



FILLMEIN

Red social de diferentes temas

[Descripción breve](#)

[Dibujar su lector con un resumen de la participación. Normalmente es un breve resumen del documento.]

Cuando esté listo para agregar contenido, haga clic aquí y empiece a escribir.]

Cecilia Llamazares López

i.es San Andrés del Rabanedo, Desarrollo de aplicaciones web

Tabla de contenido

1. Estudio del problema y análisis del sistema	2
1.1 Introducción.....	2
1.2 Finalidad.....	2
1.3 Requisitos.....	2
2. Recursos	3
2.1 Recursos hardware	3
2.1.1 Recurso para el desarrollo: Portátil 1GB RAM y 1GB de espacio mínimo	3
2.1.2 Servicio de hosting.....	3
2.2 Recursos software	3
2.2.1 Visual Studio Code	3
2.2.2 Postman	3
3. Planificación.....	5
3.1 Planificación temporal.....	5
3.2 Planificación económica	5
4. Desarrollo y pruebas	5
4.1 Diseño	5
4.2 Implementación.....	8
4.2.1 API	8
4.2.2 Inicio de sesión y registro	10
4.2.3 Perfil.....	11
4.2.4 Página de inicio.....	13
4.2.5 Perfil de otros usuarios.	15
5. Conclusiones finales	15
5.1 Grado de cumplimiento de los requisitos fijados.....	15
5.2 Propuestas de mejora o ampliaciones futuras	16
6. Guías.....	16
6.1 Guía de uso	16
6.2 Guía de instalación	20
7. Referencias bibliográficas	21
8. URL del proyecto	21

1. Estudio del problema y análisis del sistema

1.1 Introducción

Las Redes Sociales están ganando bastante popularidad en los últimos años y todas y cada una de ellas están destinadas propósitos en concreto, ya sea para buscar información de utilidad o para conectar con otras personas.

Fill Me In es una red social en la que los usuarios podrán publicar entradas de cualquier cosa como en cualquier otra red social. Esta página web también cuenta con cuentas profesionales, que cualquiera las podrá crear. A lo que se dedican estas cuentas es a publicar información de un tema en específico.

Con esto dicho, los usuarios podrán buscar entradas ya sea por cuentas personales o profesionales, si les apetece buscar algo de información y no contactar con otras personas.

1.2 Finalidad

Este proyecto pretende ofrecer un aspecto informativo en cuanto a los diferentes usuarios de la aplicación y del mundo actual en caso de que una cuenta esté especializada en compartir información de este tipo.

1.3 Requisitos

Los requisitos de la aplicación son los siguientes:

- **Cuentas profesionales:** Cuentas que se dedican única y exclusivamente a publicar contenido relacionado con el tema que se eligió a la hora de su creación. Ej: Agricultura.
- **Muro:** Cada cuenta tiene una zona llamada “muro”, donde la gente podrá poner varios comentarios de esta, puede ser a modo de sugerencia.
- **Valoraciones:** Las cuentas profesionales tienen un sistema de valoración en el que los usuarios podrán decidir si es una buena cuenta que cumple con su función o no, dependiendo de sus contenidos.
Las entradas de todos los usuarios también se podrán valorar, especificando mediante un botón si al usuario le ha gustado la publicación.
- **Suscripción a temas:** Los usuarios pueden suscribirse a diversos temas de interés y podrán buscar entradas acordes a estos temas elegidos, ya sea por novedades o por mejores valorados.
- **Filtros de búsqueda:** Los usuarios pueden buscar entradas por cuentas normales, cuentas profesionales, cuentas a las que siguen o por temas. También pueden buscar entradas de un tema en específico mejores valoradas, así como las últimas entradas de estos. También pueden buscar entradas de usuarios en concreto.

2. Recursos

2.1 Recursos hardware

2.1.1 Recurso para el desarrollo: Portátil 1GB RAM y 1GB de espacio mínimo

El portátil, ordenador o dispositivo debe cumplir con los requisitos mínimos de 1GB de RAM y 1GB de espacio de almacenamiento disponible para poder ejecutar Visual Studio Code, la aplicación donde se desarrolla todo el código.

También debe tener acceso a internet para poder ejecutar la aplicación.

2.1.2 Servicio de hosting

En cuanto al servicio hosting, se podrá utilizar cualquiera público gratuito para poder alojar la aplicación.

En este proyecto utilicé [Alwaysdata.com](https://alwaysdata.com), un servicio hosting que ofrece un plan gratuito con 100 MB de almacenamiento. Utilicé el plan de 9 € al mes ya que ofrecía 10 GB de almacenamiento.

2.2 Recursos software

2.2.1 Visual Studio Code

El software que utilizo principalmente para el proyecto es Visual Studio Code. Es un editor de código fuente que soporta de muchos lenguajes de programación, como Java, JavaScript, PHP, Python, etc.

Ventajas:

- Es un editor gratuito de código abierto.
- Soporta muchos lenguajes de programación.
- Es compatible con varias plataformas (Windows, macOS y Linux)
- Tiene muchas opciones de personalización (Extensiones y apariencia)

Desventajas:

- Interfaz de usuario confusa para nuevos usuarios.
- Bajo rendimiento en proyectos grandes.
- Requiere conocimientos técnicos para la personalización.

2.2.2 Postman

Utilizo esta aplicación para mandar peticiones al servidor de api que estoy utilizando y ver las respuestas que me envía.

Me resulta una aplicación muy sencilla de utilizar para ver todas aquellas respuestas que recibas.

Estas respuestas pueden variar de formato, si el servicio lo permite. Como el que yo estoy utilizando es parte HTML y parte JSON, lo utilizo mayormente para ver esa parte en JSON y poder leerlo más tarde. (Ver en la **Figura 4**)

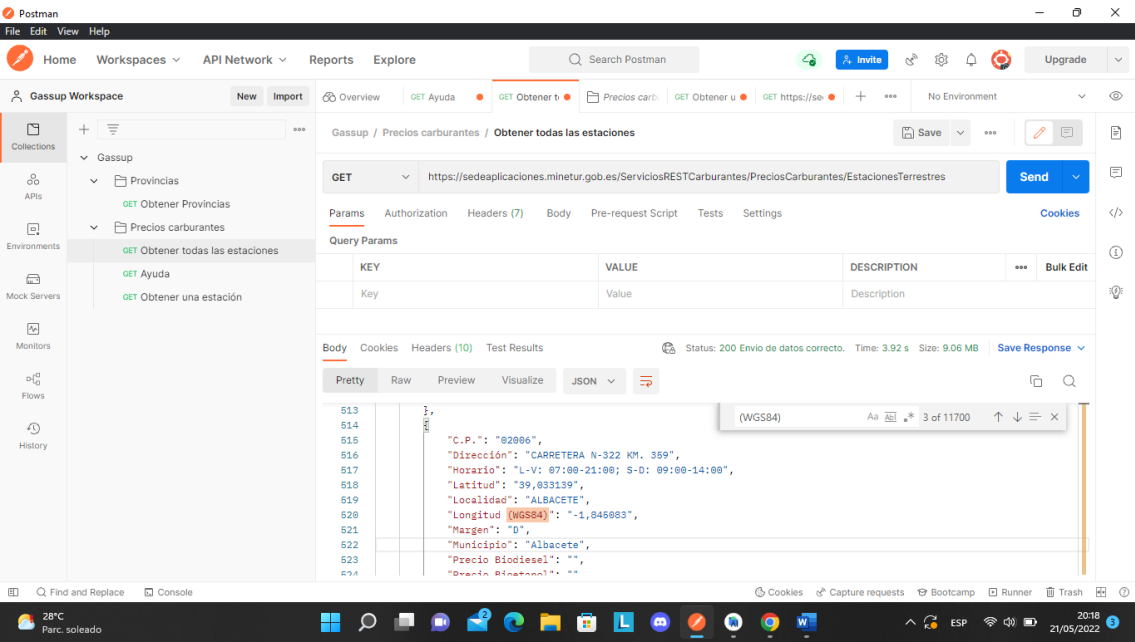


Figura 2.

