

# MEMORIA FINAL DE PROYECTO

VAOHLAND

CICLO FORMATIVO DE GRADO SUPERIOR

DESARROLLO DE APLICACIONES WEB

AUTOR

Javier Borbolla Ureña

TUTOR

Jesús Vives Céspedes

COORDINADOR

Jesús Vives Céspedes

# Índice

<b>1. INTRODUCCIÓN</b>	<b>4</b>
<b>2. ALCANCE DEL PROYECTO</b>	<b>5</b>
<b>3. ESTUDIO DE VIABILIDAD</b>	<b>6</b>
3.1. Estado actual del sistema . . . . .	6
3.2. Requisitos del cliente . . . . .	6
3.3. Posibles soluciones . . . . .	6
3.4. Solución elegida . . . . .	6
3.5. Planificación temporal de las tareas del proyecto Vaohland . . . . .	7
3.6. Planificación de los recursos a utilizar . . . . .	7
<b>4. ANÁLISIS</b>	<b>8</b>
4.1. Requisitos funcionales . . . . .	8
4.2. Requisitos no funcionales . . . . .	8
<b>5. DISEÑO</b>	<b>9</b>
5.1. Estructura de la aplicación . . . . .	9
5.2. Componentes del sistema . . . . .	9
5.3. Herramientas . . . . .	10
<b>6. IMPLEMENTACIÓN</b>	<b>11</b>
6.1. Entorno de implementación . . . . .	11
6.2. Tablas creadas . . . . .	12
6.3. Carga de datos . . . . .	13
6.4. Configuraciones realizadas en el sistema . . . . .	13
<b>7. Implentaciones de código realizadas</b>	<b>14</b>
<b>8. PRUEBAS</b>	<b>15</b>
8.1. Casos de prueba . . . . .	15
<b>9. EXPLOTACIÓN</b>	<b>18</b>
9.1. Planificación . . . . .	18

9.2. Plan de formación . . . . .	18
9.3. Pruebas de implantación . . . . .	19
<b>10. DEFINICIÓN DE PROCEDIMIENTOS DE CONTROL Y EVALUACIÓN</b>	<b>20</b>
<b>11. CONCLUSIONES</b>	<b>21</b>
<b>12. FUENTES</b>	<b>22</b>
<b>13. ANEXOS</b>	<b>23</b>
13.1. Manual de usuario . . . . .	23
13.1.1. Registro . . . . .	23
13.1.2. Login . . . . .	24
13.1.3. Index . . . . .	25
13.1.4. Modales . . . . .	25
13.1.5. Botones en publicaciones . . . . .	27
13.1.6. Buscar . . . . .	28
13.1.7. Perfiles . . . . .	29
13.1.8. Mensajes directos (MD) . . . . .	30
13.1.9. Chat Global . . . . .	31
13.2. Manual técnico . . . . .	32
13.2.1. Cliente – chatGlobal.js . . . . .	32
13.2.2. Cliente – index.js . . . . .	35
13.2.3. Cliente – library.js . . . . .	37
13.2.4. Cliente – login.js . . . . .	43
13.2.5. Cliente – md.js . . . . .	44
13.2.6. Cliente – nick.js . . . . .	45
13.2.7. Cliente – register.js . . . . .	47
13.2.8. Cliente – search.js . . . . .	48
13.2.9. Servidor – scriptNode.js . . . . .	50



# 1. INTRODUCCIÓN

Este documento recoge y describe el estudio realizado sobre la viabilidad, el diseño, la planificación, el desarrollo y la implementación del trabajo realizado para el módulo de Proyecto del Ciclo Formativo de Grado Superior en Desarrollo de Aplicaciones Web.

A continuación se determinará que se va a realizar y como se desarrollará:

## **Estudio de viabilidad**

Se definen los requisitos del cliente, se comentan las posibles soluciones y se desarrolla la opción más viable. Además, se planificarán las tareas a realizar para el desarrollo del proyecto.

## **Análisis**

Describe las partes funcionales y no funcionales de la aplicación.

## **Diseño**

Describe la estructura de la aplicación, los componentes y las herramientas del sistema (programación, gestión de datos y despliegue).

## **Implementación**

Se describe el entorno que será usado al desplegar la aplicación.

## **Pruebas**

Se proporcionan ejemplos de diferentes casos de prueba.

## **Explotación**

Se explican los recursos y tareas necesarias para la implementación de la aplicación.

## 2. ALCANCE DEL PROYECTO

Este proyecto se lleva a cabo para crear una nueva red social.

En la parte pública estarán los usuarios. Estos podrán:

Compartir contenido

Seguir perfiles

Dar me gusta/no me gusta a las publicaciones

Comentar publicaciones

Tener chats privados

Hablar en un chat global

Además, tendrán a su disposición una sección de videojuegos en la que podrán jugar y comentar entre ellos a tiempo real las jugadas.

En la parte de administración se encontrará el soporte técnico. Los administradores podrán:

Eliminar publicaciones

Dar de baja/alta a clientes existentes

Eliminar mensajes enviados en el chat global

### **3. ESTUDIO DE VIABILIDAD**

En este apartado se considera si el proyecto es viable, o no, basándose en las circunstancias internas y externas de la empresa, las diferentes soluciones posibles y los recursos de los cuales se dispone. Para ello se hace una valoración del estado actual del sistema y de los requisitos del cliente, se presentará un estudio de soluciones alternativas y la solución elegida por el cliente.

#### **3.1. Estado actual del sistema**

La red social se sustenta en un servidor contratado a la empresa OVH. Esto facilita la conectividad a nivel global con todos los dispositivos.

#### **3.2. Requisitos del cliente**

El cliente requiere una aplicación web donde se permita compartir, valorar y comentar mensajes publicados por cualquier usuario. También se pide que se tenga la opción de seguir a los usuarios que se desee para estar al tanto de todas las publicaciones que haga. En caso de querer hablar con un usuario de forma privada, se creará un chat privado. Además, tiene que ser un servicio 24/7 al que se pueda acceder desde cualquier parte del mundo.

#### **3.3. Posibles soluciones**

Las herramientas que podrían cumplir con los requisitos serían:

Un servidor con MongoDB para un rápido almacenamiento.

El uso de sockets entre el cliente y el servidor para el uso de chats instantáneos.

#### **3.4. Solución elegida**

La solución más viable es crear la red social desde cero con las siguientes tecnologías: Node, MongoDB, Socket.io y Javascript.

### 3.5. Planificación temporal de las tareas del proyecto Vaohland

Se tardará alrededor de 28 días para realizar todo el proyecto.

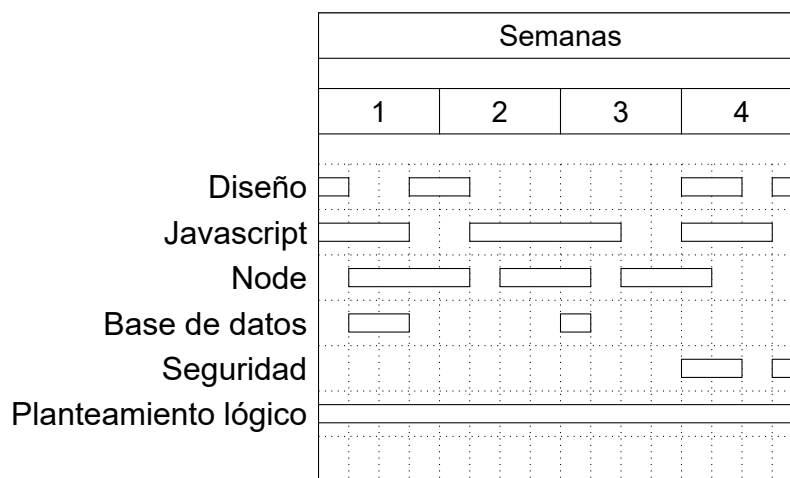
El trabajo se dividirá en: front-end, back-end, planteamiento lógico y documentación.

Todo el proyecto será realizado por un solo miembro: Javier Borbolla Ureña.

En la parte front-end tendremos la parte de diseño con CSS y la parte funcional con Javascript.

En la parte back-end tendremos MongoDB como la BBDD y Node en la parte funcional.

Podríamos dividir de la siguiente forma el tiempo planteado que llevará cada sección durante las 4 semanas de producción:



### 3.6. Planificación de los recursos a utilizar

Para llevar a cabo este proyecto se necesitará un ordenador en el que desarrollará y se realizarán las pruebas y, un servidor en el que desplegar la aplicación una vez terminada.

Tanto en el ordenador como en el servidor se precisará de la instalación de MongoDB y Node. En el ordenador donde se trabajará también se precisará la instalación del editor de texto "Atom".



## 4. ANÁLISIS

En este apartado se describen los requisitos principales de la aplicación, expresados por el cliente.

### 4.1. Requisitos funcionales

La aplicación ha de:

- Tener un servicio 24/7.
- Ser accesible desde cualquier parte del mundo.
- Mostrar publicaciones de los usuario seguidos.
- Poder seguir a usuarios.
- Valorar y comentar publicaciones.
- Tener chats privados y uno público.
- Tener una sección exclusiva para administradores.

### 4.2. Requisitos no funcionales

- El sistema ha de ser compatible visualmente con los navegadores y dispositivos más usados.
- El servidor que mantenga la aplicación ha de tener un tiempo de respuesta casi instantáneo.
- Debe de ser intuitiva.

## 5. DISEÑO

En este apartado se describe la estructura de la aplicación.

### 5.1. Estructura de la aplicación

VaohLand se divide en dos partes: Panel de administración y Red social.

