proporciona servicios de backend, SDK (Software Development Kit) fáciles de usar y bibliotecas de Identificador de Usuario, ya elaboradas para la autenticación de los usuarios en la aplicación. Admitirá la autenticación mediante contraseñas, números de teléfono, proveedores de identidad federada populares, como Google, Facebook, Twitter, y muchos otros.

Firebase Authentication se integra estrechamente con otros servicios de Firebase y aprovecha estándares de la industria como OAuth 2.0 y OpenID Connect, por lo que se puede integrar con facilidad en tu backend personalizado.

4.1.2.3. Hosting.

Es un servicio que permite hacer un hosting seguro y rápido para las aplicaciones web, proporciona certificados de seguridad SSL y HTTP2 de forma automática y gratuita para los dominios y haciendo la navegación en nuestra aplicación más segura. El contenido de las aplicaciones se publica rápidamente sin importar la ubicación del usuario, ya que se almacenan en caché en SSD ubicados en servidores perimetrales de una CDN (Content Delivery Network) distribuidos por todo el mundo y todo el contenido se entrega segura desde el servidor más cercano, para subir las carpetas de la app a la web se hace con un solo comando desde un directorio local.

Otra de sus funciones es poder revisar el historial de implementaciones y revertir a una versión anterior desde la consola de Firebase.

Permite configurar el comportamiento del Hosting especificando qué archivos del directorio del proyecto local se van a implementar en Firebase Hosting, o poner una página personalizada para el error 404/ Not Found entre otras configuraciones.

Firebase ofrece varias opciones de dominios y subdominios, por defecto cada proyecto tiene subdominios gratuitos en los dominios “web.app” y “firebaseapp.com”. Estos dos ofrecen la misma configuración y contenido implementado.

Para implementarlo solo hay que instalar Firebase CLI. Esto te permite crear un nuevo proyecto de Hosting, ejecutar un servidor de desarrollo local y también implementar contenido.

Después hay que configurar un directorio de proyecto, para esto hay que agregar los elementos estáticos al directorio de un proyecto local y luego ejecuta “firebase init”, con el objetivo de conectar el directorio a un proyecto de Firebase.

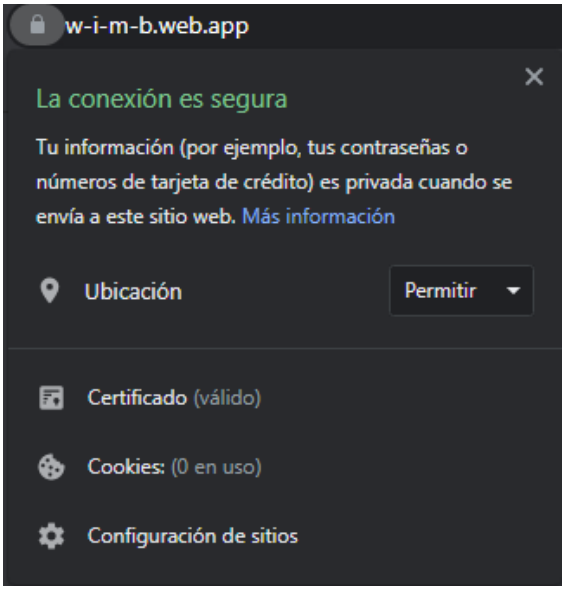
Más tarde, para poder visualizar, probar y compartir los cambios antes de publicarlos, ejecutar “firebase emulator:start ” para emular hosting y los recursos de tu proyecto de backend en una URL alojada localmente, luego para ver y compartir los cambios en una URL de vista previa temporal hay que ejecutar “firebase hosting:channel:deploy” con el fin de crear e implementar en un canal de vista previa.

Tras realizar estos pasos, para poder implementar el sitio una vez que la aplicación está terminada, ejecutar “firebase deploy”. Esto sube la versión más reciente al servidor. Si en algún momento se desea deshacer la implementación se puede hacer con un clic en la Firebase console.

Por último, se puede vincular el sitio a una aplicación web de Firebase, y con esto se puede implementar Google Analytics para recopilar datos y comportamiento de tu app y Firebase Performance Monitoring para obtener estadísticas sobre las características de su rendimiento, Pero este último paso es opcional.

4.1.2.3.1. ¿Por qué hemos escogido Firebase Hosting?

La razón por la que hemos elegido utilizar el producto “Hosting” de “Firebase” es porque siempre entrega el contenido a partir de una conexión segura mediante SSL, sin necesidad de configuración, admitiendo todo tipo de contenido, desde los archivos CSS y HTML hasta las API o microservidores.



Se puede subir el contenido rápidamente, ya que todo lo que se sube se almacena en caché en discos SSD ubicados en el perímetro de la CDN distribuidos por todo el mundo y así se puede entregar el contenido de manera automática y precisa a cualquier usuario independientemente de donde se encuentre.

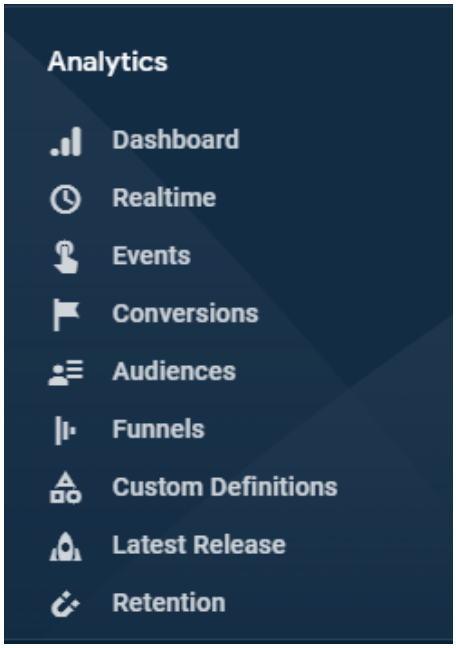
Además, nos permite observar y probar los cambios realizados en la aplicación en una URL alojada localmente y poder interactuar con un backend emulado a la par de compartir los cambios con los compañeros del equipo mediante URL de vista previa. Y, por último, admite ponerse en funcionamiento en cuestión de segundos gracias a Firebase CLI.

Y sobre todo porque sin ningún coste ofrece una mayor velocidad, gracias al protocolo HTTP2, y una importante seguridad en la aplicación, con el protocolo HTTPS, nos permite personalizar el dominio y proporciona un certificado SSL a cada uno de los dominios que se poseen y resulta sencillo de implementar y gestionar ya que pertenecer a Firebase.

4.1.2.4. Analytics.

La principal función de esta herramienta es medir el impacto de nuestra web y ver el comportamiento de los usuarios, con la finalidad de mejorar el rendimiento de la aplicación y poder, en caso de ser necesario, optimizarlo. También puede realizar un seguimiento de prácticamente todas las interacciones realizadas en las páginas con bastante detalle. Por eso, lo ideal es planificar una estrategia de seguimiento. Las funciones de esta herramienta son:

-Conocer a tus usuarios, sabiendo desde que país acceden, con que dispositivo se conectan o el idioma que utilizan, su género o edad entre otros datos. También puedes ver sus afinidades e intereses para crear campañas personalizadas, a través de cookies.



-Explorar el comportamiento, esto permite estudiar cómo y cuándo los usuarios utilizan la aplicación o que contenido consumen, el tiempo de utilización de la misma, que páginas visitan y en dónde abandonan la página web, o cuántas personas están navegando en la web.

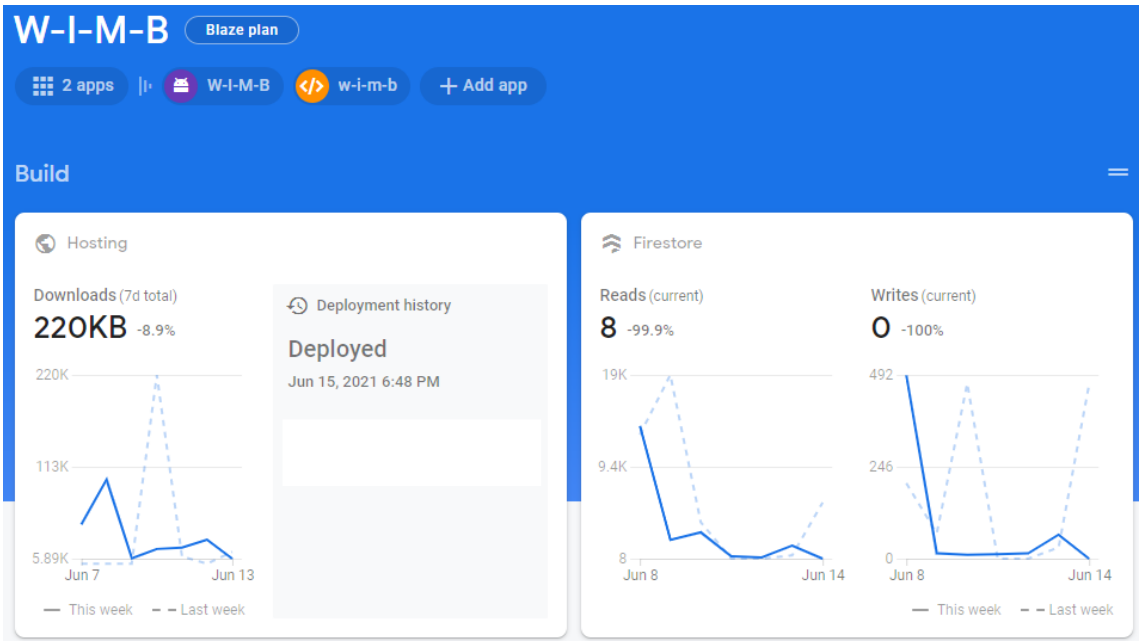
-Medir las interacciones, con una buena configuración se puede ver en qué botones hacen click los usuarios ya sea para ver vídeos, solicitar información, descargar información. La medición de las interacciones también puede avisar de problemas técnicos o de diseño en la aplicación.

-Analizar el rendimiento de canales, esta herramienta permite analizar a las visitas de un canal específico. Los canales son las redes sociales, desde un email, el buscador, anuncios u otros sitios y de esta forma puede ayudar a mejorar las campañas.