La **Figura 2** es un ejemplo del enlace que estoy utilizando, y cómo se visualiza la respuesta en la **Figura 5** de manera organizada, así como las diferentes solicitudes en archivos de la **Figura 3**.

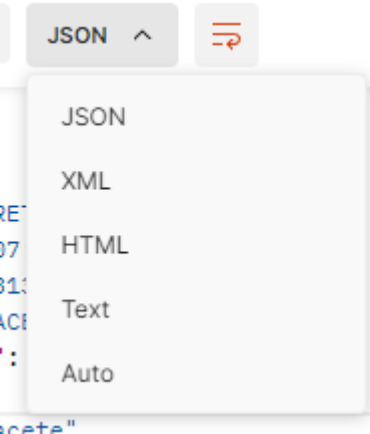


Figura 3.

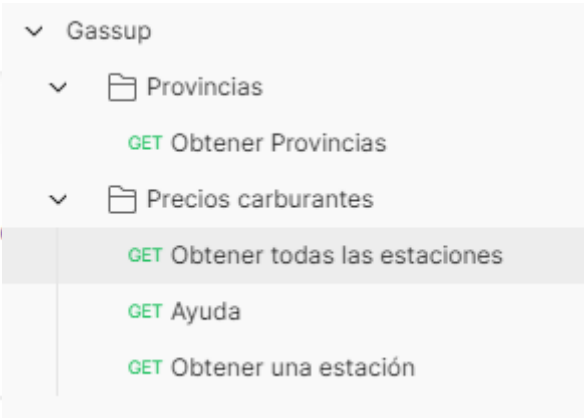


Figura 4.

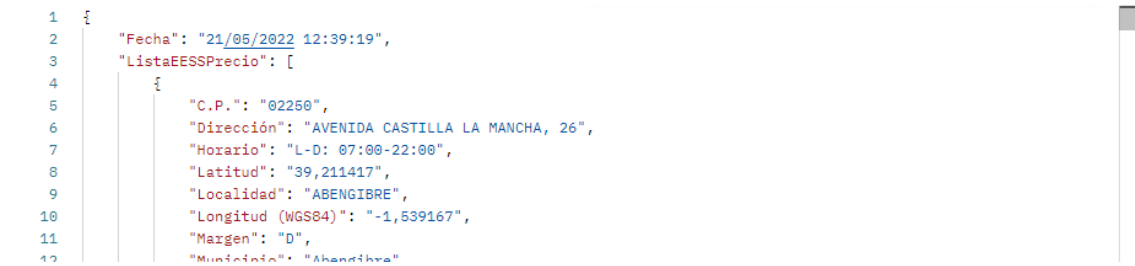


Figura 5.

3. Planificación

3.1 Planificación temporal

Esta planificación está dividida en diferentes ideas del proyecto.

Marzo	1ª semana	2ª semana	3ª semana	4ª semana
Documentación servidor para red social	5 h			
Documentación sobre JWT		10 h		
Diseño de interfaz, preparar CSS				20 h
Pantalla principal				15 h
Abril	1ª semana	2ª semana	3ª semana	4ª semana
Inicio y registro de usuarios		15 h		
Implementación de los servicios en PHP	4 h			
Creación y subida de entradas HTML				18 h
Edición del perfil			8 h	
Mayo	1ª semana	2ª semana	3ª semana	4ª semana
Seguimiento de usuarios		16 h		
Guardar entradas			26 h	
Valorar entradas			8 h	
Filtro de cuentas y entradas				25 h

3.2 Planificación económica

El presupuesto económico para desarrollar se basa en el precio de los dispositivos mencionados anteriormente.

Recurso	Precio
Licencia Windows 10	11,90 €
Lenovo IdeaPad 330 i5	369 €
Software	9 €
Total	389,90 €

4. Desarrollo y pruebas

4.1 Diseño

La base de datos de la página web consta de tres tablas: Usuarios, muro y entradas. Se va a centrar principalmente en la de usuarios, ya que de esta se puede sacar todo lo demás.

No he creado claves foráneas puesto que lo vi más sencillo haciendo simplemente referencia al nombre del usuario en las otras dos tablas.

En la **Figura 7** se pueden ver los diferentes campos de cada una de las tablas.

fillme in usuarios	fillme in entradas	fillme in muro
id : int(11)	id : int(11)	id : int(11)
username : varchar(20)	username : varchar(25)	user_post : varchar(25)
name : varchar(50)	content : varchar(1000)	user_profile : varchar(25)
pass : varchar(1000)	date : timestamp	content : varchar(200)
email : varchar(50)		date : timestamp
biography : varchar(200)		
birthday : date		
followers : mediumtext		
following : mediumtext		
professional : int(11)		
type : varchar(30)		
image : varchar(1000)		
banner : varchar(1000)		
saved_entries : mediumtext		

Figura 6.

La paleta de colores de la página web se basa en tonos morados, grises, blancos y amarillo. Predomina el color morado en todas las secciones de esta, por lo que será el color representativo de la aplicación.



Figura 7.

En cuanto al icono de la aplicación, será algo muy sencillo, lo suficiente como para poder representar a la página web de manera adecuada con su tema de colores.

Es un cuadrado con la letra F en morado situada en el centro.



Figura 8.

En cuanto a los mockups, en las siguientes imágenes se puede ver el diseño principal de algunas de las pantallas:

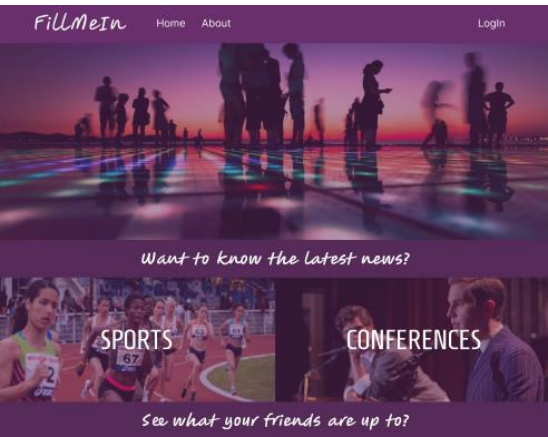


Figura 9.

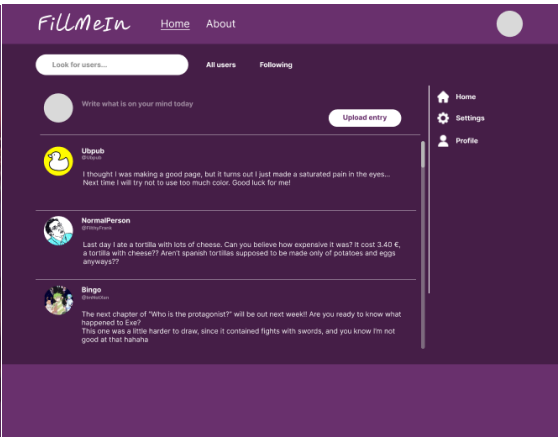


Figura 10.