#### **Panel de administración**

Parte exclusiva de la aplicación limitada a los administradores. Estos podrán eliminar cualquier cosa publicada y dar de baja o alta a los usuarios existentes. Tendrán un buscador para buscar por nick o por texto de publicación.

#### **Red social**

Parte usada por los clientes. Aquí es donde los usuarios interactuarán con otros usuarios y publicarán mensajes.

### 5.2. Componentes del sistema

El trabajo se compondrá por:

- BBDD (MongoDB). Mantenido en el mismo servidor donde se ejecuta la parte servidora.
- Un servidor web que mantenga tanto la aplicación como la BBDD.
- Los navegadores de los usuarios. Estos navegadores deberán de soportar CCS3, HTML5 y Javascript.

Los clientes realizarán peticiones al servidor a través de la interfaz de la red social. El servidor realizará peticiones a la BBDD y responderá a la petición del cliente.

### **5.3. Herramientas**

- Se usará Atom como editor de texto.
- MongoDB como gestor de bases de datos.
- MongoDB Compass Community para una fácil visualización de los datos de la BBDD.
- Chrome/Firefox para visualizar la aplicación.
- Node para ejecutar el lado servidor.
- Filezilla para transferir los archivos al servidor.

## 6. IMPLEMENTACIÓN

### 6.1. Entorno de implementación

El entorno en el que será implementado será un entorno de producción habitual, donde el cliente se conecta a internet y llega al servidor.

Como no se dispone de un presupuesto elevado, la base de datos se encontrará en el mismo servidor donde se ejecutará todo.

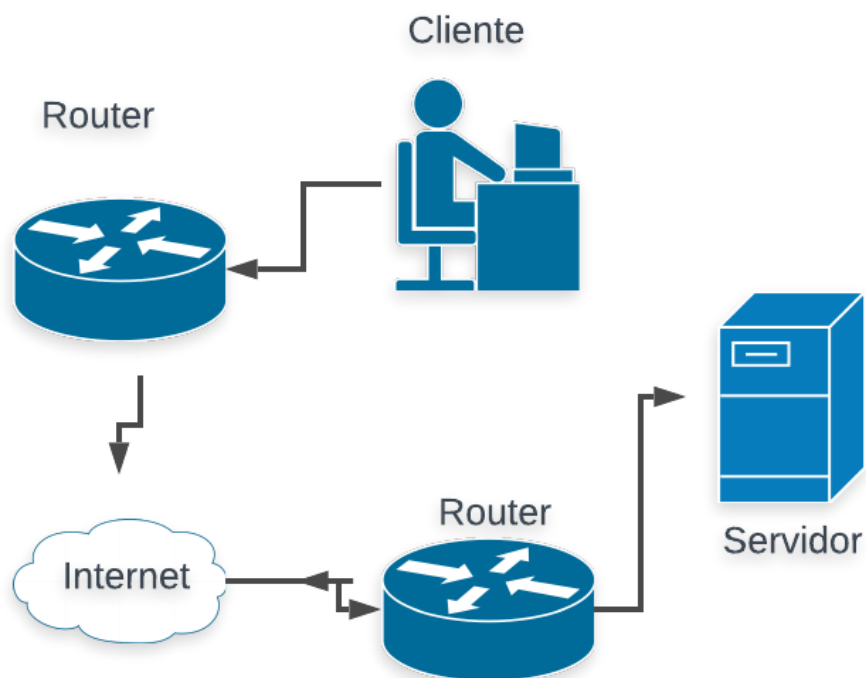


Figura 1: Entorno en el que trabajará la app.

## 6.2. Tablas creadas

La base de datos se define en el mismo fichero en el que se ejecuta el servidor ("scriptNode.js").

**Follows:**

nick\_cliente1: String,  
nick\_cliente2: String,  
date: type: Date, default: Date.now

**Clientes:**

nick: type: String, unique: true,  
password: String,  
date: type: Date, default: Date.now ,  
fecha\_cumple: Date,  
img\_perfil: type: String, unique: true,  
biografia: String,  
estado: Boolean

**Likes:**

nick\_cliente: String,  
id\_post: String

**Dislikes:**

nick\_cliente: String,  
id\_post: String

**Admin\_Users:**

nick: type: String, lowercase: true, unique: true,  
password: String

**Posts:**

nick\_cliente: String,  
img\_perfil: String,  
texto: String,  
likes: Number,  
dislikes: Number,  
fecha\_creacion: type: Date, default: Date.now,  
id\_post\_comentado: String,  
nick\_cliente2: String,  
texto2: String

**Chat\_Directo:**

nick\_cliente1: String,  
nick\_cliente2: String,  
texto: String,  
fecha\_creacion: type: Date, default: Date.now

**Chat\_Global:**

nick\_cliente: String,  
texto: String,  
fecha\_creacion: type: Date, default: Date.now

Figura 2: Estructura de MongoDB

Exceptuando la colección "Admin\_users" , todas son accesibles para el cliente.

### 6.3. Carga de datos

El único dato que es cargado en el usuario y contraseña de un usuario administrador. Este dato es cargado en el archivo ("scriptNode.js").

### 6.4. Configuraciones realizadas en el sistema

Para que la BBDD no sea accesible a todos los usuario, se ha creado un usuario:contraseña con el que acceder a mongo. Esta configuración se ha realizado desde un cmd de la siguiente forma:

#### Mongo

#### Use admin

```
db.createUser(user: "NOMBREDEUSUARIO",pwd: "CONTRASEÑA",roles: [ role: "userAdminAnyDatabase", db: "admin" , "readWriteAnyDatabase" ])
```

Además de configurar la BBDD para no poder acceder a ella de forma anónima, se han configurado todos los archivos JS para que los sockets apunten al servidor.

## 7. Implementaciones de código realizadas

VaohLand se ha programado bajo la tecnología de node. Node es una tecnología que se basa en JS en la que es frecuente usar módulos creados por la comunidad. En este servidor se han usado diferentes módulos como:

**“mongoose”**, para el manejo de la base de datos.

**“express”**, para el enrutamiento de la aplicación.

**“socket.io”**, para la conexión instantánea entre el cliente y servidor.

**“bcrypt”**, para encriptar contraseñas.

**“multer”** para subir ficheros.

La base de datos es noSQL y se configura la estructura que tendrá a través del módulo **“mongoose”** de node.

Por otra parte, se ha usado JS para crear el dinamismo de las diferentes páginas.

## 8. PRUEBAS

Para comprobar el correcto funcionamiento de la aplicación, han de realizarse diferentes casos de prueba. A continuación, se muestran los diferentes casos de prueba que se han realizado para determinar que la web cumple con todas las condiciones establecidas.

### 8.1. Casos de prueba

Caso de prueba #1 – Registro. • **Descripción:** El usuario puede registrarse en el sistema introduciendo los datos que se piden por pantalla.

- **Condiciones de ejecución:** Introducir correctamente todos los datos pedidos.
- **Entrada:** Nick, contraseña, fecha de nacimiento e imagen de perfil.
- **Resultado esperado:** Si los datos introducidos no coinciden con ninguna de las condiciones que se explicarán a continuación, se registrará al usuario y se le enviará a la sección “/login”. Condiciones que mostrarán un error correspondiente:
  - Si el nick ya está en uso.
  - Si el nick tiene menos de 3 caracteres.
  - Si la contraseña tiene menos de 8 caracteres.
  - Si la fecha introducida es mayor a la fecha actual.
  - Si la fecha introducida es superior a 18 años atrás contando desde hoy.
  - Si no se introduce ninguna imagen.
- **Resultado obtenido:** Envío a la sección “/login”.

Caso de prueba #2 - Javier Borbolla – Login.

- **Descripción:** El usuario accede al sistema introduciendo su nick y contraseña.
- **Condiciones de ejecución:** Introducir correctamente el nick y la contraseña.
- **Entrada:** Nick y contraseña.
- **Resultado esperado:** Si los datos introducidos no coinciden con ninguna de las condiciones que se explicarán a continuación, se identificará al usuario y se le enviará a la página principal de la red social. Condiciones que mostrarán un error correspondiente:
  - Si el nick no existe en la base de datos.



- Si la contraseña es incorrecta.
- Si el usuario está dado de baja.
- **Resultado obtenido:** Envío a la página principal de la red social.

Caso de prueba # 3 - Javier Borbolla – Index.

- **Descripción:** Clicar en el botón correspondiente para publicar, mostrar una ventana con una caja de texto y un botón y publicar tras escribir en la caja de texto y clicar en el botón.
- **Condiciones de ejecución:** Acceder correctamente a la ventana mostrada y publicar correctamente lo escrito.
- **Entrada:** Texto.
- **Resultado esperado:** Creación correcta de la publicación.
- **Resultado obtenido:** Creación correcta de la publicación.

Caso de prueba # 4 - Javier Borbolla – Chat Global.

- **Descripción:** Acceder a la sección del chat global y enviar algún mensaje.
- **Condiciones de ejecución:** Acceder correctamente a la sección de chat global y hacer llegar a todos los clientes un mensaje.
- **Entrada:** Texto.
- **Resultado esperado:** Acceder a la sección correcta y enviar a todos los clientes un mensaje .
- **Resultado obtenido:** Correcto acceso y envío de datos.

Caso de prueba # 5 - Javier Borbolla – Seguir.