4.2. Conectividad con la aplicación.

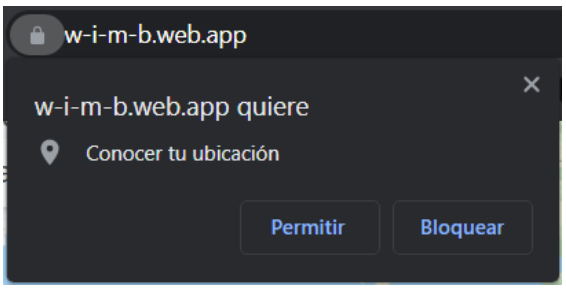
Tras crear un proyecto con firebase, puedes asociar tu aplicación o web a dicho proyecto gracias, en nuestro caso, a la implementación de una serie de configuraciones en el código que hace posible la conectividad y la comunicación entre ambos lados.

Gracias a esto, Firebase puede proporcionarnos multiples servicios y productos para desarrollar nuestra web, aparte de estadisticas y recomendaciones de seguridad.



4.3. Google API's

4.3.1. Geolocation API



Geolocation API es una API compatible con Google Maps API con la habilidad de permitir que el navegador encuentre la ubicación de manera aproximada con un margen de error de unos 5/10 metros, para hacerlo posible utiliza lo explicado a continuación. Primero de todo debe existir la dirección IP de los usuarios. Esto indica la región y comunidad desde la que se conecta un usuario a través de Internet con su servicio.

Todo ello funciona gracias a que cada país, comunidad y ciudad tiene asignado un rango de IP concreto. Siempre que el usuario esté conectado a Internet, estará ofreciendo su dirección IP pública que corresponde con su geolocalización. Algunos dispositivos se refrescan de manera regular, informando con frecuencia de su dirección IP, mientras que otros deben permitirlo de forma manual.

Para ser más precisos, el navegador recolecta la información enviada desde las señales Wifi que encuentra a su alrededor, las cuales incluyen tanto las redes públicas como las privadas. Estas señales quedan registradas y geo-referenciadas por compañías que ofrecen estos servicios, por ejemplo, las compañías telefónicas.

4.3.2. Maps JavaScript API

La función de esta API es la de mostrar de manera gráfica la reproducción de los lugares con mapas. Así, a la hora de geolocalizar a un usuario servirá para representar al mismo en un punto del mapa y poder identificar calles o barrios, siendo más sencillo llegar hasta la dirección IP.

Aunque se trata de una herramienta que no es gratuita, cuenta con más de 25.000 mapas gratis al alcance de todos, el mayor número de mapas conocidos sin necesidad de pago adicional, y con ello es la API más eficaz a la hora de ser implementado en una aplicación.

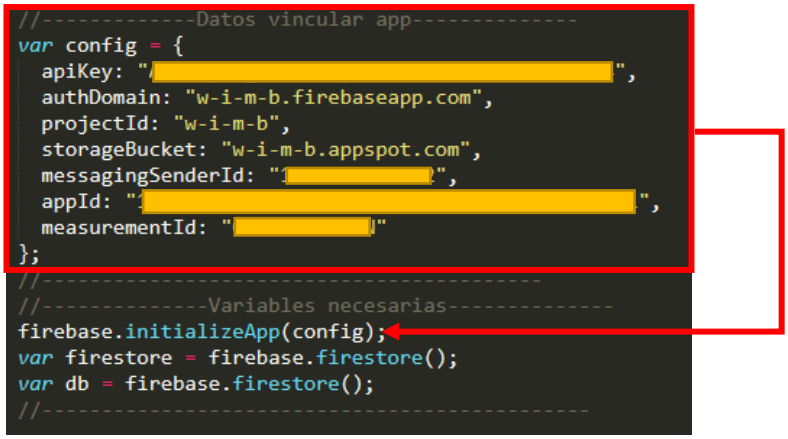
Esta herramienta no requiere de información del usuario, ya que se cargará el mapa independientemente de si el usuario cede su localización actual o no, pero será imprescindible aceptar compartir la geolocalización para poder ubicar al usuario en dicho mapa.

5. ¿Qué es capaz de hacer WIMB? ¿Cómo lo hace?

5.1. Conexión con Firebase, permisos de ubicación y mapa inicial.

En este apartado se va a hablar de las configuraciones mínimas con las que WIMB se conecta con los servicios de Firebase.

Tras crear un proyecto es esta plataforma, se facilitan usa serie de datos que son necesarios para realizar una conexión con nuestra web.



Dichas líneas de código se declaran como variables para posteriormente hacer uso de ellas, como por ejemplo es el caso de la variable "**config**". Tras ser declarada, es llamada en una función que se muestra posteriormente para que se ejecute de forma automática la conexión.

```
/*--------------------Permitir compartir ubicación---------------------*/
function ubicacion(){
    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(initMap);
        navigator.geolocation.getCurrentPosition(actualizar);
        console.log("permitirUbicacion");
    }
}
/*--------------------Permitir compartir ubicación---------------------*/
```

La función "**ubicacion()**" es llamada una vez carga la página principal y se encarga de solicitar al usuario permisos de acceso a la localización. Una vez el usuario acepta compartir su ubicación, los datos de latitud y longitud son constantemente enviados a las funciones "**initMap**" y "**actualizar**".

```
let map;
/*----------------------------FUNCION MAPA 1 y 2-----------------------------*/
function initMap1() {
/*--------------------------------MAPA-----------------------------------*/
    const mapOptions = {
        zoom: 3,
        center: { lat: 3, lng: 4 },
    };
    map = new google.maps.Map(document.getElementById("map"), mapOptions);
}
```

Tras ser declarada la variable "**map**", se crea la función "**initMap1()**" la cual es llamada al cargar la página principal y es la encargada de mostrar el mapa a la izquierda.

Esta función no es capaz de situar al usuario en el mapa dado que aun no se hace uso de los datos de ubicación. Únicamente se muestra una visión general del mapa con un zoom determinado.

5.2. Apertura de la web e inicio de sesión.

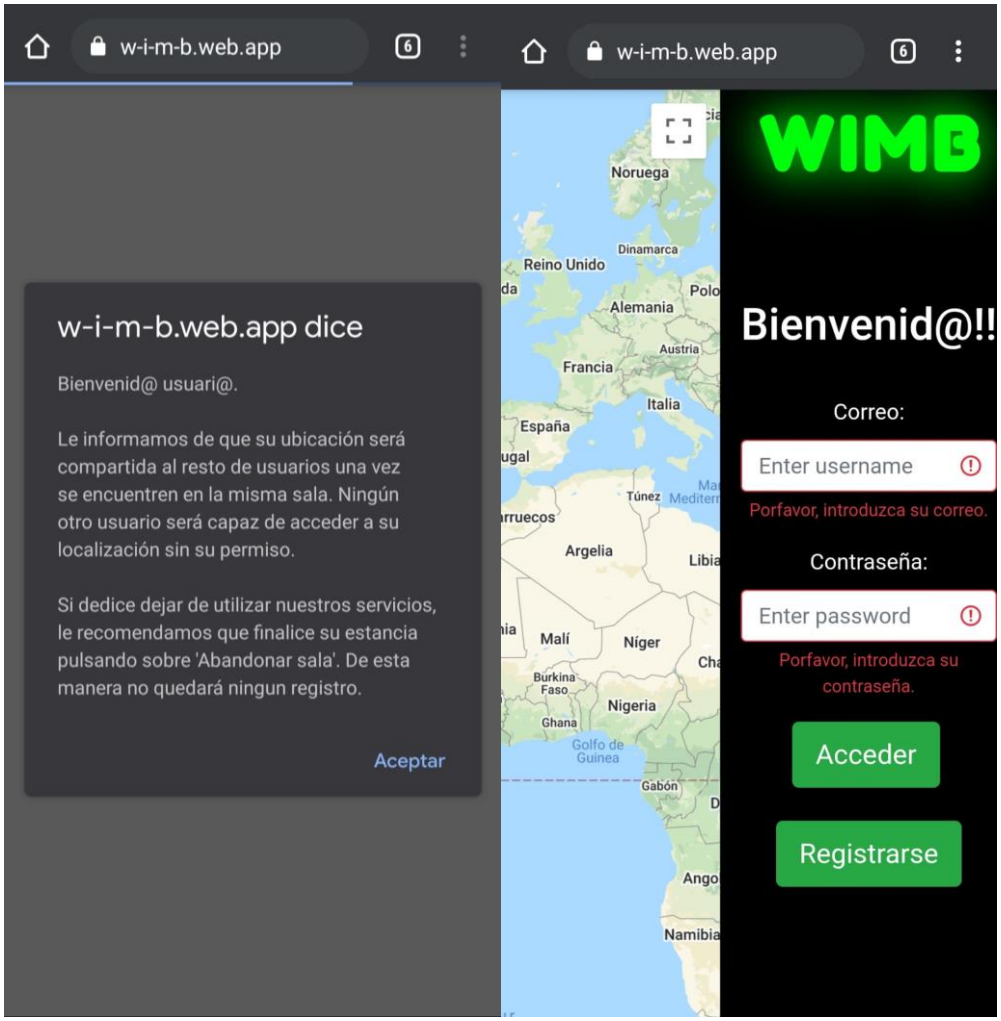
Tras acceder a la url que inicia la aplicación web, aparece un mensaje previo a su apertura que informa al usuario de que su estancia en la web es y será totalmente segura.

Una vez se pulse ‘Aceptar’, se da por hecho que el usuario está de acuerdo y se procede a mostrar el inicio de sesión.

Dicho inicio de sesión conecta directamente con el sistema de autenticación configurado en la nube, gracias a los servicios proporcionados por el producto ‘Authentication’ de ‘Firebase’.

Completar ambos campos, correo y contraseña, sirven tanto para acceder en caso de que el usuario ya esté registrado, como para registrar un usuario en sí.

Al cabo de unos segundos, la propia página web pregunta al usuario si permite compartir su ubicación. En esta pantalla no se trata con esos datos, simplemente se solicitan los permisos al principio para no tener inconvenientes a la hora de hacer un uso real de la web.