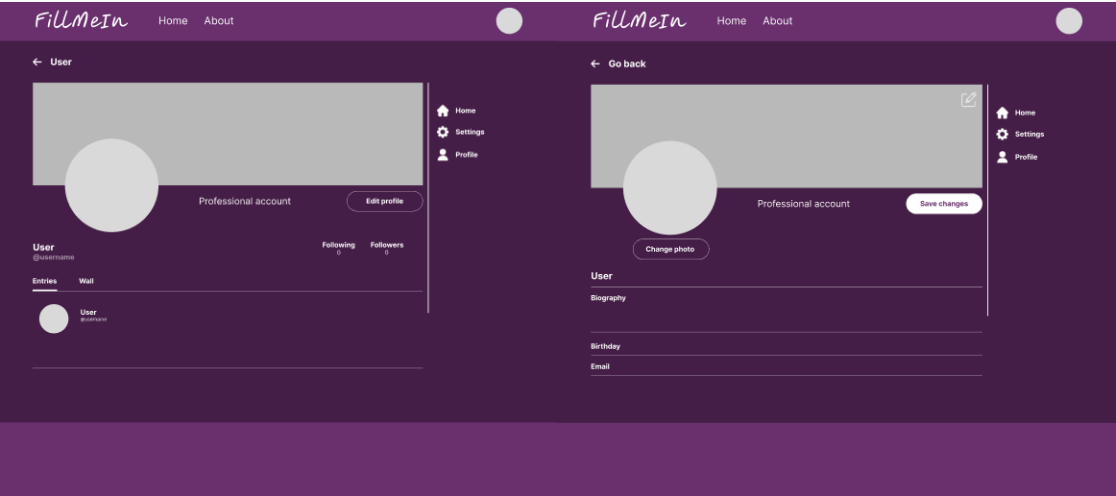


Figura 11.

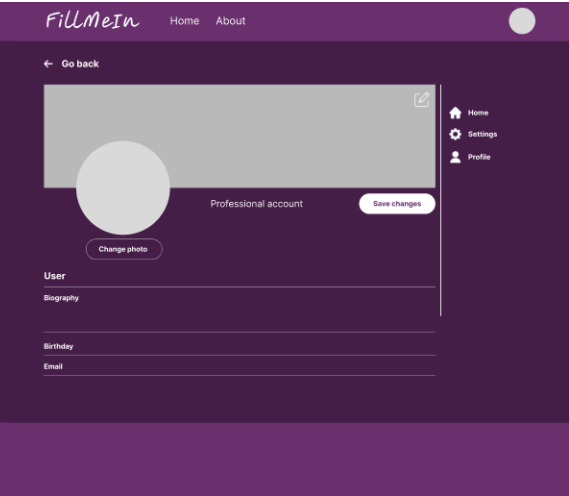


Figura 12.

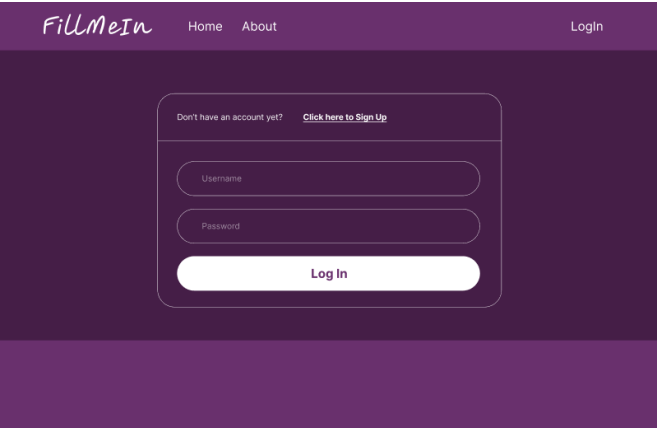


Figura 13.

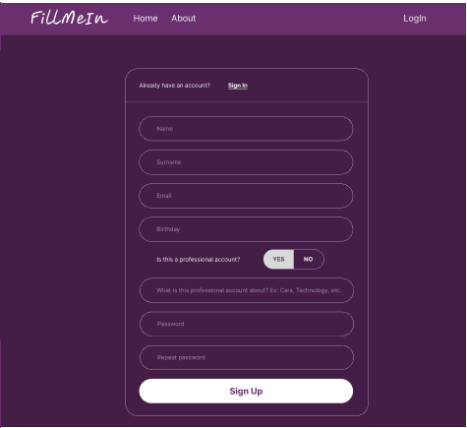


Figura 14.

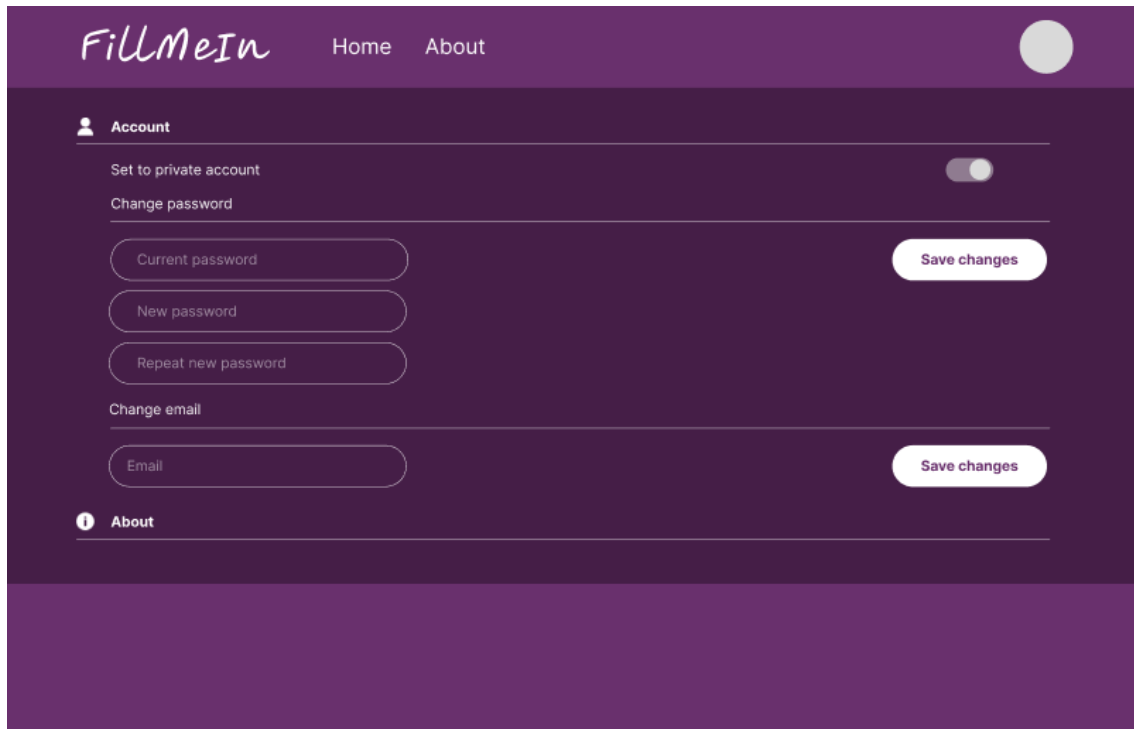


Figura 15.