- **Descripción:** Seguir un perfil.
- **Condiciones de ejecución:** Acceder correctamente a un perfil (diferente al propio) y clicar en el botón de seguir.
- **Entrada:** Clics.
- **Resultado esperado:** Al seguir al usuario, poder ver sus ultimas publicaciones en la página principal.
- **Resultado obtenido:** Correcto funcionamiento de lo esperado.

Caso de prueba # 6 - Javier Borbolla – Mensaje directo.

- **Descripción:** Enviar un mensaje directo a alguien.
- **Condiciones de ejecución:** Ha de poderse entrar a un chat directo y poder enviar y recibir mensajes.
- **Entrada:** Texto.
- **Resultado esperado:** Envío y recibo de datos.
- **Resultado obtenido:** Se envían y reciben correctamente todos los datos.

## 9. EXPLOTACIÓN

En esta sección se planificación que tendrá el lanzamiento del proyecto en un entorno real.

### 9.1. Planificación

Planificación de servidor:

- Instalación del S.O. (ubuntu server 18.0)
- Instalación de VSFTPD
- Instalación de MongoDB
- Configuración de MongoDB.
- Instalación de Node
- Configuración de firewall.
- Transferencia de archivos al servidor.
- Despliegue de aplicación.

Testing:

- Realizar casos de prueba.

### 9.2. Plan de formación

La formación será instruida únicamente a los administradores. Esto se debe a que la parte restante de la aplicación es pública. La formación será sencilla ya que la parte de administración no contiene muchas opciones(Buscar, dar de baja y eliminar publicaciones o mensajes).

### 9.3. Pruebas de implantación

Una vez implantado el sistema, se procede a realizar las pruebas finales de implantación. Si las pruebas son satisfactorias se propondrá el sistema para su publicación.

- Se comprobará que se puede acceder por ssh y ftp al servidor.
- Se probará a acceder a la aplicación desde un ordenador y un móvil a través de internet.
- Se usarán diferentes navegadores para realizar las pruebas.
- Se creará una copia de seguridad de la BBDD y se cronometraré el proceso de restauración.
- Se intentará acceder al sistema de administradores sin haber realizado autenticación.

## **10. DEFINICIÓN DE PROCEDIMIENTOS DE CONTROL Y EVALUACIÓN**

- Debido a que solo un desarrollador se ha encargado de crear la aplicación, él mismo se ha encargado de resolver las incidencias que han ido surgiendo. Esto hace tener un mayor control sobre las incidencias y por tanto, no necesitar de una herramienta para el seguimiento de las mismas. Una vez desplegada la aplicación, se podría usar un panel público para emitir informes de errores que llegarían al desarrollador, para que, este, mejore o solucione dichos errores/características.

- Tampoco se ha usado un control de versiones. Cada cierto tiempo o cada gran cambio en la aplicación, se ha ido creando una copia de seguridad. Dichas copias se almacenan en un disco duro en posesión del desarrollador.

## 11. CONCLUSIONES

Durante el ciclo de vida de VaohLand se han realizado multitud de cambios debido al desconocimiento de ciertas técnicas de optimización y almacenamiento. Al comenzar, se desconocía la manera en la que trabajaban los sockets y la BBDD MongoDB. Tras un largo aprendizaje, se ha podido crear la aplicación tal y como se esperaba. Se han cumplido todos los objetivos en el menor tiempo posible.

Algo que cabe destacar es el desuso de tecnologías como JQuery o Bootstrap. Se eligió no usar estas tecnologías debido a que, tras varios análisis de velocidad, se comprobó que escribir en lenguajes puros(CSS o JavaScript) era más eficiente. Lo mismo sucedió para elegir Mongo, sockets y node.

## 12. FUENTES

**Enseñanzas mínimas: Real Decreto 686/2010, de 20 de mayo (BOE 12/06/2010)**

[http://pdf/IFCS03/titulo/RD20100686\\_TS\\_Desarrollo\\_Aplicaciones\\_Web.pdf](http://pdf/IFCS03/titulo/RD20100686_TS_Desarrollo_Aplicaciones_Web.pdf)

**Currículo: Decreto 1/2011, de 13 de enero (BOCM 31/01/2011)**

[http://pdf/IFCS03/curriculo/D20110001\\_TS\\_Desarrollo\\_Aplicaciones\\_Web.pdf](http://pdf/IFCS03/curriculo/D20110001_TS_Desarrollo_Aplicaciones_Web.pdf)

- **Manual de MongoDB:**<https://docs.mongodb.com/>

- **Manual de sockets:**<https://socket.io/docs/>

- **Manuales de diferentes módulos usados en Node:**<https://www.npmjs.com/>

**Diagrama de diseño:** [lucidchart.com](http://lucidchart.com)

## 13. ANEXOS

### 13.1. Manual de usuario

#### 13.1.1. Registro

Para poder acceder a la web, primero hay que registrarse. El panel de registro será igual o similar a este:

The image shows a registration form on a dark green background. It contains four input fields: 'Nick\*' with a note '( 3 caracteres min.)', 'Password\*' with a note '( 8 caracteres min.)', 'Nacimiento\*' with a note '( +18)' and a placeholder 'dd/mm/aaaa', and 'Img Perfil\*'. Below these fields is a button labeled 'Resgistrar'.

Figura 3: Panel de registro

Todos los campos son necesarios para finalizar el registro. El primero campo indica el nombre con el que te verá la gente. El segundo, la contraseña que usarás para identificarte. El tercer campo pide tu fecha de nacimiento, esto se pide para saber que eres mayor de edad. Por último, se pide una imagen de perfil (Asegúrate de elegirla bien porque, por ahora, no es reemplazable!!).

En caso de no introducir bien algún campo o querer usar un nick que ya está en uso, se mostrará un mensaje de error correspondiente.



### 13.1.2. Login

Una vez identificado, podrás hacer usar este panel. En esta sección se te pedirá el nick y la contraseña con los que te registraste.

Una vez introducido los valores, puedes pulsar “Enter” o clicar en el botón “Login”. En caso de introducir el nick o la contraseña mal, o tener la cuenta deshabilitada, se informará al usuario en un mensaje de error.

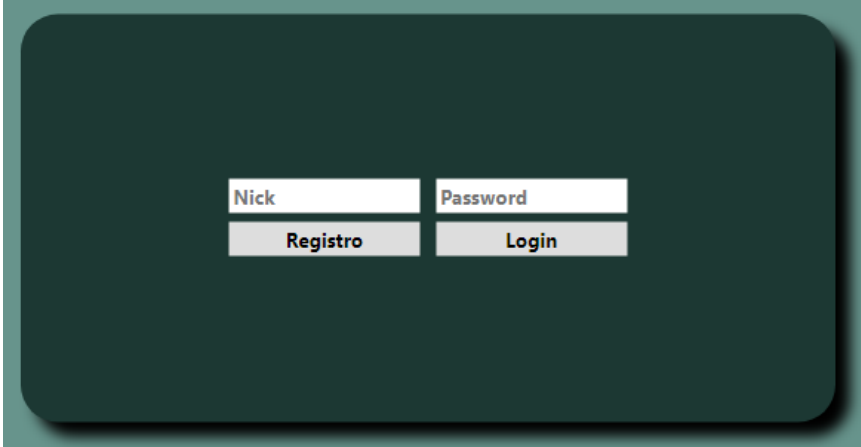
The image shows a dark green rectangular panel with rounded corners, centered on a lighter green background. Inside the panel, there are two white input fields side-by-side. The left field is labeled 'Nick' and the right field is labeled 'Password'. Below each input field is a light gray button. The button under 'Nick' is labeled 'Registro' and the button under 'Password' is labeled 'Login'.

Figura 4: Panel de identificación

En caso de no estar registrado, clicar en el botón “Registrar” para ser redireccionado al panel de registro.

### 13.1.3. Index

Una vez identificado, entrarás en la página principal de la aplicación. No te sale ninguna publicación porque aún no sigues a nadie. En la parte superior verás una barra de navegación.

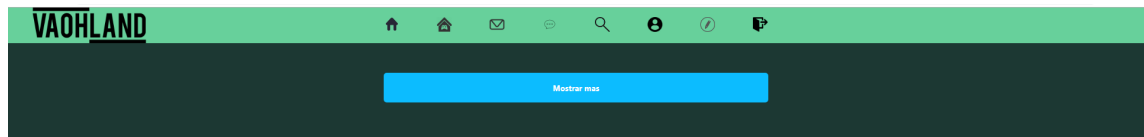


Figura 5: Página principal

### 13.1.4. Modales

#### Modal de publicaciones

Al pulsarlo, aparecerá un modal con una caja de texto y un botón.

En la caja de texto se colocará el texto que quiera ser publicado y tras ponerlo, se clicca en el botón o se pulsa “Enter”.

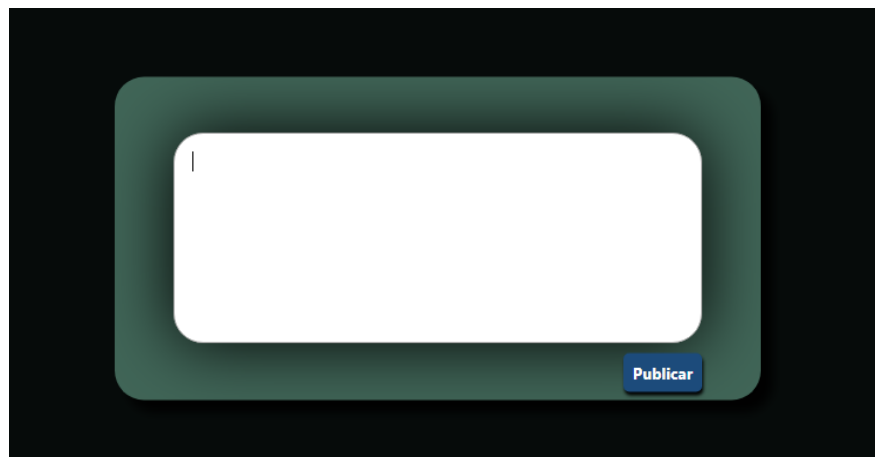


Figura 6: Modal de publicaciones

Una vez creada una publicación, aparecerá tanto en tu perfil como en la página principal.