5.2.1. Apertura de la web e inicio de sesión. Código.

La función "**registrarse()**" está asocia al botón "**Registrarse**". Declara las variables "**correo**" y "**password**" cuyo valor es el introducido por el usuario en los diferentes campos. Posteriormente, se utilizan dichas variables para hacer uso de la función propia de creacion de usuario de firebase.

Dado que previamente ha sido habilitada la creación de usuarios mediante correo electrónico y contraseña en la plataforma, podemos hacer uso de la función "**firebase.auth().createUserWithEmailAndPassword()**".

```
70  //--------------------Funciona Crear Cuenta--------------------
71  function registrarse() {
72
73      var correo=document.getElementById('correo').value;
74      var password=document.getElementById('password').value;
75
76    firebase.auth().createUserWithEmailAndPassword(correo, password)
77      .then((userCredential) => {
78        var user = userCredential.user;
79        console.log("Saved");
80        window.location.href='www/menu2.html';
81      })
82      .catch((error) => {
83        var errorCode = error.code;
84        var errorMessage = error.message;
85        window.alert("Ese correo ya existe");
86      });
87  }
88  //-------------------------------------------------------------
```

A diferencia de la función anterior, esta está asociada al acceso directo, es decir, el usuario ha de haberse creado con anterioridad. La declaración de las variables es la misma y su uso similar. La diferencia es que, en este caso, se hace uso de la función "**firebase.auth().signInWithEmailAndPassword()**".

```
89  //--------------------Funciona Inicio de sesion-------------------------
90  function acceder() {
91
92      var correo=document.getElementById('correo').value;
93      var password=document.getElementById('password').value;
94
95      firebase.auth().signInWithEmailAndPassword(correo, password)
96        .then((userCredential) => {
97          var user = userCredential.user;
98          console.log("LogIn")
99           window.location.href='www/menu2.html';
100
101       })
102       .catch((error) => {
103         var errorCode = error.code;
104         var errorMessage = error.message;
105         window.alert("Credenciales erroneas");
106       });
107 }
108
```

5.3. Crear o unirse a una sala.

Una vez iniciada la sesión, la web dirige directamente al usuario a la unión o creación de lo que llamamos “salas”. Dichas salas son únicas y totalmente privadas, ya que no solo necesitas el nombre de la misma, también será imprescindible la contraseña que la corresponde.

Tanto la creación de una nueva sala como la unión a una existente solicitan los mismos campos, aunque se diferencie una acción de la otra gracias a los botones de “Crear Sala” o “Unir Sala”.

Al crear una nueva sala, no solo se crea la sala en sí, sino que a su vez se genera una estancia en la base de datos y es creado un grupo de administradores. Al cual pertenece aquel usuario que haya creado la sala y cuyos campos asociados son el nombre de la misma y la contraseña correspondiente.

Como se puede comprobar, el mapa situado a la izquierda de la pantalla aún no está en funcionamiento. Esto corrobora que aún no son tratados los datos de ubicación.

5.3.1. Crear o unirse a una sala. Código.

La función "**crear()**" está directamente asociada al botón "**Crear Sala**". Se generan las variables "**Usala**", "**Uuser**" y "**pass**" cuyos valores son los introducidos por el usuario en los diferentes campos.

Posteriormente se reemplazan, los valores de las variables, los posibles espacios introducidos por el símbolo "_". De esta forma, unificamos los caracteres y evitamos problemas a la hora de tratar los datos.

Más tarde, las variables "**Usala**" y "**Uuser**" pasan a ser variables de sesión. Gracias a esta forma de tratar las variables, podemos acceder a las mismas desde cualquier parte del código dado que han sido creadas dentro de una función.

Tras haber alterado el tipo de variable, declaramos las constantes "**docRef**" y "**admRef**" encargadas de hacer una conexión directa con la base de datos utilizando el nombre de la sala y el nombre del usuario tanto para crear la sala como para enviar al creador de la misma al grupo "**Administradores**".

Por último, se comprueba que la variable "pass" no se encuentra vacía y así evitar crear una sala sin contraseña. Una vez se confirma que la variable no se encuentra vacía, se procede a enviar al creador de la sala al grupo "Administradores", se crea la sala y se redirige al usuario a la misma.

```
109  //--------------------Función Crear Sala------------------------------
110  function crear(){
111
112      var Usala=document.getElementById('Usala').value;
113      var Uuser=document.getElementById('Uuser').value;
114      var pass=document.getElementById('pass').value;
115
116      Usala = Usala.replace(" ","_");
117      Uuser = Uuser.replace(" ","_");
118
119      sessionStorage.setItem('NombreSala', Usala);
120      sessionStorage.setItem('NombreUsuario', Uuser);
121
122      const docRef = firestore.collection(Usala).doc(Uuser);
123      const admRef = firestore.collection("Administradores").doc(Uuser);
124
125      //COMPROBAR QUE EL CAMPO CONTRASEÑA NO ESTÉ VACIO PARA EVITAR CREAR UNA SALA SIN CONTRASEÑA
126      if (pass == ""){
127          window.alert("Faltan contraseña.");
128      }else{
129          //AÑADE AL CREADOR DE LA SALA AL GRUPO ADMINISTRADORES JUNTO CON
130          //EL NOMBRE DE LA SALA Y SU CORRESPONDIENTE CONTRASEÑA
131          admRef.set({
132              NombreSala: Usala,
133              Password: pass
134          });
135          //AÑADE AL CREADOR DE LA SALA A LA PROPIA SALA Y LO REDIRECCIONA
136          docRef.set({
137              Nombre: Uuser
138          }).then(function() {
139              console.log("Saved");
140              window.location.href='sala.html';
141          }).catch(function() {
142              console.log("Got error.");
143          });
144      }
145  }
146  //*************************************************************************
```

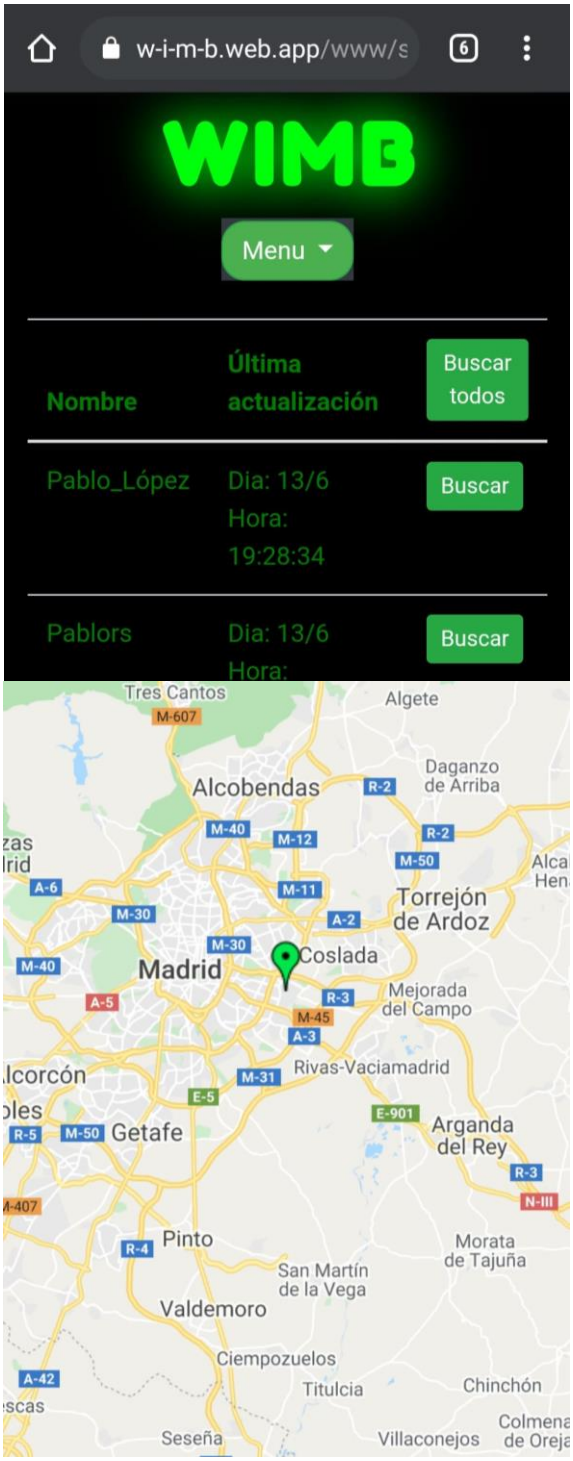
La función “**unirse()**” está ligada al botón “**Unir Sala**” y cuyas variables iniciales, la modificación de las mismas y el acceso a la base de datos son similares a la función mencionada anteriormente “**crear()**”.

Lo que diferencia a esta función de la anterior es que antes de realizar la conexión con la sala, primero se comprueba que, tanto el nombre de la sala como la contraña introducidas, coinciden con los datos asociados al creador de la sala. Dicha información es contrarrestada tras consultarlo con la colección “**Administradores**”, de la cual pertenece el creador de la sala y tiene vinculadas dichas credenciales de acceso.

Una vez se verifican los datos, el usuario es redirigido a la sala correspondiente.

```
147  //--------------------Función Unirse Sala-----------------------------
148  function unirse(){
149      var Usala=document.getElementById('Usala').value;
150      var Uuser=document.getElementById('Uuser').value;
151      var pass=document.getElementById('pass').value;
152
153      Usala = Usala.replace(" ","_");
154      Uuser = Uuser.replace(" ","_");
155
156      sessionStorage.setItem('NombreSala', Usala);
157      sessionStorage.setItem('NombreUsuario', Uuser);
158
159      var docRef = firestore.collection(Usala).doc(Uuser);
160
161      //OBTIENE LOS CAMPOS 'NombreSala' Y 'Password' DEL GRUPO ADMINISTRADORES
162      //Y LOS COMPARA CON LAS CREDENCIALES INTRODUCIDAS POR LOS USUARIOS
163      db.collection('Administradores').get().then((querySnapshot) => {
164          querySnapshot.forEach((doc) => {
165              if (doc.data().NombreSala == Usala && doc.data().Password == pass) {
166                  //TRAS COMPROBAR QUE LOS CAMPOS COINCIDEN,
167                  //AÑADE AL USUARIO A LA SALA Y LE REDIRIGE A LA MISMA
168                  docRef.set({
169                      Nombre: Uuser
170                  }).then(function() {
171                      console.log("Saved");
172                      window.location.href='sala.html';
173                  }).catch(function() {
174                      console.log("Got error.");
175                  });
176              }else{}
177          });
178      });
179
180  }
181  //*************************************************************************
```

5.4. Sala privada.



Tras haber creado la sala o haberse unido a una existente, se accede a la siguiente pantalla. En esta pantalla podemos comprobar la presencia de aquellos usuarios a los que se les han compartido las credenciales y han podido acceder a la sala.