4.2 Implementación

4.2.1 API

Creación de diferentes Apis en la parte de servidor para poder pasar los datos de la base de datos a la parte del cliente.

Las Apis son las siguientes:

- **Conexión:** Donde se inicia la conexión a la base de datos. Esta clase es la que se utilizará en las demás Apis para hacer consultas.
- **Usuarios:** Compuesto por dos comprobaciones: GET y POST. En la petición GET comprueba si se le ha pasado el nombre de usuario, la contraseña o el id en la URL. También genera un JWT para mantener iniciada la sesión del usuario. Esto se consigue con una “key” y un “Payload”:

// Creación de una clave que se utilizará para generar un webToken
\$key = 'example_key';

Figura 16.

```
// Genera un payload con el nombre de usuario (para el webToken)
$payload = [
  'iss' => $users[0]['username'],
];

$jwt = JWT::encode($payload, $key, 'HS256');

$users[0]['webToken'] = $jwt;
```

Figura 17.

En la **Figura 17** se genera una variable “payload” en la que agrega el nombre de usuario como valor ‘iss’. Seguido, crea un JWT con ayuda del JWT::encode, el cual cifra el contenido, en este caso el “payload”, “key” y el tipo de cifrado HS256.

Seguido, se añade este JWT al usuario obtenido en un nuevo campo llamado *'webToken'*.

En cuanto al POST, recibe unos parámetros del cuerpo de la petición de la parte JavaScript, comprueba que en la base de datos no exista un registro con el mismo nombre de usuario e inserta los datos.

- **Filtro de usuarios:** En el método GET busca a los usuarios cuyos nombres de usuario o tipo de cuenta profesional coincidan o tengan los caracteres que se han pasado en la URL.

```
if (isset($_GET['professional'])) {
    $professional = $_GET['professional'];
    if ($professional == 'personal') {
        $sql .= " AND professional = '0'";
    } else {
        $sql .= " AND professional = '1'";
    }
}
```

Figura 18.

En la **Figura 18** hace una comprobación, si ha recibido otro parámetro que se llame *"professional"* comprueba si su valor es *"personal"* o *"profesional"*. En cada caso añade a la consulta en la parte WHERE una comprobación para la columna *"professional"*. El 0 significa falso y el 1 verdadero. Esto se explicará más adelante.

El POST de este API se utiliza para editar los seguidores, las personas a las que sigue el usuario conectado y sus entradas guardadas. Llegan unos valores en el cuerpo y, mediante esos valores, realiza las siguientes comprobaciones:

```
// Sentencia para actualizar los campos editados del usuario
$sql = "UPDATE usuarios SET username = '{$_user->username}'";
if ($_user->followers != null) {
    $sql .= ", followers = '{$_user->followers}'";
}
if ($_user->following != null) {
    $sql .= ", following = '{$_user->following}'";
}
if ($_user->saved_entries != null) {
    $sql .= ", saved_entries = '{$_user->saved_entries}'";
}
$sql .= " WHERE username = '{$_user->username}'";
```

Figura 19.

- **Editar usuario:** Consta de dos comprobaciones: POST y DELETE. La parte POST actualiza el usuario con los datos modificados enviados en el cuerpo de la petición.

La parte DELETE elimina al usuario, recogiendo el id y la contraseña enviados en el cuerpo.

- **Entradas:** Las entradas están compuestas también por comprobaciones GET y POST como todas las demás. En el GET obtiene las entradas con el id o nombre de usuario que se ha pasado. En la sentencia lo ordena por *"date"*, que es la fecha de la publicación de las entradas, de manera descendente, para que las entradas más recientes aparezcan en la parte superior.

En el POST crea una variable donde almacena la fecha local actual.

```
$fecha_actual = date('Y-m-s H:i:s');
```

Figura 20.

Inserta los valores obtenidos, en este caso el nombre de usuario, el contenido y la fecha generada, en la tabla de entradas.

- **Muro:** La parte GET de este API obtiene los registros del muro mediante el id, el usuario que publicó la opinión o al usuario al que se la han publicado, y lo ordena por fecha de publicación también descendiente. La parte POST hace lo mismo que en el API de las entradas, pero para la tabla del **muro**, en el que se inserta el nombre de usuario del usuario que la publicó, el nombre de usuario en el que la opinión ha sido publicada (hablo de su muro del perfil), el contenido de la opinión y la fecha generada anteriormente.

4.2.2 Inicio de sesión y registro

A la hora de implementar el inicio de sesión y el registro hago uso de los Apis creados anteriormente.

En la parte de **registro**, una vez pulsado el botón de registrarse (hecho con un event listener), se comprobarán los valores uno a uno, ya sea mediante patrones o si están vacíos. Si hay algún campo que esté mal se generarán varias alertas indicándolo.

```
const patrones = {
  'email': /^\\w+([\\._-]+)?\\w+@\\w+([\\._-]+)?(\\.\\w{2,10})+$/ ,
  "fecha": /^(\\[1-2\\][0-9]{3})\\/(\\[0-9\\]|1[0-2])\\/(\\[0-9\\]|1\\d|2\\d|3[0-1])$/ ,
}
```

Figura 21.

En la **Figura 21** se pueden ver los diferentes patrones que he utilizado para la fecha de cumpleaños y el correo electrónico. Se comprueba si el valor recogido es válido mediante ese patrón con el siguiente código:

```
patrones['email'].test(email)
```

Figura 23.

Una vez comprobados los campos y que todos sean válidos, se generará un JSON con la siguiente estructura:

```
return user = {
  "username": username,
  "name": name,
  "pass": pass,
  "email": email,
  "birthday": birthday,
  "followers": 0,
  "following": 0,
  "professional": isProfessional,
  "type": type,
}
```

Figura 24.

Seguido se hará un **fetch** o una petición al API de usuarios para insertar al usuario, pasando este JSON generado en el cuerpo de la petición.

```

case 409:
  document.querySelector("#username").style.border = "1px solid red";
  document.querySelector('.username-alert').textContent = "Username is already in use";
  document.querySelector('.username-alert').style.display = "block";
  console.log("CONFLICTO");

```

Figura 25.

Importante que se indique en el código de error 409 que el nombre de usuario ya está escogido, por lo que el usuario tendrá que elegir otro nombre.

En la parte del **login** comprueba las credenciales del usuario, tanto el nombre de usuario como la contraseña. Si todo va bien almacena el *“webToken”* generado en la parte del servidor, el nombre del usuario, el id y su enlace de imagen si tiene una en el almacenamiento local del navegador de la siguiente manera:

```

// Añade los datos a localStorage para mantener la sesión activa
localStorage.setItem('webToken', data[0].webToken);
localStorage.setItem('username', data[0].username);
localStorage.setItem('id', data[0].id);

// Comprueba si tiene una imagen
if(data[0].image != null) localStorage.setItem('image', data[0].image);
else localStorage.setItem('image', 'Gray');

```

Figura 26.

En caso de que algo vaya mal, se le notificará al usuario que hay algo que ha fallado y que compruebe sus credenciales.