Los botones de la publicación serán explicados en el siguiente apartado.

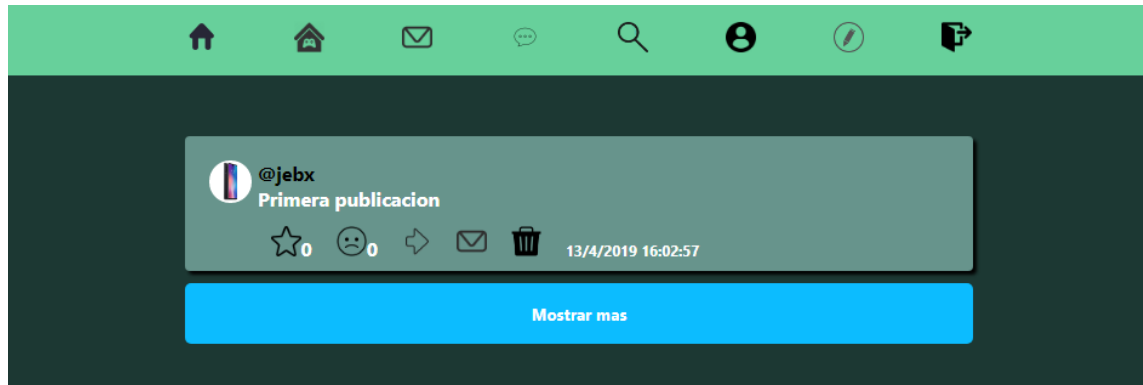


Figura 7: Publicación

### Modal de mensajes directos

En todas las publicaciones aparecerá un sobre. Al pulsarlo, aparecerá un modal con el chat que haya tenido con la persona propietaria de la publicación, una caja de texto y un botón.

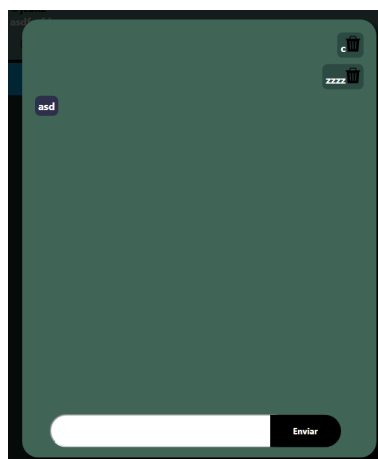


Figura 8: Modal de mensajes directos

Podrás mandarle un mensaje privado a esa persona si introduces un texto en la caja de texto y pulsas “Enter” o clicas en el botón “Enviar”.

### Modal de comentarios

En todas las publicaciones aparecerá una flecha. Al pulsarlo, aparecerá un modal con el mensaje de la publicación elegida, una caja de texto y un botón. Este modal sirve para crear comentarios. Si escribes en la caja de texto y clicas en el botón “Responder” o pulsas “Enter” se creará un comentario en dicha publicación.

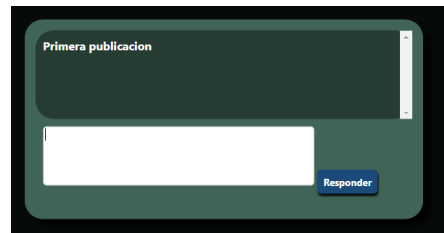


Figura 9: Modal de comentarios

Al crear el comentario, podrás verlo en tu perfil o en tu página principal.

#### 13.1.5. Botones en publicaciones

Todas las publicaciones contiene ciertas imágenes/botones que realizarán diferentes acciones según en la que cliques.

##### **Estrella**

Utilizada para dar “me gusta” a la publicación.

##### **Cara triste**

Utilizada para dar “no me gusta” a la publicación.

##### **Flecha**

Utilizada para abrir el modal de comentarios.

##### **Sobre**

Utilizado para abrir el modal de mensajes directos .

## Basura

En caso de ser propietario de una publicación, aparecerá una basura. Al clicarla, se eliminará dicha publicación.

### 13.1.6. Buscar

En la barra de navegación podrás ver una lupa. Esta es la sección para buscar usuarios o publicaciones.

En caso de querer buscar a un usuario, deberás introducir un “@” al inicio de la búsqueda.

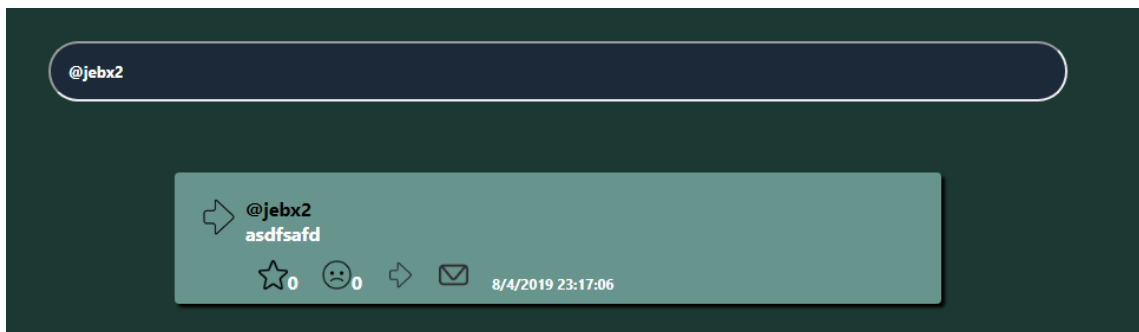


Figura 10: Buscar a un usuario específico

También está la posibilidad de buscar ciertas palabras. Para buscar una publicación con una palabra específica, escribe la palabra que deseas buscar y pulsa “Enter”.

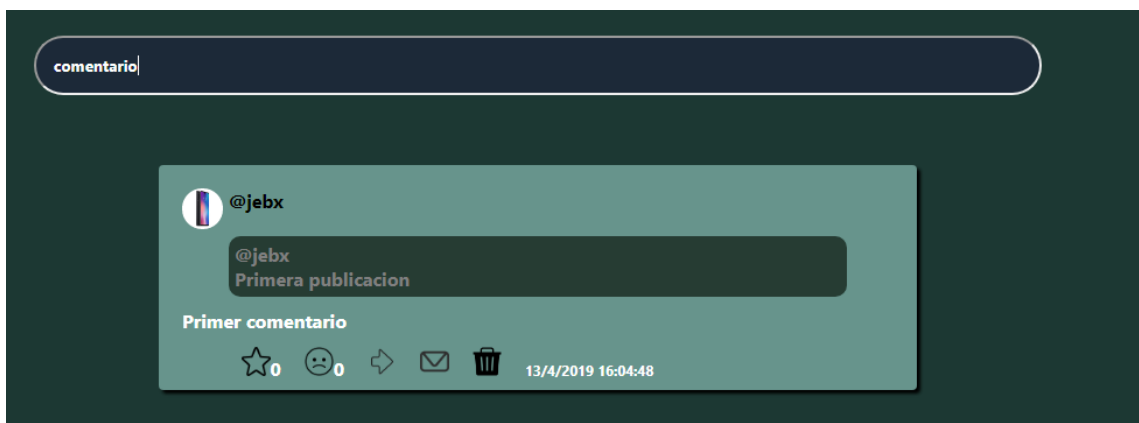


Figura 11: Buscar a un usuario específico

### 13.1.7. Perfiles

En un perfil, se mostrarán las publicaciones del usuario seleccionado. También, podrás ver su biografía, número de seguidores y seguidos y su fecha de cumpleaños.

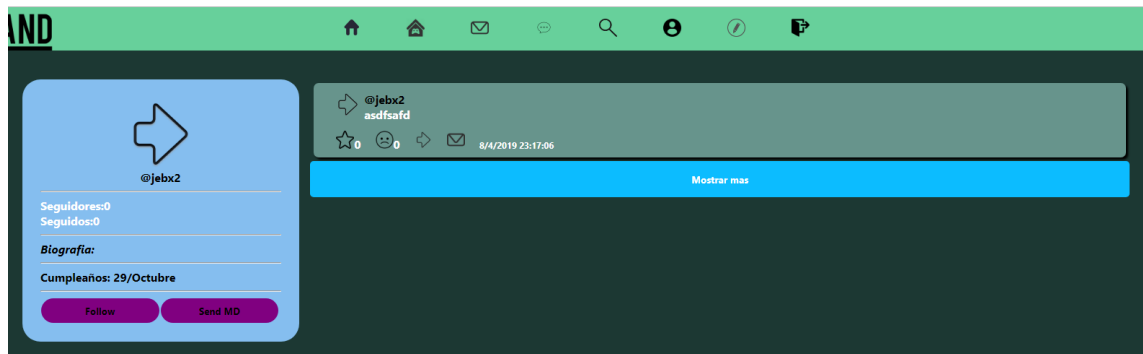


Figura 12: Perfil Usuario

Desde el perfil de un usuario, tienes la opción de seguirlo o mandarle un mensaje directo. Si lo sigues, podrás ver sus publicaciones en la página de inicio.