La columna "**Nombre**" nos muestra cómo se identifican los usuarios que han accedido. Mientras que la columna "**Última actualización**" se refresca automáticamente indicando la última vez que se ha compartido la ubicación con el resto de usuarios.

Cada sala se compone de 4 elementos importantes:

Menú: el menú es común para todos los usuarios de la sala, independientemente de si eres administrador o un usuario corriente. Se compone de las opciones:

Abandonar sala. Como su nombre indica, el usuario que acceda a esta opción abandonará la sala borrando cualquier registro que le asocie con su estancia en la misma (ubicación, nombre, etc.) y le devolverá a la página mencionada en el apartado anterior.

Cerrar sesión. Esta opción devuelve al usuario al "login" inicial eliminando cualquier tipo de datos que se le haya asociado.

Administrador. Cierto es que esta alternativa está visible para cualquier usuario, pero no por ello es accesible para todos. Solo el creador de la sala puede acceder al "Área de administradores".

Mapa: un mapa situado en la parte inferior de la página el cual juega un papel importante en nuestra web, puesto que todas las acciones se reflejan en dicho mapa.

Botón de búsqueda individual y **botón de buscar todos**.

5.4.1. Sala privada. Código.

La función "**initMap()**" está asociada al mapa que podemos observar en la parte inferior de la sala. Como ha sido indicado con anterioridad, los datos de ubicación no habían entrado en juego aún dado que solo se mostraba una imagen general del mapa.

Esta función trabaja directamente con dichos datos. Tras solicitar en un principio los permisos de localización, se envía el parámetro "position" a esta función. De esta manera, se pueden tratar los datos de latitud y longitud para situar una marca, en este caso es la marca personal del usuario, en el mapa.

Gracias al parámetro mencionado, podemos recoger los valores de latitud y longitud declarando las variables correspondientes y cuyo valor es "**position.coords.latitude**" y "**position.coords.longitude**" respectivamente.

Se declara la constante "**mapOptions**" para determinar la variación que va a sufrir el mapa y se pasan dichos valores a la variable "**map**" que es aquella la cual altera el estado del contenedor "**map**", que se encuentra en la página y, es aquel que muestra el mapa en sí.

Una vez centrado el mapa, se declara la constante "**marker**", la cual va a situar una marca de color verde, con el objetivo de diferenciar la marca personal a la del resto de usuarios, en el mapa gracias a las variables declaradas anteriormente de "Latitud" y "**Longitud**".

Por último, es personalizada dicha marca creando la constante "**infowindow()**", indicando el nombre del usuario y sus coordenadas. Esta personalización se puede comprobar gracias al evento "**click**" que, tras pulsar sobre la marca, nos indica los datos mencionados.

```
40  function initMap(position) {
41  /*------------------------------MAPA-----------------------------------*/
42
43      var Latitud = position.coords.latitude;
44      var Longitud = position.coords.longitude;
45
46      const mapOptions = {
47          zoom: 10,
48          center: { lat: Latitud, lng: Longitud },
49      };
50      map = new google.maps.Map(document.getElementById("map"), mapOptions);
51   /*---------------------------------------------------------------------*/
52   /*------------------------------MARCA 1--------------------------------*/
53    const marker = new google.maps.Marker({
54
55      position: { lat: Latitud, lng: Longitud },
56      map: map,
57      icon: 'https://maps.google.com/mapfiles/ms/icons/green-dot.png',
58    });
59
60    const infowindow = new google.maps.InfoWindow({
61     content: "<p>Usted se encuentra aquí: " + marker.getPosition() + "</p>",
62    });
63    google.maps.event.addListener(marker, "click", () => {
64      infowindow.open(map, marker);
65    });
66   /*---------------------------------------------------------------------*/
67  }
68  /*---------------------------FUNCION MAPA-----------------------------*/
```

La función “**traer()**”, como muchas otras que se muestran a continuación, va a jugar un papel importante en muchos aspectos, y es por ello que en este apartado solo se va a explicar una parte del código y, más tarde, se desarrollará la importancia que tiene el resto del código.

En un principio vamos a centrarnos en su funcionalidad principal, “actualizar” los datos en la pantalla del usuario. Esta función es llamada a si mísma en un intervalo de unos 5 segundos, gracias a la función que se muestra tras esta imagen llamada “**set(refrescar,5000)**” que a su vez llama a “**traer()**”. `setInterval(refrescar,5000);`

“**Traer()**” obtiene las variables de sesión “**Usala**” y “**Uuser**” para poder interactuar directamente con la base de datos en la que se encuentran todos los usuarios de la sala y así poder acceder a los datos de cada individuo.

Una vez dicha conexión ha tenido lugar, se imprime la información en la pantalla en forma de tabla y se obtiene tanto el nombre del usuario, reconocido por firestore como “**doc.id**”, el histórico de actualización, es decir, la última actualización de ubicación de cada usuario, y a su vez, se generan una serie de botones individuales de “**Buscar**”.

Dichos botones estan asociados a una función que veremos más adelante para mostrar, de forma individual, la ubicación de un miembro de la sala en concreto.

Para conseguir diferenciar los botones, estos reciben un identificador numérico único (el incremento de “**i**”) a medida que avanza el bucle que los imprime. No solo es necesario ese identificador, cada botón obtiene el valor del nombre de usuario (**doc.id**) que se imprime en ese momento.De esta forma, podemos diferenciar los botones y, a su vez, trabajar directamente con el nombre de usuario que pretendemos que asocie.

Por último, cada botón es impreso con un evento que lo interconecta con la función encargada de mostrar al usuario que se pretende buscar en el mapa y, a su vez, a dicha función se le pasa el parámetro “**i**”, el cual coincide con el identificador de cada botón (“**buscar(i)**”).

```
//------------------------Función traer datos----------------------------
//ACCEDE E IMPRIME TODOS LOS USUARIOS Y SUS CORRESPONDIENTES HISTORICOS CONSTANTEMENTE AL NO SER QUE LA CONDICIÓN
// DE LA FUNCIÓN refrescar() SE HAYA CUMPLIDO
function traer(){

    Usala=sessionStorage.getItem('NombreSala');
    Uuser=sessionStorage.getItem('NombreUsuario');

    var i=0;
    let usuarios=[];

    contenido.innerHTML = "";
    db.collection(Usala).get().then((querySnapshot) => {
        querySnapshot.forEach((doc) => {
            document.getElementById("contenido").innerHTML += ("<tr><td>"+doc.id+"</td><td>"
                +doc.data().Actualizado+"</td><td><button onclick='buscar("+i+")' id="+i+
                " value="+doc.id+" class='btn btn-success btn-sm'>Buscar</button></td></tr>");
            i = i+1;
            let newLength = usuarios.push(doc.id);
            //console.log(usuarios);
        });
    //BUSQUEDA DEL NOMBRE DEL USUARIO DENTRO DEL ARRAY GENERADO AL LISTAR LOS USUARIOS DE LA SALA
    const found = usuarios.find(element => element == Uuser);

    //CONDICION: SI EL USUARIO NO SE ENCUENTRA Y POR LO TANTO NO ESTÁ EN LA CONSTANTE FOUND, EXPULSA AL MISMO
    if(found == undefined){
        sessionStorage.clear();
        window.location.href='menu2.html';
    }

    });
}
//*********************************************************************
```

En esta imagen podemos ver dos funciones muy importantes en este código ya que, sin su existencia, nuestra página no seria capaz de cambiar dependiendo del estado en el que se encuentre.

```
/*----------------Actualizar posición automáticamente--------------------*/
setInterval(actualizar,15000);
/*----------------Actualizar posición automáticamente--------------------*/

/*----------------Refrescar página automáticamente--------------------*/
setInterval(refrescar,5000);
```

"**setInterval(actualizar,15000)**" es una función que llama a la función "**actualizar()**" cada 15 segundos. Gracias a la función "actualizar()" es posible recoger los datos de ubicación y el momento exacto en el que se envían para más tarde ser tratados.

```
/*----------------Actualizar posición automáticamente--------------------*/
setInterval(actualizar,15000);
/*----------------Actualizar posición automáticamente--------------------*/
```

De esta manera, cada 15 segundos se llama a la función "**actualizar()**", la cual obtiene el parámetro "**position**". Es capaz de obtener el instante en el que es llamada (día, mes, hora, minuto y segundo) y la ubicación actual para, posteriormente, conectarse con la base de datos y asociar dicha información al usuario.

Al cumplirse este proceso sobre todo aquel que se encuentre en la sala, cada 15 segundos son actualizados todos los datos de cada miembro en la base de datos. Dado que la base de datos y nuestra web estan en constante comunicación, solo hace falta hacer uso de la función "**traer()**" para que se actualice la información en la pantalla de cada usuario. Como se ha comprobado anteriormente, "**traer()**" es llamada cada 5 segundos.

```
/*----------------------Actualizar mi posición--------------------------*/
function actualizar(position){
    console.log("actualizar");

    Usala=sessionStorage.getItem('NombreSala');
    Uuser=sessionStorage.getItem('NombreUsuario');

    var d = new Date();

    var date =('Dia: ' +d.getDate()+'/');
    var month = (d.getMonth()+1 +' Hora: ');
    var hour = (d.getHours()+':');
    var minute = (d.getMinutes()+':');
    var second = d.getSeconds();

    var res = date.concat(month,hour,minute,second);

    navigator.geolocation.getCurrentPosition(act);


    var docRef = firestore.collection(Usala).doc(Uuser);

    function act(position){
        docRef.set({
            Nombre: Uuser,
            Latitud: position.coords.latitude,
            Longitud: position.coords.longitude,
            Actualizado: res
        });
    }
}
```

A continuación, se puede observar a la función "**refrescar()**". Recordamos que esta función es llamada cada 5 segundos de forma automática.

```
/*----------------Refrescar página automáticamente--------------------*/
setInterval(refrescar,5000);
```

Nos encontramos en la misma situación que anteriormente. El objetivo principal de esta función se basa en llamar a la función "**traer()**" constantemente.