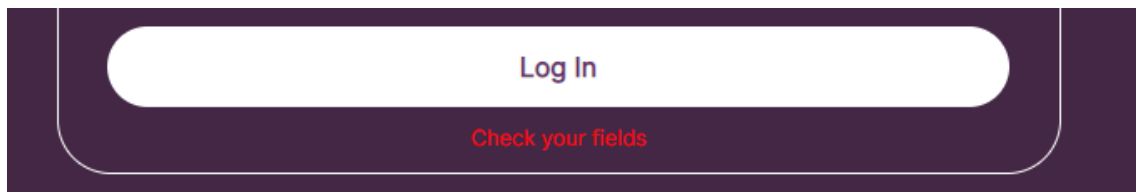


Figura 27.

4.2.3 Perfil

Aquí explicaré el tema de la vista del perfil y la edición de este en el lado del cliente.

PRIMERO LA VISTA DE PERFIL NORMAL

El perfil tiene tres secciones diferentes de entradas: Las entradas publicadas, el mural en el que los usuarios pueden publicar cosas sobre el dueño del perfil y las entradas guardadas.

Voy controlando la vista de cada una de estas secciones con un `eventListener` click y ocultando los demás contenedores dependiendo en qué opción pulse.

Para obtener los datos del usuario y mostrarlos en el perfil primero compruebo que exista un *“webToken”* con el que trabajar. Comprueba mediante el almacenamiento local si tiene foto de perfil o no, en caso de que tenga recupera el enlace mediante el almacenamiento local, ya que lo guardé con anterioridad.

En cuanto a la otra información, es necesario hacer una petición al servidor para obtener los datos.

```
let url = (`http://localhost/FillMeIn/api/usuario.php?username=${ username }`);
fetch( url )
.then(response => {
  switch (response.status) {
    case 200:
      return response.json();
    case 404:
      console.log("ERROR");
  }
})
```

Figura 28.

Con el fetch de la **Figura 28**, pasando el nombre de usuario en el enlace, se obtienen todos los datos necesarios. Ahora podremos pintarlos en la página seleccionándolos y cambiando su valor como en la siguiente imagen:

```
if (data[0]['banner'] != null) {
  document.querySelector('#banner').style.backgroundImage = `url('${ data[0]['banner'] }')`;
}
document.querySelector('.user-profile').textContent = `${ data[0]['name'] }`;
document.querySelector('.username').textContent = `@${ data[0]['username'] }`;
```

Figura 29.

Para **obtener las entradas** de este usuario y mostrarlas en su sección de entradas, hace falta otra petición, pero esta vez al API de las entradas, pasando en la URL el nombre del usuario.

Por cada entrada obtenida va creando un elemento en el HTML con la información necesaria obtenida.

En cuanto a la **visión del mural**, se hace de la misma manera que con las entradas. Haciendo un fetch al API del mural, pasando su nombre de usuario en la URL. Por cada opinión obtenida se crea una opinión en la parte HTML haciendo un fetch al usuario que la publicó, que se obtiene del nombre de usuario (*user_post*) de la opinión. Ahora se pueden rellenar la opinión con los datos obtenidos.

El usuario puede subir una opinión dentro de su mismo perfil, gracias a una llamada al API del muro, pasando la opinión a modo de JSON en el cuerpo de la petición.

Para **ver las entradas guardadas** del usuario primero hace una petición al API de usuarios pasando el nombre de usuario en la URL. Dentro de esta, si todo ha ido bien, se comprueba que tiene entradas guardadas y hace un fetch a estas pasando sus id, para después hacer otra petición para obtener toda la información del usuario que ha publicado la entrada y poder pintarla de la misma manera que antes en la página.

SEGUNDO LA EDICIÓN DE PERFIL

Utiliza la misma comprobación con los mismos patrones que en el registro, pero esta vez los campos no se pueden guardar si no se ha introducido la contraseña dos veces.

A la hora de guardar la biografía, como puede haber caracteres especiales como (‘, & ó `), llamo a una función que filtra esos símbolos por código HTML. En la **Figura 30** se puede ver la función utilizada.

```
// Filtra los símbolos del texto para pasarlos a códigos HTML
function filterSymbols(text) {
    text = text.replace(/</g, "&lt;");
    text = text.replace(/>/g, "&gt;");
    text = text.replace(/`/g, "&#96;");
    text = text.replace(/'/g, "&acute;");
    text = text.replace(/'/g, "&apos;");
    text = text.replace(/"/g, "&quot;");
    return text
}
```

Figura 30.

Tengo otra función que hace lo mismo, pero al revés, por si acaso sea necesaria. Cuando se pulsa el botón de guardar campos y está todo correcto, hará un **fetch** al API de editar, donde pasará un JSON con los campos editados. Si sale algo mal mostrará un error en pantalla. El JSON tiene la estructura que se muestra en la **Figura 31**.

```
function createUserJSON(username, name, biography, email, birthday, pass) {
    return user = {
        "username": username,
        "name": name,
        "biography": biography,
        "email": email,
        "birthday": birthday,
        "pass": pass,
    };
}
```

Figura 31.

4.2.4 Página de inicio

En la página de inicio se muestran todas las entradas y se hace uso de filtros para buscar usuarios o entradas, dependiendo de si la persona sigue al usuario o no.

INICIO Y OBTENCIÓN DE ENTRADAS

Obtengo las entradas de la misma manera que en el perfil del usuario y las añado al HTML, con la única diferencia de que, al pulsar en la imagen, nombre o nombre de usuario del usuario en concreto, me lleva a su página de perfil. Esto se consigue dentro del fetch del usuario de la entrada, donde se hará lo de la **Figura 32** para cada elemento mencionado anteriormente.

```
user_image_element.addEventListener('click', () => {
    if (localStorage.getItem('webToken') != null &&
        data[0]['id'] == localStorage.getItem('id') &&
        data[0]['username'] == localStorage.getItem('username'))
    {
        window.location.href = 'user.html';
    } else {
        window.location.href = `userprofile.php?id=${ data[0]['id'] }`;
    }
})
```

Figura 32.

La primera condición comprueba si es el usuario que ha iniciado sesión, para llevarle a su página de perfil con la opción de editar. La otra condición los lleva a otro **js** donde tendrán la opción de seguir el perfil.

En cada entrada hay un icono de guardado que cambia de clase al ser pulsado.

FILTROS EN EL CAMPO DE TEXTO

Cuando el campo de texto del menú superior es modificado, irán apareciendo los usuarios cuyo nombre coincida o contenga el valor obtenido de este. Si el campo de texto está vacío, obtiene todas las entradas, pero si tiene algo escrito, hará lo mencionado anteriormente.

Para obtener a los usuarios con el valor del campo, se hace un **fetch** al API de filtrado de usuarios, donde se creará un contenedor para cada usuario, con su foto, su nombre y su nombre de usuario. Si el usuario es una cuenta profesional, también aparecerá el tipo de cuenta que es.

En cuanto a los filtros de mostrar todos los usuarios o sólo a los que sigue, se selecciona el que tenga la clase **selected** y, en caso de que sea sólo el de los seguidos, se hace un **fetch** al API de usuarios pasando el nombre de usuario del usuario que ha iniciado sesión. Obtenidos los datos se hace una petición a las entradas y por cada entrada obtenida otra petición a los usuarios, con el nombre del usuario de la entrada en la URL. Aquí va comprobando si el id del usuario de la entrada aparece en la lista de usuarios a los que sigue el usuario con la sesión iniciada. En caso de que aparezca la entrada se muestra. Esto se puede ver en la **Figura 33**.

```
JSON.parse(data_user[0]['following']).forEach(item => {
  if (item == data_user_found[0]['id']) {
```

Figura 33.