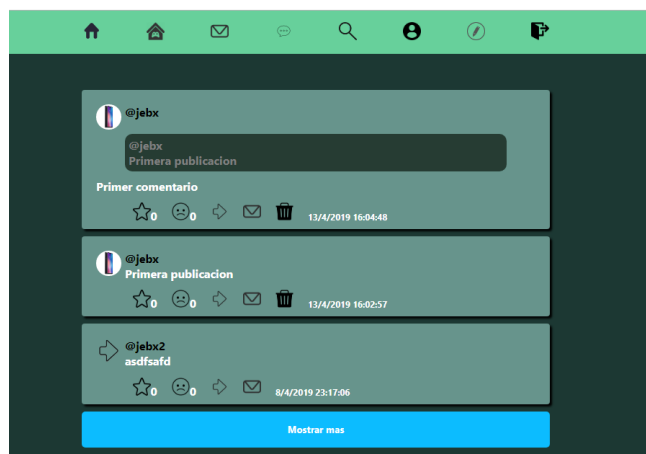


Figura 13: Perfil Usuario Anónimo

Desde la barra de navegación puedes acceder a tu propio perfil.  
En tu perfil, no podrás seguirte pero sí que podrás editar la biografía que tiene tu perfil.

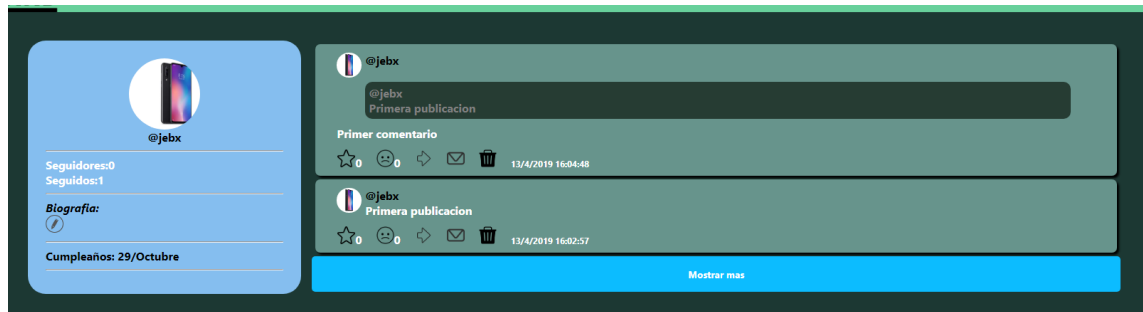


Figura 14: Perfil Propietario

### 13.1.8. Mensajes directos (MD)

Desde la barra de navegación también podrás acceder a la sección de mensajes directos. En esta sección encontrarás todos los chats que has tenido hasta el momento.

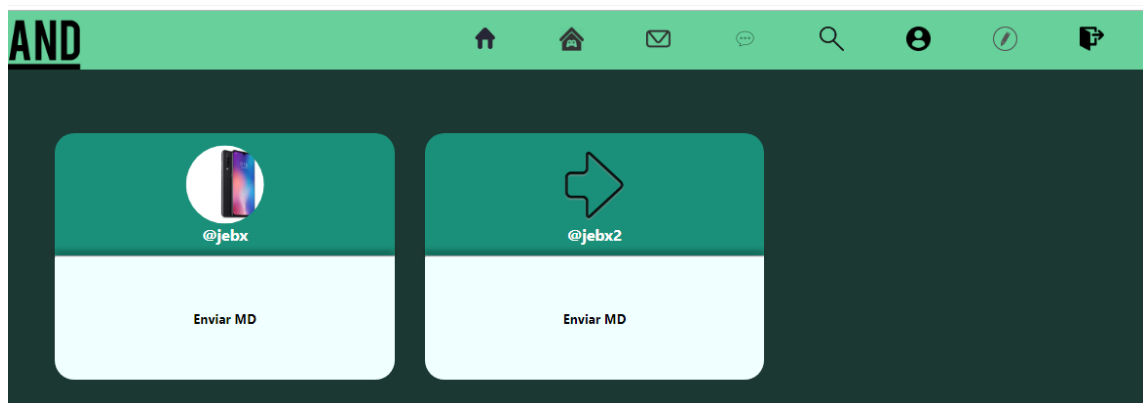


Figura 15: Mensajes Directos

Cada etiqueta está dividida en dos partes:

- La parte superior te llevará al perfil del cliente seleccionado.
- La parte inferior te abrirá el modal de mensajes directos.

### 13.1.9. Chat Global

El chat global es una sección a la que puedes acceder desde la barra de navegación. Esta sección te mostrará los 100 últimos mensajes que han sido enviados por este mismo chat.



Figura 16: Chat Global

Podrás enviar mensajes escribiendolos en la caja de texto que se encuentra en la parte inferior y pulsando “Enter” o clicando en el botón del avión de papel.



## 13.2. Manual técnico

### 13.2.1. Cliente – chatGlobal.js

Este archivo se usará únicamente en la sección del chat global.

Se definen las variables **creador** y **socket** como en la mayoría de JS de esta app.

La variable **creador** define el id de sesión del usuario. Esta cookie se define al loguearse en el sistema.

La variable **socket** almacena el socket que se va a usar durante la navegación en la app.

La variable **nickColor** es específica de este archivo. Esta variable es un array y guardará como key el nick de un usuario y como valor de dicha key el color que tendrá en el chat global.

Esta variable solo se usará en la función **showCG**.

Ahora se definen las funciones:

- **getCookie** devuelve el valor de una cookie. El parámetro que recibe es el nombre de la cookie de la que se quiere devolver el valor. Es llamada al iniciar este mismo archivo.
- **deleteMSG** envía al servidor el id del mensaje que se quiere eliminar. El parámetro que recibe es el id del mensaje que se quiere eliminar. Es llamada cuando el usuario clicca en el botón de borrar.
- **showCG** muestra en pantalla los mensajes recibidos como parámetro. Es llamada cuando el servidor envía al cliente los mensajes.
- **textSend** envía al servidor el texto que quiere ser publicado en el chat global. Es llamada cuando el usuario quiere publicar un mensaje.

Una vez definidas las funciones, pasamos a lo que se ejecutará una vez se cargue el DOM de la web.

Se crea el evento en el botón del menú desplegable de móvil. Esto cambia a visible o invisible el menú según esté abierto o cerrado.

```
//menu desplegable movil
let iconHamburguer = document.getElementById('iconHamburguer');
let menuOpen = false;
iconHamburguer.addEventListener("click", () => {
  if (menuOpen) {
    document.getElementById('navbarContent').className = 'navbarContent';
    menuOpen = false;
  } else {
    document.getElementById('navbarContent').className = 'menuMovilOpen';
    menuOpen = true;
  }
})
```

Figura 17: Evento para el botón del desplegable del menú

Se crea el evento en el botón de logout del navbar. Esto hace que se eliminen las cookies y se reenvía al login.

```
//botón logout del navbar
document.getElementById('logout').addEventListener('click', ()=>{
  document.cookie = 'nick=;expires=Thu, 01 Jan 1970 00:00:01 GMT;';
  document.cookie = 'session=;expires=Thu, 01 Jan 1970 00:00:01 GMT;';
  window.location.replace('/login');
})
```

Figura 18: Evento para el botón logout

Se comprueba si el usuario está logueado. En caso de que no, se reenvía a login.

```
//verification
if (getCookie('session')) {
  socket.emit('verifyCookie', getCookie('session'));
} else {
  window.location.replace('/login');
}
```

Figura 19: Verificación

Una vez logueado, se le pide al servidor que devuelva los mensajes del chat.

```
socket.on('verificado', function(){
  socket.emit('getCG');
});
```

Figura 20: Pedir los mensajes del chat

Se coloca el link al perfil del usuario actual en el boton correspondiente del navbar.

```
document.getElementById('urlNick').setAttribute("href", "/nick/"+getCookie('nick'));
```

Figura 21: Link al perfil propietario

Se hace scroll down al scrollbar del chat.

```
let todosMsg = document.getElementById('todosMsg');
todosMsg.scrollTop = todosMsg.scrollHeight;
```

Se crean los eventos para que al pulsar el botón de enviar o se pulse enter se llame a la función **textSend**

```
document.getElementById('send').addEventListener("click", () => {
  textSend();
})
document.addEventListener("keydown", (even) => {
  if (even.key == 'Enter') {
    textSend();
  }
})
```

Figura 22: Enviar mensaje

Por último, cuando alguien envíe un mensaje, el servidor emitará a todos los usuarios dicho mensaje. Al recibirlo se llama a **showCG**.

```
socket.on('refreshCG', function(data){
  showCG(data);
});
```

Figura 23: Mensaje recibido

### 13.2.2. Cliente – index.js

Este archivo se usará únicamente en la pagina principal de la web. Como se usará junto al fichero **library** hay algunas funciones o eventos que no están definidos.

La función **loadMore** llama al servidor para poder recibir las siguientes publicaciones. Es llamada cuando el usuario clica en el botón **cargar más**.

Una vez definida la función, pasamos a lo que se ejecutará una vez se cargue el DOM de la web.