El código escrito en medio se mencionará más tarde a la hora de hablar del "**Área de administradores**".

```
//FUNCIÓN LLAMADA A SI MISMA CADA 5 SEGUNDOS. COMPRUEBA LA EXISTENCIA DE UN DOCUMENTO 'OK',
// EN CASO DE SER ASI, BORRA AL USUARIO DE LA SALA
// Y LA CONDICIÓN DE LA FUNCIÓN traer() PASA A LA ACCIÓN.
//EN CASO DE NO CUMPLIRSE LA CONDICIÓN, PASA DIRECTAMENTE A EJECUTAR LA FUNCIÓN traer() SIN BORRAR DATOS.

function refrescar(){
console.log("Refrescar");
    Usala=sessionStorage.getItem('NombreSala');
    Uuser=sessionStorage.getItem('NombreUsuario');

    db.collection(Usala).get().then((querySnapshot) => {
        querySnapshot.forEach((doc) => {
            if (doc.id == 'OK' && doc.data().FUERA == 'OK') {
                db.collection(Usala).doc(Uuser).delete().then(() => {
                    console.log("Document successfully deleted!");
                    sessionStorage.clear();
                    window.location.href='menu2.html';
                });
            }else{
            }
        });
    });
    traer();
}
/*----------------Refrescar página automáticamente--------------------*/
```

La función “**SalirSala()**”, como su propio nombre indica, está vinculada a la opción “**Abandonar Sala**” y se encarga de borrar todas las variables de sesión y redirigir al usuario a la pantalla de “**Crear o unirse a una sala**”.

Antes de realizar dicha acción, se comprueba que el usuario que accede a esta opción sea o no el administrador de la sala. Se hace una consulta a la colección “**Administradores**” y se comprueba que el nombre de usuario que esta haciendo dicha consulta coincida con el nombre del administrador y, que la sala en la que se encuentra, coincida a su vez con la sala asociada al administrador.

De ser así, es borrado el creador de la sala de la colección “**Administradores**” y, posteriormente, son borradas todas las variables de sesión, su estancia en la base de datos y es redirigido fuera de la sala.

En caso de no ser el creador de la sala quien hace la consulta, solo serían borradas todas las variables de sesión, su estancia en la base de datos y es redirigido fuera de la sala.

```
427 function SalirSala(){
428
429     Usala=sessionStorage.getItem('NombreSala');
430     Uuser=sessionStorage.getItem('NombreUsuario');
431
432     db.collection('Administradores').get().then((querySnapshot) => {
433         querySnapshot.forEach((doc) => {
434             if (doc.id == Uuser && doc.data().NombreSala == Usala) {
435
436                 db.collection('Administradores').doc(Uuser).delete().then(() => {
437                   console.log("Document successfully deleted!");
438                 }).catch((error) => {
439                   console.error("Error removing document: ", error);
440                 });
441
442                 db.collection(Usala).doc(Uuser).delete().then(() => {
443                   console.log("Document successfully deleted!");
444                   sessionStorage.clear();
445                   window.location.href='menu2.html';
446                 }).catch((error) => {
447                   console.error("Error removing document: ", error);
448                 });
449
450             }else{
451                 db.collection(Usala).doc(Uuser).delete().then(() => {
452                   console.log("Document successfully deleted!");
453                   sessionStorage.clear();
454                   window.location.href='menu2.html';
455                 }).catch((error) => {
456                   console.error("Error removing document: ", error);
457                 });
458             }
459         });
460     });
461
462 }
```

Por último, la función “**CerrarSesion()**” borra todo tipo de variables de sesión y redirige al usuario al login principal.

```
464 function CerrarSesion(){
465     sessionStorage.clear();
466     window.location.href='../index.html';
467 }
```

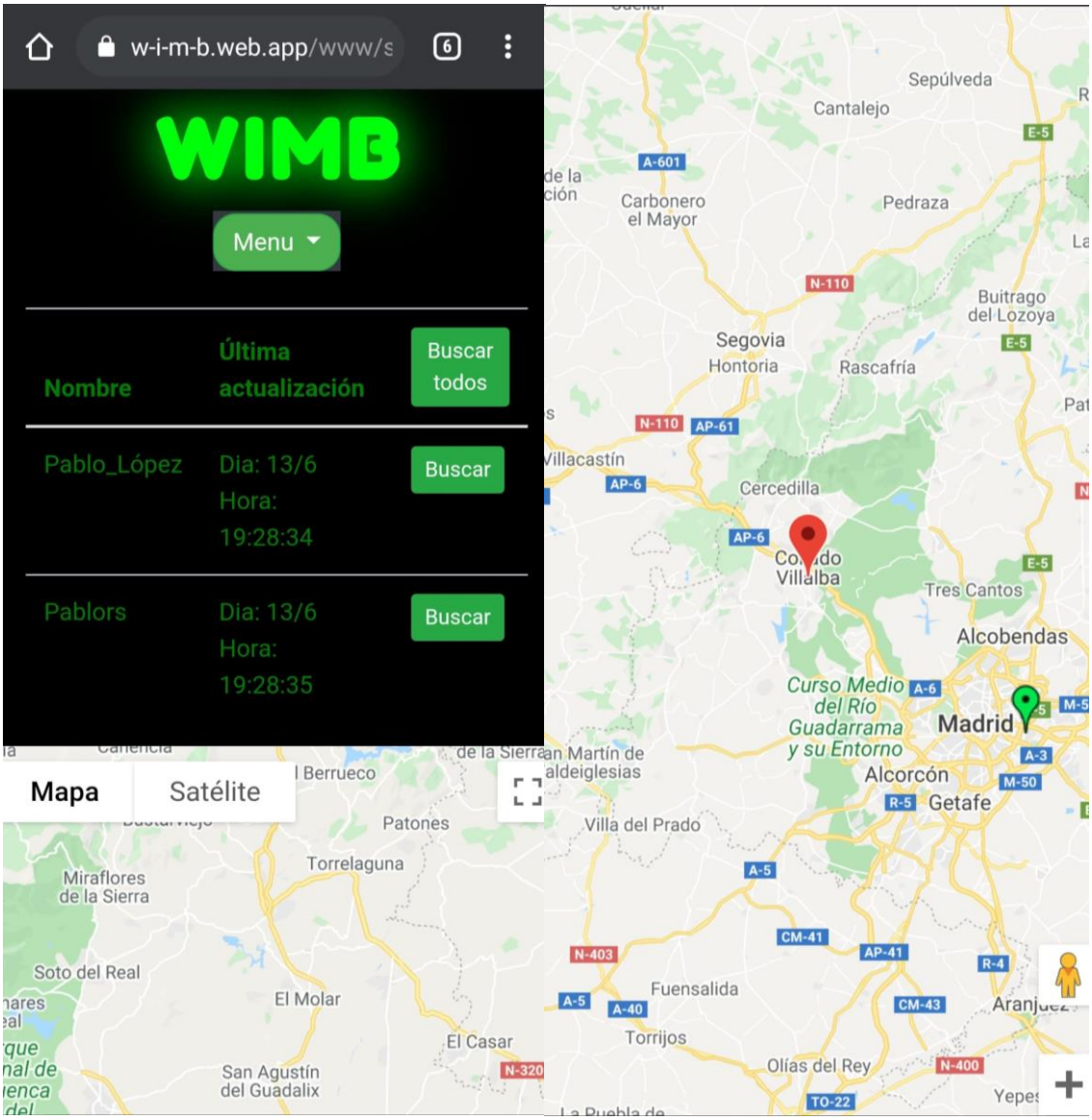
5.5. Buscar.

Este botón nos permite generar una marca individual en el mapa que nos muestra la ubicación de aquel usuario que queremos buscar.

Gracias a que nuestra marca personal tiene un color diferente al resto (verde), podemos identificar al usuario de una forma rápida y precisa.

También es posible pulsar sobre la marca que sitúa al usuario y comprobar que corresponde con el usuario buscado dado que se identifica por su nombre, latitud y longitud.

En el caso de que el usuario cambie su posición, puesto que este en movimiento, también lo hará la marca que lo identifica en el mapa.



5.5.1. Buscar. Código.

La función "**buscar(i)**" esta directamente asociada a los botones creados en la función "**traer()**". Al pasar a la función el parámetro "**i**", podemos identificar a los botones y trabajar directamente con su valor, es decir, el nombre del usuario asociado.

De esta manera, interactuamos con la base de datos gracias al nombre de la sala y al nombre del usuario a buscar, obtenemos los valores de latitud y longitud que le corresponden y los introducimos en las variables "**lat**" y "**long**" respectivamente.

Dichos datos se envían a la función "**buscarAmigo()**", a la cual se le pasan los parámetros "**lat**" y "**long**", que es la encargada de situar la marca del individuo en el mapa del usuario.

```
276  /*----------------------Función para buscar persona concreta--------------------*/
277  function buscar(i){
278      Usala=sessionStorage.getItem('NombreSala');
279      usr = document.getElementById(i).value;
280
281      sessionStorage.setItem('usr',usr);
282
283      var docRef = db.collection(Usala).doc(usr);
284
285      docRef.get().then((doc) => {
286          if (doc.exists) {
287              var lat = doc.data().Latitud;
288              var long = doc.data().Longitud;
289
290              buscarAmigo(lat,long);
291
292          } else {
293              console.log("No such document!");
294          }
295      }).catch((error) => {
296          console.log("Error getting document:", error);
297      });
298  }
299  /*----------------------Función para buscar persona concreta--------------------*/
```

La función “**buscarAmigo(lat,long)**” se basa unica y exclusivamente, en generear tanto la marca personal del usuario como la marca del individuo que se pretende localizar en el mapa.Ya que solo se pretende buscar a una única persona, el código de esta función está dividido en dos partes.