Data_user es el usuario con la sesión iniciada, y **data_user_found** es el usuario que ha subido la entrada.

SUBIDA DE ENTRADAS Y GUARDADO DE ESTAS

Pulsado el botón de Subir entrada, genera un JSON de la entrada y con un **fetch** a las entradas la sube.

Para guardar una entrada, una vez pulsado el icono de guardar, realizo una petición al API de usuarios con el nombre de usuario. En la **Figura 34** añado las entradas guardadas del usuario a un array, y en la **Figura 35** compruebo si la entrada va a ser guardada o no, para añadirla al array o para eliminarla.

```
let isSaved = false;
if (icon.classList.contains('bi-bookmark-fill')) {
  isSaved = true;
}
let saved_entries = [];
if (data[0]['saved_entries'] != null && data[0]['saved_entries'].length > 0) {
  JSON.parse(data[0]['saved_entries']).forEach(item => {
    saved_entries.push(item);
  })
}
```

Figura 34.

```
if (isSaved) {
  saved_entries.push(e.target.getAttribute('data-id'));
} else {
  saved_entries = saved_entries.filter(item => item != e.target.getAttribute('data-id'));
}
```

Figura 35.

Finalmente hace un **fetch** al API de filtros de usuario y añade la entrada pasada como JSON.

4.2.5 Perfil de otros usuarios.

Lo único que resaltar aquí es la parte de seguir al usuario, todo lo demás se hace exactamente igual que en el perfil del usuario con la sesión iniciada.

```
// Evento para el botón de Seguir y Dejar de seguir
let follow = document.querySelector('#follow-bt');
follow.addEventListener('click', () => {
  if (!follow.classList.contains('unfollow')) {
    getUserInfo(username, true, false, false);
  } else {
    let remove = true;
    getUserInfo(username, true, false, remove);
  }
  follow.classList.toggle('unfollow');
})
```

Figura 36.

En la **Figura 36**, lo que hace es ir cambiar la clase del botón cuando se pulsa. Dependiendo de su clase, lo elimina de seguidos o lo añade. En cuanto los parámetros de la función **getUserInfo**, se pasa el nombre del usuario, una variable booleana que sirve para determinar si se añade el usuario o no, que se explicará más adelante, el segundo es otro booleano para determinar si se le sigue actualmente o no, y el tercero otro booleano para determinar si se elimina o no.

Si el usuario que ha iniciado sesión no le sigue, hago lo mismo que hice con las entradas guardadas. Dentro del **fetch** al usuario creo un array para los seguidores de este usuario y compruebo con el último parámetro pasado en la función si se quiere dejar de seguir o no. Con un **fetch** lo paso al servidor y lo actualiza en la base de datos.

En caso de que el tercer parámetro fuese false, que significa que no le sigue, llamaría a otra función que haría lo mismo pasando la variable **add**, pero esta vez para los seguidos, que solo se ejecuta si se quiere añadir. Si esa variable es falsa lo único que hace es cambiar la clase al botón de seguir para que aparezca como que ya le sigue, en caso de que lo encuentre como en la **Figura 37** aparece.

```
} else if (data[0] && !add) {
  let following = data[0]['following'];
  if (following != null) {
    JSON.parse(following).forEach(item => {
      if (item == user_follow) {
        document.querySelector('#follow-bt').classList.add('unfollow');
      }
    })
  }
}
```

Figura 37.

5. Conclusiones finales

5.1 Grado de cumplimiento de los requisitos fijados

Marzo	1 ^a semana	2 ^a semana	3 ^a semana	4 ^a semana
Documentación servidor para red social	5 h			
Documentación sobre JWT		16 h		
Diseño de interfaz, preparar CSS			14 h	

Pantalla principal	17 h			
Abril	1ª semana	2ª semana	3ª semana	4ª semana
Inicio y registro de usuarios		14 h		
Implementación de los servicios en PHP		10 h		
Creación y subida de entradas HTML				19 h
Edición del perfil			7 h	
Mayo	1ª semana	2ª semana	3ª semana	4ª semana
Seguimiento de usuarios	8 h			
Guardar entradas		4 h		
Valorar entradas				
Filtros de cuentas y entradas			3 h	

Horas totales: 117 horas

5.2 Propuestas de mejora o ampliaciones futuras

Las posibles mejoras o ampliaciones futuras son las siguientes:

- **Valoración de entradas:** Pensado anteriormente. Un usuario podrá valorar una entrada, ya sea mediante unas estrellas o mediante “me gusta” acumulados.
- **Valoración de perfiles:** A pesar de tener un muro, este no deja de ser un lugar en el que la gente exprese sus opiniones sobre la cuenta, pero no dan puntuación. Será buena idea que las cuentas profesionales tuviesen un sistema de puntuación parecido al de las valoraciones de las entradas mencionado arriba.
- **Cuentas privadas:** Un usuario puede decidir si su cuenta será pública o privada. Las personas no podrán ver el contenido del usuario privado a no ser que éste acepte su solicitud de seguimiento, como se hace en muchas otras redes sociales.
- **Mensajería:** Algo que no estaría de menos es tener un propio sistema de mensajería privado entre usuarios.

6. Guías

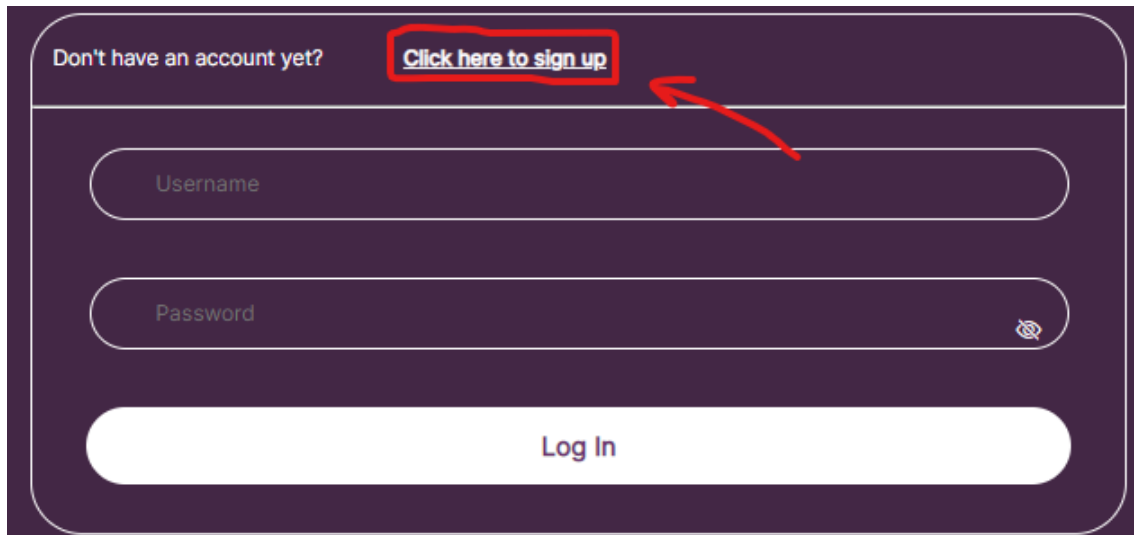
6.1 Guía de uso

Nada más entrar en la página, el usuario podrá acceder desde el menú superior al inicio en la pestaña “**Home**”, a información de la página, en la pestaña “**About**” e iniciar sesión en “**Login**” en la parte derecha del menú superior.