Se crea el evento en el botón de logout del navbar. Esto hace que se eliminen las cookies y se reenvía al login.

```
//botón Logout del navbar  
document.getElementById('logout').addEventListener('click', ()=>{  
  document.cookie = 'nick=;expires=Thu, 01 Jan 1970 00:00:01 GMT;';  
  document.cookie = 'session=;expires=Thu, 01 Jan 1970 00:00:01 GMT;';  
  window.location.replace('/login');  
})
```

Figura 24: Evento para el botón logout

Una vez logueado, se le pide al servidor que devuelva los me gustas y no me gustas del usuario actual y las ultimas publicaciones de los usuarios a los que sigue.

```
//once verificated  
socket.on('verificado', function(){  
  let nick = document.location.pathname.split('/')[2];  
  socket.emit('getFavs');  
  socket.emit('getDislikes');  
  socket.emit('getTL');  
});
```

Figura 25: Una vez logueado

Se inicializan las variables de me gustas y no me gustas con valores nulos.

```
socket.on('sendMGs', function(mg){  
  //make an array full of nulls  
  //see library when updates this array  
  for (let i = 0; i < mg.length; i++) {  
    favs[mg[i]['id_post']] = null;  
  }  
});  
  
socket.on('sendDislikes', function(dis){  
  //make an array full of nulls  
  //see library when updates this array  
  for (let i = 0; i < dis.length; i++) {  
    dislikes[dis[i]['id_post']] = null;  
  }  
});
```

Figura 26: Inicializa arrays

Si el servidor envía las publicaciones, se verifica si es porque el usuario acaba de entrar o quiere cargar más publicaciones. En los dos casos, se envía el array de publicaciones a la función **showPosts**.

```
socket.on('refreshTL', function(data){
  if (data.loadMore == 'true') {
    showPosts(data.posts, 'loadMore');
  } else {
    showPosts(data);
  }
});
```

### 13.2.3. Cliente – library.js

Este archivo será usado en la página principal, en los perfiles, en la sección de búsqueda y en la sección de mensajes directos. Como se usará junto a otros ficheros hay algunas funciones o eventos que no están definidos.

En el inicio, se definen las variables. Las primeras en definirse son:

- **creator**: define el id de sesión del usuario. Esta cookie se define al loguearse en el sistema.
- **socket**: almacena el socket que se va a usar durante la navegación en la app.
- **favs**: guardará el id de los posts a los que el usuario actual le ha dado me gusta.
- **dislikes**: guardará el id de los posts a los que el usuario actual le ha dado no me gusta.
- **contador**: será usada en la función **showPosts** cuando el usuario quiera cargar publicaciones más antiguas.
- **crear**: boolean que ayudará a saber si el cliente actual ya ha enviado o recibido algún mensaje de otro cliente. Se usará cuando se habrá un chat privado .

El siguiente grupo de variables definidas son elementos del DOM que serán usados a lo largo de toda la librería.

```
//elementsDOM
let backgroundBlack;
let seePostear;
let seeChat;
let answerPost;
let textAnswer;
let sendAnswer;
let textPostear;
let sendMDElement;
let chat;
let textMD;
```

Pasamos a definir funciones:

- **getCookie** devuelve el valor de una cookie. El parámetro que recibe es el nombre de la cookie de la que se quiere devolver el valor. Es llamada al iniciar este mismo archivo.
- **showPosts** muestra por pantalla las publicaciones recibidas. El parámetro **posts** es un array de las publicaciones que mostrará, el parámetro **cargarMas** servirá para saber en que posición del parametro posts empezar a mostrar y el parámetro **porArriba** sirve para saber si colocar los posts recibidos como lastChild o como firstChild. Es llamada al recibir publicaciones o al crear una publicación.
- **showBoxAnswer** muestra el modal para crear una respuesta. Recibe como parámetro el id de la publicación a la que se quiere comentar.
- **showBoxPostear** muestra el modal para crear una publicación.
- **showChat** muestra el modal para ver un chat privado y llama al servidor para recibir los mensajes de ese chat. Recibe como parámetro el nick del usuario con el que se quiere hablar.

- **sendMD** envía el mensaje escrito a la persona seleccionada. Recibe como parámetro el nick de la persona a la que se quiere mandar el mensaje. Recibe como parámetro el nick del usuario con el que se quiere hablar.
- **sendPost** envía al servidor el texto de la publicación que se quiere crear. Es llamada cuando el usuario quiere crear una publicación
- **answer** envía al servidor la respuesta a una publicación. Recibe como parámetro el id de la publicación a la que se va a responder. Es llamada cuando el usuario quiere responder a una publicación
- **putChat** coloca en pantalla el chat privado recibido. Recibe como parámetro los mensajes del chat. Es llamada cuando se abre un chat y el servidor envía los mensajes al cliente.
- **deleteMD** elimina un mensaje privado. Recibe como parámetro el id del mensaje que se quiere eliminar. Es llamada cuando el cliente quiere borrar un mensaje.
- **deletePost** elimina una publicación. Recibe como parámetro el id de la publicación que se quiere eliminar. Es llamada cuando el cliente clica en la papelera que aparece junto a sus mensajes en un chat privado.
- **addMG** crea un 'me gusta'. Recibe como parámetro el id de la publicación a la que se quiere dar me gusta. Es llamada cuando el cliente clica en el botón de dar me gusta en una publicación.
- **deleteMG** elimina un 'me gusta'. Recibe como parámetro el id de la publicación de la que se quiere quitar el me gusta. Es llamada cuando el cliente clica en el botón de dar me gusta en una publicación.



- **addDislike** crea un 'no me gusta'. Recibe como parámetro el id de la publicación a la que se quiere dar no me gusta. Es llamada cuando el cliente clica en el botón de no me gusta en una publicación.

- **deleteDislike** elimina un 'no me gusta'. Recibe como parámetro el id de la publicación de la que se quiere quitar el no me gusta. Es llamada cuando el cliente clica en el botón de no me gusta en una publicación.

Una vez definidas las funciones, pasamos a lo que se ejecutará una vez se cargue el DOM de la web.

Se crea el evento en el botón del menú desplegable de móvil. Esto cambia a visible o invisible el menú según esté abierto o cerrado.

```
//menu desplegable movil
let iconHamburguer = document.getElementById('iconHamburguer');
let menuOpen = false;
iconHamburguer.addEventListener("click", () => {
  if (menuOpen) {
    document.getElementById('navbarContent').className = 'navbarContent';
    menuOpen = false;
  } else {
    document.getElementById('navbarContent').className = 'menuMovilOpen';
    menuOpen = true;
  }
})
```

Figura 27: Evento para el botón del desplegable del menú

Se comprueba si el usuario está logueado. En caso de que no, se reenvía a login.

```
//verification
if (getCookie('session')) {
  socket.emit('verifyCookie', getCookie('session'));
} else {
  window.location.replace('/login');
}
```

Figura 28: Verificación

Se le da valor a las variables que se inicializaron al comienzo del archivo.

```
let content = document.getElementsByClassName('content')[0];
backgroundBlack = document.getElementById('backgroundBlack');
seePostear = document.getElementById('seePostear');
seeChat = document.getElementById('seeChat');
answerPost = document.getElementById('answerPost');
textAnswer = document.getElementById('textAnswer');
sendAnswer = document.getElementById('sendAnswer');
textPostear = document.getElementById('textPostear');
sendMDElement = document.getElementById('sendMDElement');
chat = document.getElementById('chat');
textMD = document.getElementById('textMD');
```

Figura 29: Valorizando las variables

Se coloca el link al perfil del usuario actual en el botón correspondiente del navbar.

```
document.getElementById('urlNick').setAttribute("href", "/nick/"+getCookie('nick'));
```

Figura 30: Link al perfil propietario

Se oculta el background que tendrán los modales y se crea un evento para que cuando cliques en él, se oculte.

```
backgroundBlack.style.display = 'none';
backgroundBlack.addEventListener("click", (event) => {
  if (event.path[0] == backgroundBlack) backgroundBlack.style.display = 'none';
});
```

Figura 31: Background de modales

Ahora crearemos un evento en el documento para que cuando se pulse la tecla 'enter' llame a una función según el modal que se esté mostrando por pantalla.

```
document.addEventListener("keydown", (even) => {  
  if (even.key == 'Enter' && backgroundBlack.style.display == 'inherit' && seePostear.style.display == 'inherit') {  
    sendPost();  
  } else if (even.key == 'Enter' && backgroundBlack.style.display == 'inherit' && seeChat.style.display == 'inherit'){  
    sendMD(sendMDElement.value);  
  } else if (even.key == 'Enter' && backgroundBlack.style.display == 'inherit' && answerPost.style.display == 'inherit'){  
    answer(sendAnswer.value);  
  }  
})
```

Figura 32: Eventos Enter

Para acabar, se definen los sockets que están a la espera de que el servidor envíe datos.

- **sendUniquePost** se activará cuando se reciba la publicación que acaba de crear el usuario. Llamará a **showPosts** para mostrar por pantalla la nueva publicación.
- **sendUniqueComment** se activará cuando se reciba el comentario que acaba de crear el usuario. Llamará a **showPosts** para mostrar por pantalla el nuevo comentario.
- **sendOneMD** se activará cuando se reciba un chat privado completo. Llamará a **put-Chat** para mostrar por pantalla todo el chat.
- **getOwnSendedMD** se activará cuando se reciba el nuevo mensaje que acaba de crear el usuario en un chat privado. Llamará a **putChat** para mostrar por pantalla (En caso de tener el chat abierto) el nuevo mensaje.
- **getOtherMD** se activará cuando se reciba un nuevo mensaje de la persona con la que se está conversando. Mostrará por pantalla (En caso de tener el chat abierto) el nuevo mensaje

### 13.2.4. Cliente – login.js

Este archivo se usará únicamente en la sección de login.

Se define la variable **socket**.

La variable **socket** almacena el socket que se va a usar durante el login.