En primer lugar se obtiene la localización personal tras una consulta con la base de datos, gracias a los comentarios del código, se puede observar que este punto a tratar se encuentra entre “**Mi Localización**” y “**FIN Mi Localización**”. Tras la obtención de los datos de localización que corresponden con el usuario personal, se genera una marca verde en el mapa (**MARCA 1**).

```
/*----------------------Función para buscar persona MAPA--------------------*/
function buscarAmigo(Lat,Long){

    usr=sessionStorage.getItem('usr');

    Usala=sessionStorage.getItem('NombreSala');
    Uuser=sessionStorage.getItem('NombreUsuario');

    //-----------------Mi Localización-----------------
    var docRef = db.collection(Usala).doc(Uuser);

    docRef.get().then((doc) => {
        if (doc.exists) {
            var latitud = doc.data().Latitud;
            var longitud = doc.data().Longitud;

                /*------------------------------MARCA 1---------------------------------*/
                  const marker = new google.maps.Marker({

                    position: { lat: latitud, lng: longitud },
                    map: map,
                    icon: 'https://maps.google.com/mapfiles/ms/icons/green-dot.png',
                  });

                  const infowindow = new google.maps.InfoWindow({
                    content: "<p>Usted se encuentra aquí: " + marker.getPosition() + "</p>",
                  });
                  google.maps.event.addListener(marker, "click", () => {
                    infowindow.open(map, marker);
                  });
                /*----------------------------------------------------------------------*/
        }
    });
    //-----------------FIN Mi Localización-----------------
     //-----------------Localización Amigo------------------

    const mapOptions = {
        zoom: 10,
        center: { lat: lat, lng: long },
    };
    map = new google.maps.Map(document.getElementById("map"), mapOptions);

    /*------------------------------MARCA 2---------------------------------*/

    const marker2 = new google.maps.Marker({

      position: { lat: lat, lng: long },
      map: map,
    });
    const infowindow2 = new google.maps.InfoWindow({
      content: "<p>" + usr + " se encuentra aquí: " + marker2.getPosition() + "</p>",
    });
    google.maps.event.addListener(marker2, "click", () => {
      infowindow2.open(map, marker2);
    });
    /*----------------------------------------------------------------------*/
}
/*----------------------Función para buscar persona MAPA--------------------*/
```

Tras cumplirse este proceso, se genera la marca del individuo de color rojo gracias a los parámetros introducidos en la función (**lat** y **long**), evitando así tener que realizar otra consulta a la base de datos.

5.6. Buscar todos.

Este botón nos permite generar X marcas en el mapa dependiendo de los usuarios que se encuentren en la sala.

Dichas marcas sitúan a cada uno de los usuarios en el mapa y los identifica por su nombre.

En este caso, obtenemos 3 marcas diferentes, siendo la marca verde la manera de identificar al usuario que usa de forma directa la aplicación (usted), y las marcas en rojo el resto de miembros de la sala.

5.6.1. Buscar todos. Código.

En la función "**buscarTodos()**", el proceso de obtener los datos de ubicación y generar las marcas correspondientes es similar al de la función "**buscarAmigo()**". Se realiza una consulta a la base de datos, se obtiene la latitud y la longitud y se genera la marca.

Pero a diferencia de dicha función, no solo queremos buscar a un único individuo, el objetivo es buscar a todo aquel que se encuentre en la sala sin tener conocimiento de cuantos usuarios se encuentran en la misma. Para ello, se realiza una conexión con la base de datos hacia la sala en la que se encuentran y se genera un bucle que comprueba fila por fila la información. Con esto conseguimos realizar una acción por cada fila impresa.

De esta manera, creamos una condición para comprobar si el nombre del usuario que se esta consultando es el mismo nombre que aquel que hace la consulta, es decir, el objetivo es diferencial al propio usuario del resto de miembros de la sala. Si esta condición se cumple, que tiene que ser así, la marca personal será de color verde y el resto de marcas acorde con el resto de miembros, serán de color rojo.

```
//*********************************************************************
function buscarTodos(){

    Usala=sessionStorage.getItem('NombreSala');
    Uuser=sessionStorage.getItem('NombreUsuario');

//------------------------TRAER NOMBRE USUARIOS--------------------------------
    contenido.innerHTML = "";
    db.collection(Usala).get().then((querySnapshot) => {
        querySnapshot.forEach((doc) => {

//------------------------MOSTRAR MI LOCALIZACIÓN--------------------------------
            //usr = doc.id;
            var docRef = db.collection(Usala).doc(doc.id);
            if (doc.id == Uuser) {

                var docRef = db.collection(Usala).doc(Uuser);

                docRef.get().then((doc) => {
                    if (doc.exists) {
                        var latitud = doc.data().Latitud;
                        var longitud = doc.data().Longitud;

                                const marker = new google.maps.Marker({

                                  position: { lat: latitud, lng: longitud },
                                  map: map,
                                  icon: 'https://maps.google.com/mapfiles/ms/icons/green-dot.png',
                                });

                                const infowindow = new google.maps.InfoWindow({
                                  content: "<p>Usted se encuentra aquí: " + marker.getPosition() + "</p>",
                                });
                                google.maps.event.addListener(marker, "click", () => {
                                  infowindow.open(map, marker);
                                });
                    }
                });
//----------------------MARCAR POSICIÓN DE CADA USUARIO----------------------
            }else{
                docRef.get().then((doc) => {
                    if (doc.exists) {
                        var latitud = doc.data().Latitud;
                        var longitud = doc.data().Longitud;

                                const marker2 = new google.maps.Marker({

                                  position: { lat: latitud, lng: longitud },
                                  map: map,
                                });

                                const infowindow = new google.maps.InfoWindow({
                                  content: "<p>" + doc.id + " se encuentra aqui: " + marker2.getPosition() + "</p>",
                                });
                                google.maps.event.addListener(marker2, "click", () => {
                                  infowindow.open(map, marker2);
                                });
                    }
                });
            }
        });
    });
}
```

5.7. Área de Administradores.

Como se ha mencionado anteriormente, solo el creador de la sala es capaz de acceder al "**Área de administradores**" desplegando el menú como se muestra en la imagen de la derecha.

Una vez dentro, el administrador puede realizar diversas tareas que le corresponden.

El botón "**Expulsar todos**", como su propio nombre indica, expulsa a todos los miembros de la sala exceptuando al propio creador de la sala y dejándola totalmente inaccesible al público. Más tarde será mostrado cómo el administrador puede liberar la sala y dejarla accesible al público.

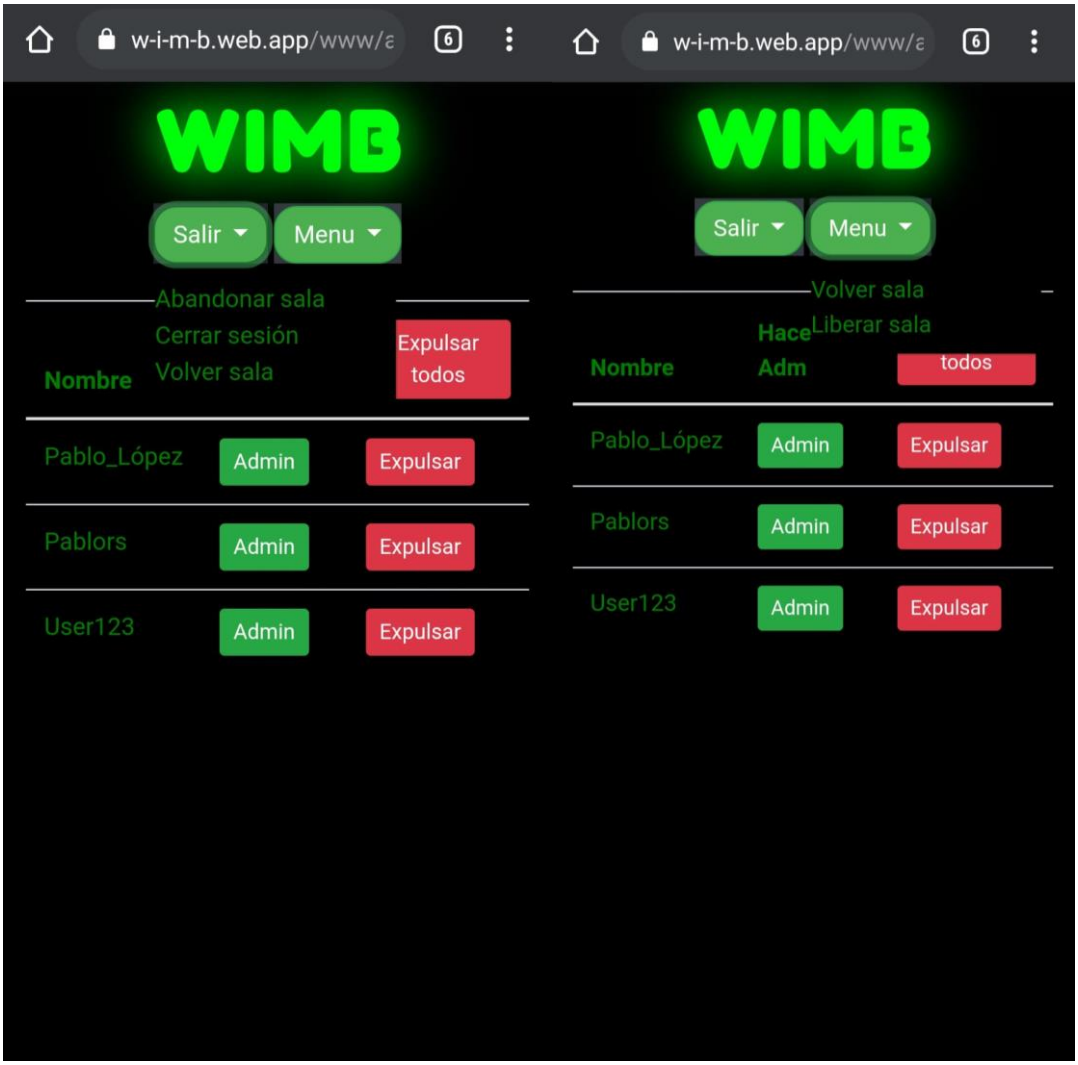
También existen botones de "**Expulsar**" individuales, destinados únicamente a borrar registros y redirigir al usuario fuera de la sala. Bloqueando así su acceso.