Figura 38.

En la pantalla de Inicio de sesión hay un enlace en la parte superior que lleva a la pantalla de registro. En esta pantalla hay un campo especial, que aparece si se ha seleccionado la opción de **YES** en el campo de “Cuenta profesional”. Estas cuentas se dedican especialmente a subir contenido relacionado con el tipo de cuenta que han escrito en este campo.



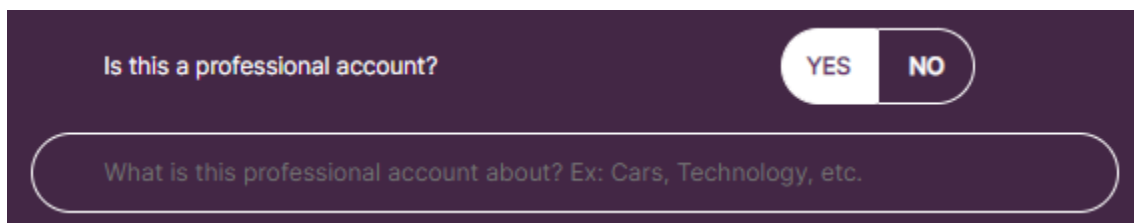
Don't have an account yet? [Click here to sign up](#)

Username

Password

Log In

Figura 39.



Is this a professional account? YES NO

What is this professional account about? Ex: Cars, Technology, etc.

Figura 40.

Una vez creado el perfil o iniciado sesión, se podrá acceder a la ventana del perfil pulsando sobre la imagen gris o imagen de perfil del menú superior en la parte derecha. También se puede acceder desde la página de inicio, en el menú lateral derecho, en el apartado de "Perfil". Así mismo, podrá cerrar la sesión pulsando en el icono de al lado.



Figura 41.

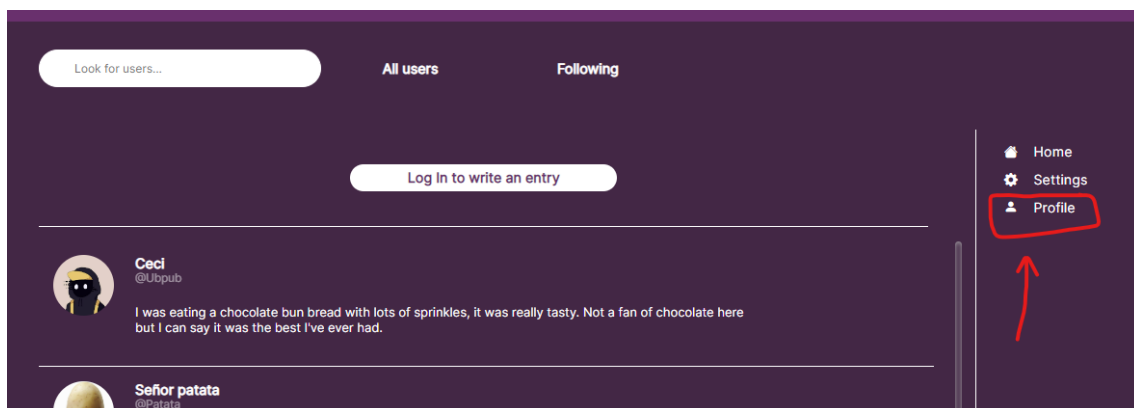


Figura 42.

El usuario podrá editar su perfil pulsando en el botón de “Editar perfil” que aparece en la ventana de este. En esta pestaña, el usuario podrá modificar el nombre, la biografía, la fecha de nacimiento y el email, escribiendo su contraseña dos veces para poder guardar los cambios.

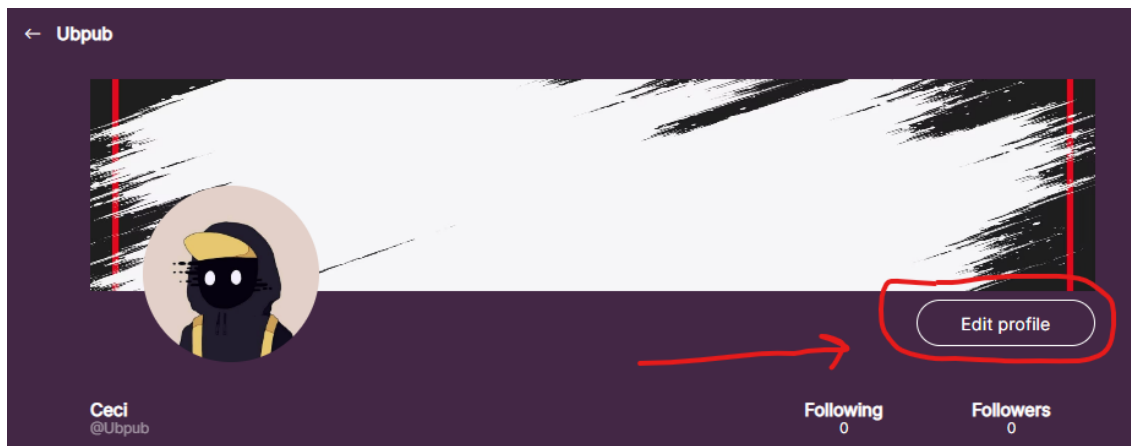


Figura 43.

En la página de inicio, si el usuario ha iniciado sesión, podrá crear entradas con el campo de texto que aparece por encima de todas las entradas. Una vez escrito lo que se quiere publicar, sólo deberá pulsar el botón de “Subir entrada” para publicarla.



Figura 44.

Si se quiere ver el perfil de un usuario, sólo se deberá pulsar en su nombre o en su foto, y le llevará a la página de su perfil, donde podrán seguirle y mirar las entradas que ha publicado.

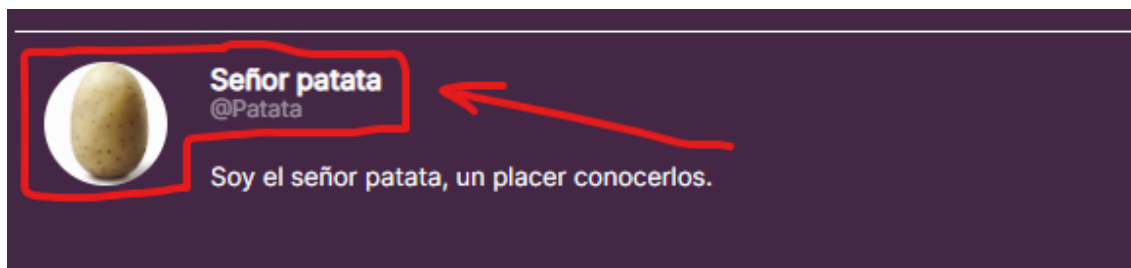


Figura 45.



Figura 46.

Se puede cambiar la contraseña desde Ajustes como se ve en la **Figura 47**. Para ello es necesario escribir la contraseña actual y la nueva contraseña dos veces.