Ahora, se crean los siguientes eventos para que tanto clicando como pulsando 'Enter', se emita al servidor el login.

```
document.addEventListener("keydown", (even) => {
  if (even.key == 'Enter') {
    socket.emit('login', {
      nick: document.getElementById('nick').value,
      password: document.getElementById('password').value
    });
  }
})

document.getElementById('Login').addEventListener("click", (evet) => {
  socket.emit('login', {
    nick: document.getElementById('nick').value,
    password: document.getElementById('password').value
  });
});
```

Figura 33: Eventos para identificarse

En caso de recibir respuesta del servidor, dependiendo de si existe un usuario con esos credenciales o no, se crearán las cookies necesarias para la navegación por la web, o por el contrario, se mostrará un mensaje correspondiente.

```
socket.on('loginVerify', function(data){
  if (data) {
    let date = new Date();
    date.setTime(date.getTime()+(1*24*60*60*1000));
    expires = date.toGMTString();
    document.cookie = `session=${data._id} ; expires=${expires}`;
    document.cookie = `nick=${data.nick} ; expires=${expires}`;
    location.replace("/");
  } else {
    document.getElementById('error').textContent = 'Usuario o contrasena incorre
  }
});
```

Figura 34: Respuesta del servidor

En caso de estar dado de baja, se recibirá por otro socket.

```
socket.on('baja', function(data){  
    document.getElementById('error').textContent = 'Usuario dado de baja. Conta  
});
```

Figura 35: Respuesta del servidor

### 13.2.5. Cliente – md.js

Este archivo se usará únicamente en la sección de mensajes directos. Como se usará junto al fichero **library** hay algunas funciones o eventos que no están definidos. Solo hay una función en este fichero y es **showCardsMDs**. Esta función, mostrará 'cajas' con los chats privados que ya han sido empezados. Recibirá como parámetro un array con los diferentes usuarios con los que ya ha tenido una conversación.

Una vez cargado el DOM, se creará el evento para el botón de logout. Este evento eliminará las cookies existentes y redireccionará a la ventana de login.

```
//botón logout del navbar  
document.getElementById('logout').addEventListener('click', ()=>{  
    document.cookie = 'nick=;expires=Thu, 01 Jan 1970 00:00:01 GMT;';  
    document.cookie = 'session=;expires=Thu, 01 Jan 1970 00:00:01 GMT;';  
    window.location.replace('/login');  
})
```

Figura 36: Logout

Una vez el servidor verifique que el usuario está logueado, se le pedirá al servidor que devuelva los chats ya iniciados.

```
socket.on('verificado', function(){
  let nick = document.location.pathname.split('/')[2];
  socket.emit('getMDs');
});

socket.on('sendMDs', function(data){
  showCardsMDs(data);
});
```

Figura 37: Verificado

Una vez reciba la respuesta, se llamará a **showCardsMDs** para mostrarlos por pantalla.

#### 13.2.6. Cliente – nick.js

Este archivo se usará únicamente en los perfiles. Como se usará junto al fichero **library** hay algunas funciones o eventos que no están definidos.

Para empezar se definen tres variables: - **follows** indicará si el usuario actual sigue al usuario del perfil seleccionado.

- **followers** indicará la cantidad de seguidores que tiene el perfil seleccionado.

- **followsCount** indicará la cantidad de usuarios a los que sigue el perfil seleccionado.

Ahora se definen las funciones:

- **loadMore** llama al servidor para poder recibir las siguientes publicaciones. Es llamada cuando el usuario clic en el botón **cargar más**.

- **unfollow** deja de seguir al usuario del perfil seleccionado. Recibe como parámetro el nick del perfil seleccionado. Es llamada cuando el usuario clic en el botón **unfollow**.

- **unfollow** comienza a seguir al usuario del perfil seleccionado. Recibe como parámetro el nick del perfil seleccionado. Es llamada cuando el usuario clic en el botón

**follow.**

- **editBiografia** edita la biografía del perfil. Solo es posible llamarla cuando el usuario actual está en su propio perfil.
- **insertInputBio** muestra la caja que permite editar la biografía.

Una vez definidas las funciones, se define lo que ocurre cuando carga el DOM.

Se crea el evento en el botón de logout del navbar. Esto hace que se eliminen las cookies y se reenvía al login.

```
//botón logout del navbar
document.getElementById('logout').addEventListener('click', ()=>{
  document.cookie = 'nick=;expires=Thu, 01 Jan 1970 00:00:01 GMT;';
  document.cookie = 'session=;expires=Thu, 01 Jan 1970 00:00:01 GMT;';
  window.location.replace('/login');
})
```

Figura 38: Evento para el botón logout

Una vez el servidor verifique que el usuario está identificado, se le pide al servidor que devuelva:

```
socket.on('verificado', function(){
  socket.emit('getFavs');
  socket.emit('getFollow', nick);
  socket.emit('getDislikes');
  socket.emit('getTLNick', {nick:nick});
  socket.emit('getFollowers', nick);
  socket.emit('getFollows', nick);
  socket.emit('getPerfil', nick);
});
```

Figura 39: Una vez verificado

- Los 'me gustas' que ha dado el usuario actual.
- Si el usuario actual sigue al perfil seleccionado.
- Los 'no me gustas' que ha dado el usuario actual.

- Las publicaciones del perfil seleccionado.
- La cantidad de seguidores que tiene el perfil seleccionado.
- La cantidad de seguidos que tiene el perfil seleccionado.
- Información como su cumpleaños, biografía e imagen de perfil.

Cada una de estas peticiones tendrá su propio socket a la espera. Cada vez que reciba alguno de los datos pedidos, se procesarán en los siguientes sockets.

```
socket.on('sendMGs', function(mg){=});
socket.on('sendDislikes', function(dis){=});
socket.on('sendFollow', function(fl){=});
socket.on('sendFollowers', function(followes){=});
socket.on('sendFollows', function(followes){=});
socket.on('sendTLNick', function(data){=});
socket.on('sendPerfil', function(data){=});
```

Figura 40: Sockets esperando al servidor

### 13.2.7. Cliente – register.js

Este archivo se usará únicamente en la ventana de registro.

Este fichero solo funcionará cuando cargue el DOM.

Lo primero que hará será mirar en la URL si hay algún mensaje de error. Este mensaje aparece si ha ocurrido algo mal al registrarse. Dependiendo del error recibido, se mostrará un mensaje u otro.

Después de verificar si ha habido errores, se crea un evento 'click' en el submit del formulario. Este evento bloqueará el submit del formulario y verificará que todos los datos introducidos son válidos. En caso de haber un inválido, se muestra un mensaje correspondiente. En caso de no haber errores, el formulario continuará.



### 13.2.8. Cliente – search.js

Este archivo se usará únicamente en la sección de búsqueda.

Este fichero solo funcionará cuando cargue el DOM.

Lo primero que hará será crear el evento en el botón de logout del navbar. Esto hace que se eliminen las cookies y se reenvía al login.

```
//botón logout del navbar
document.getElementById('logout').addEventListener('click', ()=>{
  document.cookie = 'nick=;expires=Thu, 01 Jan 1970 00:00:01 GMT;';
  document.cookie = 'session=;expires=Thu, 01 Jan 1970 00:00:01 GMT;';
  window.location.replace('/login');
})
```

Figura 41: Evento para el botón logout

Después, se crea un evento para que al pulsar 'Enter' se mande al servidor lo que el usuario quiere buscar. En caso de que el texto que se quiera buscar comience por '@' se buscarán publicaciones con el nick de usuario escrito. En caso de no tener '@' se buscarán publicaciones con el texto escrito.

```
let inputSearch = document.getElementById('inputSearch');
inputSearch.addEventListener("keydown", (even) => {
  if (even.key == 'Enter') {
    document.getElementsByClassName('content')[0].textContent = '';
    if (inputSearch.value.trim()[0] == '@'){
      socket.emit('searchByNick', inputSearch.value.trim().substring(1));
    } else {
      socket.emit('searchByText', inputSearch.value.trim());
    }
  }
})
```

Figura 42: Eventos 'Enter'

Cuando reciba las publicaciones, se mostrará por pantalla.

```
socket.on('returnSearch', function(data){  
    showPosts(data);  
});
```

Figura 43: Eventos 'Enter'

Lo primero que hará será mirar en la URL si hay algun mensaje de error. Este mensaje aparece si ha ocurrido algo mal al registrarse. Dependiendo del error recibido, se mostrará un mensaje u otro.

Después de verificar si ha habido errores, se crea un evento 'click' en el submit del formulario. Este evento bloqueará el submit del formulario y verificará que todos los datos introducidos son validos. En caso de haber un invalido, se muestra un mensaje correspondiente. En caso de no haber errores, el formulario continuará.

### 13.2.9. Servidor – scriptNode.js

Es el único fichero ejecutado en el servidor.

Primero se definirán los módulos, después la estructura de la BBDD, más abajo el enrutamiento y para terminar los sockets y una función.

Módulos:

- **Express**, usado para el enrutamiento y permitir el uso de los sockets.

**Sockets**, conectan al cliente con el servidor.

- **Multer y dependencias**, usado para la subida de imágenes.

- **Bcrypt**, usado para encriptar las contraseñas.