Por último, dentro de las funcionalidades visibles e importantes del administrador, cabe destacar el botón de "**Admin**" en cada uno de los miembros de la sala.

Este botón permite al administrador crear múltiples administradores y así, compartir su rol y permitir que, aquellos usuarios que hayan recibido este cambio, puedan acceder al "**Área de administradores**" y, por lo tanto, ser capaces de utilizar dichas funcionalidades.

Por último, cabe destacar los menús desplegables que tiene el "**Área de administradores**".

En el botón "**Salir**", se muestran las mismas funcionalidades que tiene el botón "**Menu**" de la sala principal, pero uno de las opciones para de ser "**Administrador**" a "**Volver a sala**". Esta opción únicamente te redirige a la sala principal y cierra el "**Área de administradores".**



5.7.1. Área de administradores. Código.

Para comenzar a hablar sobre esta área, cabe destacar la función "**adm()**". El objetivo de esta función simplemente es comprobar que, aquella persona que esta intentando acceder al "**Área de administradores**", tenga permiso para ello.

Se realiza una consulta a la colección "**Administradores**" de la base de datos y, si el usuario y la sala coinciden con los datos existentes en dicha colección, se concede el acceso y se redirige al usuario al "**Área de administradores**".

De no ser así, la opción aparenta ser inservible a ojos del usuario.

```
17  //FUNCIÓN PARA COMPROBAR SI EL USUARIO QUE INTENTA
18  // ENTRAR AL ÁREA DE ADMINISTRADORES TIENE PERMISOS
19  function adm(){
20
21      Usala=sessionStorage.getItem('NombreSala');
22      Uuser=sessionStorage.getItem('NombreUsuario');
23
24      db.collection('Administradores').get().then((querySnapshot) => {
25          querySnapshot.forEach((doc) => {
26              if (doc.id == Uuser && doc.data().NombreSala == Usala) {
27                  window.location.href='adm.html';
28              }else{
29                  //QUITO ALERT PORQUE COMPRUEBA TODOS LOS NOMBRES EN LA
30                  // SALA ADMINISTRADORES Y AQUEL QUE NO COINCIDE DEVUELVE ALERT
31                  //alert("Permiso denegado");
32              }
33          });
34      });
35  }
```

Una vez ha sido concedido el acceso a dicha área, entra en juego la función "**admTraer()**" que, a diferencia de "**traer()**", no se llama automáticamente, solo realiza la consulta y la impresión de los datos, sustituye el botón "**Buscar**" por "**Expulsar**" y la columna "**Última actualización**" por el botón "**Admin**".

La asignación de identificadores, valores y funciones en los botones es similar a la realizada en la función "**traer()**" a la hora de generar el botón "**Buscar**".

El botón "**Admin**" tiene asociada la función "**hacerAdm()**" y el botón "**Expulsar**" esta directamente asociado a la función "**admExpulsar()**".

En esta área no es necesario actualizar constantemente la pantalla ni tampoco mostrar el mapa, dado que no es su objetivo.

```
37  //FUNCIÓN PARECIDA A traer(), EN VEZ DE TRAER 'Última actualización' TRAER BOTON 'Admin'.
38  function admTraer(){
39
40      Usala=sessionStorage.getItem('NombreSala');
41      Uuser=sessionStorage.getItem('NombreUsuario');
42      var i=0;
43
44      contenido.innerHTML = "";
45      db.collection(Usala).get().then((querySnapshot) => {
46          querySnapshot.forEach((doc) => {
47              document.getElementById("contenido").innerHTML += ("<tr><td>"+doc.id+
48                  "</td><td><button onclick='hacerAdm("+i+")' id="+i+" value="+doc.id+
49                  " class='btn btn-success btn-sm'>Admin</button></td><td><button onclick='admExpulsar("+i+")' id="+i+
50                  " value="+doc.id+" class='btn btn-danger btn-sm'>Expulsar</button></td></tr>");
51              i = i+1;
52          });
53      });
54  }
```

La función “**hacerAdm()**” unicamente altera el estado de la base de datos y añade el nombre del usuario asociado al botón a la colección “**Administradores**”, junto con el nombre de la sala.

De esta forma, un usuario corriente pasaría a cumplir el rol de administrador y podría acceder al “**Área de administradores**” y realizar cualquier acción correspondiente.

```
//FUNCIÓN hacerAdm() QUE RECOGE VALOR i DE LA FUNCIÓN admTraer() DEL BOTON PARA CREAR ADMINISTRADOR
function hacerAdm(i){

    Usala=sessionStorage.getItem('NombreSala');
    usr = document.getElementById(i).value;

    sessionStorage.setItem('usr',usr);

    var docRef = db.collection('Administradores').doc(usr);
    docRef.set({
        NombreSala: Usala
    });

}
```

Aparentemente, la función “**admExpulsar()**” elimina las variables de sesión del individuo a expulsar y borra su estancia en la base de datos sin dejar registro alguno.

```
//FUNCIÓN QUE RECOGE EL VALOR DE LA VARIABLE i PROCEDENTE DE LA FUNCIÓN admTraer()
// DEL BOTON EXPULSAR PARA BORRAR LOS DATOS DEL
//USUARIO DE LA BASE DE DATOS. DE ESTA MANERA, LA CONDICION DE LA FUNCIÓN traer() SE CUMPLE
function admExpulsar(i){

    Usala=sessionStorage.getItem('NombreSala');
    usr = document.getElementById(i).value;

    sessionStorage.setItem('usr',usr);

  db.collection(Usala).doc(usr).delete().then(() => {
      console.log("Document successfully deleted!");

  }).catch((error) => {
      console.error("Error removing document: ", error);
  });
// admExpulsar(i) borra el documento que tiene el nombre del usuario actual en la base de datos.

// traer(), tras mostrar todos los usuarios que se encuentran en la sala (cada 6 segs) y añadirlos a un array,
//          tiene una condicion que comprueba si existe el nombre del usuario en dicho array, y si no es asi le echa

// traer() es llamada cada 6 segundos gracias a la función refrescar().

// refrescar() tiene una condición que comprueba la existencia de un documento llamado 'OK'.
//  Si 'OK' existe, no llama a la función traer()
// sino que directamente borra la sesion del usuario actual y lo redirecciona (todos los usuarios).
}
```

Una vez el administrador pulsa este botón y se realizan las acciones determinadas en la función correspondiente, el usuario que ha sido “**Expulsado**” no tiene ningún tipo de registro en la base de datos de la sala, es decir, aparentemente no existe. Esto no quiere decir que el usuario no se encuentre en la sala, quiere decir que él es capaz de ver la misma pero no aparece en la base de datos.

Recordamos que existen dos funciones que son llamadas de forma automática cada 15 segundos (**actualizar**) y 5 segundos (**refrescar**).

Cada 15 segundos, todo aquel que se encuentre físicamente en la sala, envía tanto sus datos de ubicación como el instante en el que es compartido a la base de datos. Esto significa que el usuario, que aun se encuentra en la sala físicamente, volvería a generar sus datos en la base de datos y, por lo tanto, no habría servido de nada el botón “**Expulsar**”.

Aquí entra en juego el “efecto mariposa” que hemos creado.

Volviendo a la función “**traer()**”, la cual es llamada cada 5 segundos por la función “**refrescar()**”, hemos creado una condición que se comprueba cada vez que se llama a la función.

En explicaciones anteriores, esta función únicamente obtiene e imprime los datos de la base de datos. Pero cabe destacar que se realizan una serie de comprobaciones que, al no cumplirse, parecen no existir.

Dichas comprobaciones son las siguientes:

Se crea un array de usuarios llamado “**usuarios[]**”. Tras realizar su cometido principal, que es consultar e imprimir usuarios, se añade el nombre de cada uno de los miembros de la sala a dicho array.

Más tarde se crea la constante “**found**” la cual posee una función que busca el nombre del propio usuario dentro de los elementos que componen el array.

En caso de que dicha “consulta” no encuentre al propio usuario dentro de los elementos del array, la constante “**found**” pasa a tener el valor “**undefined**”.

Esto quiere decir que el propio usuario no aparece en la lista de usuarios de la base de datos. Al cumplirse la condición de que “**found**” vale “**undefined**”, se borran todas las variables de sesión y se redirige al usuario fuera de la sala.

Todo esto es posible gracias al intervalo de tiempo en el que se llama a esta función, es decir, existen 10 segundos de diferencia entre los que esta función es llamada antes que la función que envía los datos de ubicación y actualización a la base de datos.