- **Mongoose**, facilita la conexión y el uso de la MongoDB.

```
//express
const express = require('express');
const app = express();

//sockets
const http = require("http");
const server = http.Server(app);
const io = require('socket.io')(server);

//upload imgs
const readChunk = require('read-chunk');
const fileType = require('file-type');
const multer = require('multer');
const upload = multer({ dest: 'uploads/' });
const cpUpload = upload.fields([ { name: 'avatar', maxCount: 1 }, { name: 'password', maxCount: 1 }]);

//encrypt pass
const bcrypt = require('bcrypt');
const saltRounds = 10;

//bd
const mongoose = require('mongoose');
mongoose.set('useCreateIndex', true);
mongoose.connect("mongodb://localhost:27017/redSocial?authSource=admin&gssapiServiceName=redSocial");
```

Figura 44: Módulos

A continuación, se define la estructura de la BBDD. El nombre usado para las constantes es igual al nombre de los modelos creados.

```
const Schema = mongoose.Schema;
const ObjectId = Schema.ObjectId;
const Follows = mongoose.model('Follows', new Schema ({=
},{versionKey: false}));
const Clientes = mongoose.model('Clientes', new Schema ({=
},{versionKey: false}));
const Posts = mongoose.model('Posts', new Schema ({=
},{versionKey: false}));
const Likes = mongoose.model('Likes', new Schema ({=
},{versionKey: false}));
const Dislikes = mongoose.model('Dislikes', new Schema ({=
},{versionKey: false}));
const Chat_Directo = mongoose.model('Chat_Directo', new Schema ({=
},{versionKey: false}));
const Chat_Global = mongoose.model('Chat_Global', new Schema ({=
},{versionKey: false}));
const Admin_Users = mongoose.model('Admin_Users', new Schema ({=
},{versionKey: false}));
```

Figura 45: Estructura de la BBDD.

Pasamos a la parte de enrutamiento. Primero daremos acceso a las carpetas donde se almacena el css, js y html.

```
//access
app.use(express.static('css'));
app.use(express.static('js'));
app.use(express.static(__dirname));
```

Figura 46: Habilitando acceso a ciertas carpetas.

Las siguientes líneas se basan en el enrutamiento

```
app.get('/', function(req, res) {
  res.sendFile(__dirname + "/index.html");
});

app.get('/login', function(req, res) {
  res.sendFile(__dirname + "/login.html");
});

app.get('/login?', function(req, res) {
  res.sendFile(__dirname + "/login.html");
});

app.get('/register', function(req, res) {
  res.sendFile(__dirname + "/register.html");
});

app.get('/nick/*', function(req, res) {
  res.sendFile(__dirname + "/nick.html");
});

app.get('/MD', function(req, res) {
  res.sendFile(__dirname + "/MD.html");
});

app.get('/chatGlobal', function(req, res) {
  res.sendFile(__dirname + "/chatGlobal.html");
});

app.get('/search', function(req, res) {
  res.sendFile(__dirname + "/search.html");
});

app.get('/loginAdmin', function(req, res) {
  res.sendFile(__dirname + "/loginAdmin.html");
});

app.get('/indexAdmin', function(req, res) {
  res.sendFile(__dirname + "/indexAdmin.html");
});

app.get('/chatGlobalAdmin', function(req, res) {
  res.sendFile(__dirname + "/chatGlobalAdmin.html");
});

app.get('/seePerfil/*', function(req, res) {
  res.sendFile(__dirname + "/seePerfil.html");
});

app.get('/register', function(req, res) {
  res.sendFile(__dirname + "/register.html");
});

app.get('/register/*', function(req, res) {
  res.sendFile(__dirname + "/register.html");
});
```

Figura 47: Enrutamiento de la web.

El último enrutamiento es más largo ya que es la llamada que se hace en el registro de usuarios y se ha de comprobar que todos los datos introducidos son correctos. Además se tiene que guardar la imagen subida. Esta llamada es la única que se realiza por POST.

```
app.post('/register', cpUpload, function(req, res) {
  if (req.body['nick'].trim().length > 20 || req.body['nick'].trim().length < 3) {
    res.redirect("/register?errorNickLength");
  } else if (req.body['password'].trim().length < 8) {
    res.redirect("/register?errorPassword");
  } else {
    let buffer = readChunk.sync(req.files['image'][0]['path'], 0, fileType.minimumBy
    if (fileType(buffer)['mime'].indexOf("image") == 0) {
      Clientes.findOne({ nick: req.body['nick']}, function (err, client) {
        if (err) return err;
        if (client == null && req.body['nick'] != 'admin') {
          bcrypt.hash(req.body['password'], saltRounds, function(err, hash) {
            Clientes.create({ nick: req.body['nick'], password: hash, img_perfil: req
              if (err) return err;
              res.redirect('/login');
            });
          });
        } else {
          console.log('a');
          res.redirect("/register?errorNick");
        }
      });
    } else {
      res.redirect("/register?errorImage");
    }
  }
});
```

Figura 48: Enrutamiento de registro.

Antes de comenzar con los sockets, se definirán dos variables que serán usadas en

estos.

La primera es **users**. Este array almacenará los sockets que se estén usando. Junto al socket, se guardará el nick para un facilitar el encuentro de dicho socket.

La segunda variable definida es un array con los meses del año. Solo se usa en el apartado de perfiles.

Comencemos con los sockets:

- **disconnect**: Activado cuando un socket se desconecta. Elimina dicho socket del array 'users'.
- **loginAdmin**: Comprueba que el usuario y la contraseña recibidos se encuentran en la tabla de usuarios administrativos. Devuelve el id del cliente o null dependiendo del resultado encontrado.
- **login**: Comprueba que el usuario y contraseña recibidos se encuentran en la tabla de usuarios clientes. Devuelve el id y nick del cliente si se ha encontrado.
- **verifyCookieAdmin**: Verifica que el id recibido se encuentra en la tabla de usuarios administrativos. En caso de existir, almacena el socket en la variable 'users' y confirma al cliente que es válido.
- **verifyCookie**: Verifica que el id recibido se encuentra en la tabla de usuarios clientes. En caso de existir, almacena el socket en la variable 'users' y confirma al cliente que es válido.
- **getTL**: Devuelve las publicaciones de los perfiles a los que sigue el usuario actual.
- **savePost**: Guarda el post recibido en la tabla Posts y envía su id.
- **deletePostAdmin**: Elimina una publicación siendo administrador junto a sus 'me gusta' y 'no me gusta'. Es diferente a eliminar un post normal porque en este, se verifica que realmente eres administrador.
- **deletePost**: Elimina una publicación siendo cliente junto a sus 'me gusta' y 'no me gusta'.
- **getFavs**: Devuelve los 'me gusta' del usuario actual.
- **getDislikes**: Devuelve los 'no me gusta' del usuario actual.
- **addMG**: Añade un 'me gusta' a la publicación recibida como parámetro.
- **deleteMG**: Elimina un 'me gusta' a la publicación recibida como parámetro.
- **addDislike**: Añade un 'no me gusta' a la publicación recibida como parámetro.
- **deleteDislike**: Elimina un 'no me gusta' a la publicación recibida como parámetro.

- **getCG**: Devuelve los últimos 100 mensajes del chat global.
- **sendMsg**: Crea el mensaje recibido como parámetro en el chat global.
- **deleteMSG**: Elimina el mensaje recibido como parámetro del chat global.
- **getTLNick**: Devuelve las publicaciones del perfil pasado como parámetro.
- **getFollowers**: Devuelve la cantidad de 'seguidores' que tiene el perfil pasado como parámetro.
- **getFollows**: Devuelve la cantidad de 'seguidos' que tiene el perfil pasado como parámetro.
- **getPerfil**: Devuelve información sobre el perfil pasado como parámetro. Información como la fecha de nacimiento, la biografía e imagen de perfil.
- **editBiografia**: Cambia la biografía del usuario actual por el texto recibido como parámetro.
- **follow**: 'Sigue' al perfil pasado como parámetro.
- **unfollow**: 'Deja de seguir' al perfil pasado como parámetro.
- **getMDs**: Devuelve los nicks de los perfiles con los que se ha comenzado un chat privado.
- **getOneMD**: Devuelve el chat privado con el perfil pasado como parámetro.
- **addMD**: Crea el mensaje pasado como parámetro en el chat privado correspondiente.
- **deleteMD**: Elimina el mensaje privado especificado como parámetro.
- **addComment**: Crea un comentario en la publicación pasada como parámetro.
- **searchByNickAdmin**: Busca nombres de perfiles que coincidan con el string pasado como parámetro.
- **searchByNick**: Busca publicaciones creadas por el nombre de perfil que es pasado como parámetro.
- **searchByText**: Busca publicaciones que contengan el string pasado como parámetro.
- **unsubscribeAccount**: Da de baja una cuenta.
- **suscribeAccount**: Da de alta una cuenta ya existente.
- **deleteMSGAdmin**: Permite a los administradores borrar un mensaje del chat global.
- **sendMsgAdmin**: Envía el mensaje especificado por parámetro al chat global identificando al usuario como 'Admin'.

Una vez definidos los sockets, terminamos el documento definiendo la función **returnCG**. Esta función, dependiendo de si recibe un socket específico o no, envía el chat global a todo el mundo o solo a un cliente.

```
function returnCG(socket = null){  
  Chat_Global.find().sort({_id:-1}).limit(100).exec(function (err, msgs) {  
    if (err) return handleError(err);  
    try{  
      socket == null ? io.sockets.emit('refreshCG', msgs) : users[socket.id]['socket'].emit('re  
    } catch (e) {  
      console.log('no ha pasado nada');  
      console.log(e);  
    }  
  });  
}
```

Figura 49: Enrutamiento de registro.

Las dos últimas líneas indican en que puerto están escuchando los sockets y en cual la aplicación.

```
app.listen(80);  
server.listen(3000);
```

Figura 50: Enrutamiento de registro.