Aprovechando ese “vacio temporal”, podemos realizar estas comprobaciones y expulsar al usuario antes de que se repita el proceso de actualización de datos.

```
183  //----------------------Función traer datos-------------------------
184  //ACCEDE E IMPRIME TODOS LOS USUARIOS Y SUS CORRESPONDIENTES HISTORICOS CONSTANTEMENTE AL NO SER QUE LA CONDICIÓN
185  // DE LA FUNCIÓN refrescar() SE HAYA CUMPLIDO
186  function traer(){
187
188      Usala=sessionStorage.getItem('NombreSala');
189      Uuser=sessionStorage.getItem('NombreUsuario');
190
191      var i=0;
192      let usuarios=[];
193
194      contenido.innerHTML = "";
195      db.collection(Usala).get().then((querySnapshot) => {
196          querySnapshot.forEach((doc) => {
197              document.getElementById("contenido").innerHTML += ("<tr><td>"+doc.id+"</td><td>"
198                  +doc.data().Actualizado+"</td><td><button onclick='buscar("+i+")' id="+i+
199                  " value="+doc.id+" class='btn btn-success btn-sm'>Buscar</button></td></tr>");
200              i = i+1;
201              let newLength = usuarios.push(doc.id);
202              //console.log(usuarios);
203          });
204      //BUSQUEDA DEL NOMBRE DEL USUARIO DENTRO DEL ARRAY GENERADO AL LISTAR LOS USUARIOS DE LA SALA
205      const found = usuarios.find(element => element == Uuser);
206
207      //CONDICION: SI EL USUARIO NO SE ENCUENTRA Y POR LO TANTO NO ESTÁ EN LA CONSTANTE FOUND, EXPULSA AL MISMO
208      if(found == undefined){
209          sessionStorage.clear();
210          window.location.href='menu2.html';
211      }
212
213      });
214  }
215  //*********************************************************************
```

La función “**admCerrarSala()**” está asociada al botón “**Expulsar todos**”. La única acción a realizar es crear un documento dentro de la sala que recibe el nombre ‘**OK’**.

```
// FUNCIÓN QUE CREA UN DOCUMENTO LLAMADO 'OK' PARA QUE SE CUMPLA LA CONDICION DE LA FUNCIÓN refrescar().
function admCerrarSala(){

    Usala=sessionStorage.getItem('NombreSala');
    Uuser=sessionStorage.getItem('NombreUsuario');

    var docRef = firestore.collection(Usala).doc('OK');

        docRef.set({
            FUERA: 'OK'
        });
    admTraer();
}
```

Aparentemente, esta función no tiene mucha utilidad, pero vuelve a entrar en juego el “efecto mariposa”.

Haciendo uso, una vez más, de la función que llamada cada 5 segundos “**refrescar()**”, creamos una condición que siempre se comprueba y que, al no cumplirse normalmente, pasa directamente a la función “**traer()**”.

Dicha condición comprueba la existencia de un documento llamado “**OK**” en la colección que recibe el nombre de la sala. Dado que esta alteración afecta a todos los usuarios que se encuentran en la sala, en el momento en el que se cumple la condición, se borran todas las variables de sesión y se redirige al usuario fuera de la sala.

Gracias a que el administrador en ese momento se encuentra en el “**Área de administradores**”, esta función no le afecta ya que solo es llamada en la sala común.

Esto deja inaccesible la sala puesto que, si cualquier usuario intenta acceder a la misma, lo logrará tan solo por unos segundos. Luego será expulsado automáticamente al cumplirse la condición de la función “**refrescar()**” llamada a los 5 segundos.

```
//FUNCIÓN LLAMADA A SI MISMA CADA 5 SEGUNDOS. COMPRUEBA LA EXISTENCIA DE UN DOCUMENTO 'OK',
// EN CASO DE SER ASI, BORRA AL USUARIO DE LA SALA
// Y LA CONDICIÓN DE LA FUNCIÓN traer() PASA A LA ACCIÓN.
//EN CASO DE NO CUMPLIRSE LA CONDICIÓN, PASA DIRECTAMENTE A EJECUTAR LA FUNCIÓN traer() SIN BORRAR DATOS.

function refrescar(){
console.log("Refrescar");
    Usala=sessionStorage.getItem('NombreSala');
    Uuser=sessionStorage.getItem('NombreUsuario');

    db.collection(Usala).get().then((querySnapshot) => {
        querySnapshot.forEach((doc) => {
            if (doc.id == 'OK' && doc.data().FUERA == 'OK') {
                db.collection(Usala).doc(Uuser).delete().then(() => {
                    console.log("Document successfully deleted!");
                    sessionStorage.clear();
                    window.location.href='menu2.html';
                });
            }else{
            }
        });
    });
    traer();
}
/*----------------Refrescar página automáticamente--------------------*/
```

Como comentábamos, la sala queda inaccesible al público pero el administrador sigue en el "**Área de administradores**" correspondiente a la sala.

Si decide liberar la sala y, por lo tanto, que vuelva a estar accesible al público, únicamente debe pulsar sobre el botón "**Liberar sala**" que se encuentra dentro del "**Menu**".

Esta opción esta asociada a la función "**liberar()**" la cual elimina el documento "**OK**" de la colección y en consecuencia, dejaria de cumplirse la condición de la función "**refrescar()**" y volvería a ser posible el acceso a la sala.

```
131  //FUNCIÓN QUE BORRA EL DOCUMENTO 'OK' QUE BLOQUEA LA SALA
132  function liberar(){
133      db.collection(Usala).doc('OK').delete().then(() => {
134          console.log("Document successfully deleted!");
135      }).catch((error) => {
136          console.error("Error removing document: ", error);
137      });
138
139      admTraer();
140  }
```

La función "**admSalirSala()**" está asociada a la opción "**Abandonar Sala**". Como su nombre indica el objetivo es salir de la sala borrando las variables de sesión y redirigiendo al usuario.

Antes de realizar esta acción, se borran todos los datos del administrador de la colección "**Administradores**", para así evitar problemas a la hora de crear o unirse a otra sala.

```
95   // BORRA AL USUARIO ADMIN DE 'Administradores' Y SALE DE LA SALA
96   function admSalirSala(){
97
98     Usala=sessionStorage.getItem('NombreSala');
99     Uuser=sessionStorage.getItem('NombreUsuario');
100
101    db.collection('Administradores').doc(Uuser).delete().then(() => {
102        console.log("Document successfully deleted!");
103    }).catch((error) => {
104        console.error("Error removing document: ", error);
105    });
106
107    //MANTENER DOCUMENTO OK PARA DEJAR INACCESIBLE LA SALA
108
109    /*db.collection(Usala).doc('OK').delete().then(() => {
110        console.log("Document successfully deleted!");
111    }).catch((error) => {
112        console.error("Error removing document: ", error);
113    });*/
114    //-----------------------------------------------------
115
116    db.collection(Usala).doc(Uuser).delete().then(() => {
117        console.log("Document successfully deleted!");
118        sessionStorage.clear();
119        window.location.href='menu2.html';
120    }).catch((error) => {
121        console.error("Error removing document: ", error);
122    });
123
124  }
125
```

6. Dificultades.

6.1. Android Studio y el compilador Gradle

El principal problema que nos surgió fue en base al desconocimiento del IDE. Una vez estudiado el entorno tuvimos que investigar acerca del compilador Gradle. Tras comprender estas tecnologías, nos encontramos con el siguiente dilema.

6.2. Apache Cordova

Apache Cordova es un framework que ofrece un conjunto de APIs JavaScript que permiten que las aplicaciones puedan acceder a los diferentes elementos de dispositivos móviles. Para su desarrollo, se necesita este framework que hace posible la comunicación entre las tecnologías web, con la plataforma nativa del dispositivo.

Quisimos utilizar Apache Cordova para la utilización de plugins que conectaran nuestra aplicación con el lenguaje nativo del sistema dispositivo móvil, estos posibilitan el acceso a las características de nuestra aplicación a través del propio dispositivo, es necesario apuntar que sin el uso de los plugins la aplicación no podría funcionar ya que no podría comunicarse con el dispositivo.

Finalmente, cambiamos nuestra primera idea de proyecto y pasamos de aplicación móvil a página web, si bien el plugin "**cordova-plugin-firestore**" funcionaba correctamente, el plugin "**cordova-plugin-firebaseui-auth**" estaba obsoleto y no encontramos la manera de solucionar este problema.

Sin embargo, esto no fue un obstáculo, pues el objetivo de los plugins se puede solventar perfectamente escribiendo las funciones propias de Firebase, en nuestro código, que nos permiten usar sus productos sin necesidad de dichos plugins. Permitiéndonos trabajar directamente con los productos de autenticación y de bases de datos de esta plataforma.

6.3. Manejo de variables entre funciones:

Desde un principio, contábamos con un problema que repercute en toda la infraestructura y funcionalidad de nuestra web, el manejo de variables entre funciones.

No éramos capaces de mantener lo que se conoce como una sesión dado que todas las variables utilizadas sólo podían ser creadas dentro de funciones, lo cual nos hacía imposible utilizar las mismas en el resto de funciones.

En un principio solicitamos los mismos datos a medida que el usuario avanzaba de una página a otra o de una funcionalidad a otra. Esto supone un fallo de seguridad importante y una acción tediosa para aquel que quisiera utilizar nuestros servicios.

Tras un estudio intenso del problema y muchas pruebas, descubrimos la existencia de las "sessionStorage". Variables capaces de funcionar como las variables de sesión que tiene PHP. No solo habíamos solucionado el problema de manejar dichas variables en cualquier función, sino que también fuimos capaces de crear una "sesión de usuario".

7. Bibliografía.

Gradle: https://gradle.org/

Android studio:
https://developer.android.com/studio?hl=es&gclid=CjwKCAjwwqaGBhBKEiwAMk-FtHlljIcjFBdNzi755IoO4bNvA-ghCiACh7FvNbHT4Mc-je_biOVpxRoCFPYQAvD_BwE&gclsrc=aw.ds

Información Android estudio:
https://www.youtube.com/watch?v=BQaxPwZWboA&ab_channel=MoureDevbyBraisMoure

Android Studio y Kotlin:
https://www.youtube.com/watch?v=KYPc7CAYJOw&ab_channel=MoureDevbyBraisMoureMoureDevbyBraisMoureVerificada

Firestore: https://firebase.google.com/docs/firestore/quickstart

Authentication: https://firebase.google.com/docs/auth

Hosting: https://firebase.google.com/docs/hosting

Analytics: https://firebase.google.com/docs/analytics

Manejo de datos en Firestore:
https://www.youtube.com/watch?v=itNsRn1kjLU&ab_channel=JohnOrtizOrdo%C3%B1ezJohnOrtizOrdo%C3%B1ez

Geolocation API:
https://developers.google.com/maps/documentation/javascript/overview?hl=es

Maps JavaScript API:
https://developers.google.com/maps/documentation/javascript/examples/marker-labels?hl=es

Geolocalización:
https://www.evaluandosoftware.com/la-geolocalizacion-funciona/
https://es.wikipedia.org/wiki/Geolocalizaci%C3%B3n

Boostrap:
https://getbootstrap.com/
https://www.w3schools.com/bootstrap4/

Clases de Boostrap:
https://www.w3schools.com/bootstrap/bootstrap_ref_all_classes.asp