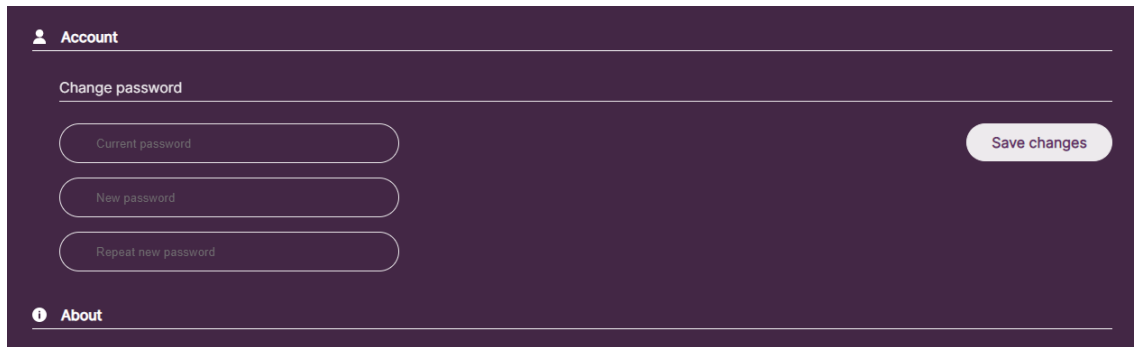


Figura 47.

En la **Figura 48** se ve el uso de filtros. En la caja de texto puede escribir cualquier cosa y buscará a los usuarios que tengan el nombre parecido. También buscará aquellas cuentas profesionales cuyo tipo de cuenta coincida con el cuadro de búsqueda. En cuanto a las dos selecciones de arriba, puede filtrar en todos los usuarios o sólo personas a las que sigue. Este filtro también se aplica a la vista de entradas.

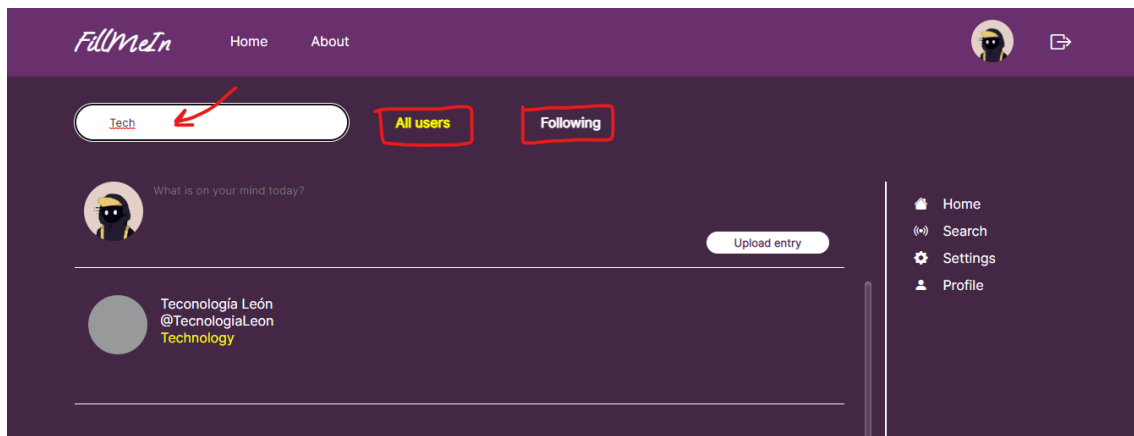


Figura 48.

Los usuarios pueden buscar también en una sección especializada sólo para usuarios. Su apariencia sería la misma que la de las entradas, pero con más filtros.

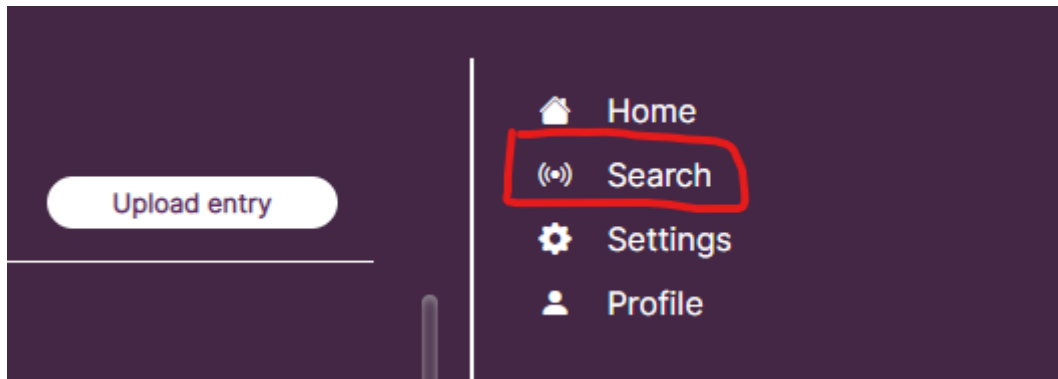


Figura 49.

El usuario puede elegir buscar cuentas ya sean personales, profesionales o ningún filtro en concreto.

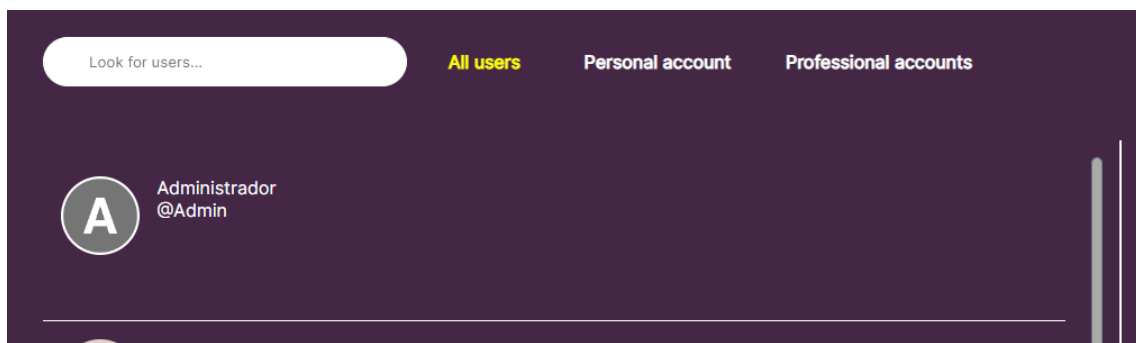


Figura 50.

6.2 Guía de instalación

En este proyecto utilicé la página AlwaysData. En la **Figura 51** se ve la página creada con su enlace. Esta será la página en la que vamos a desplegar el proyecto.

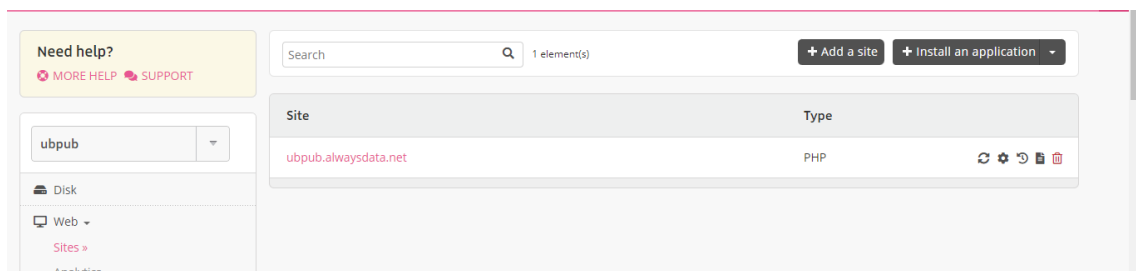


Figura 51.

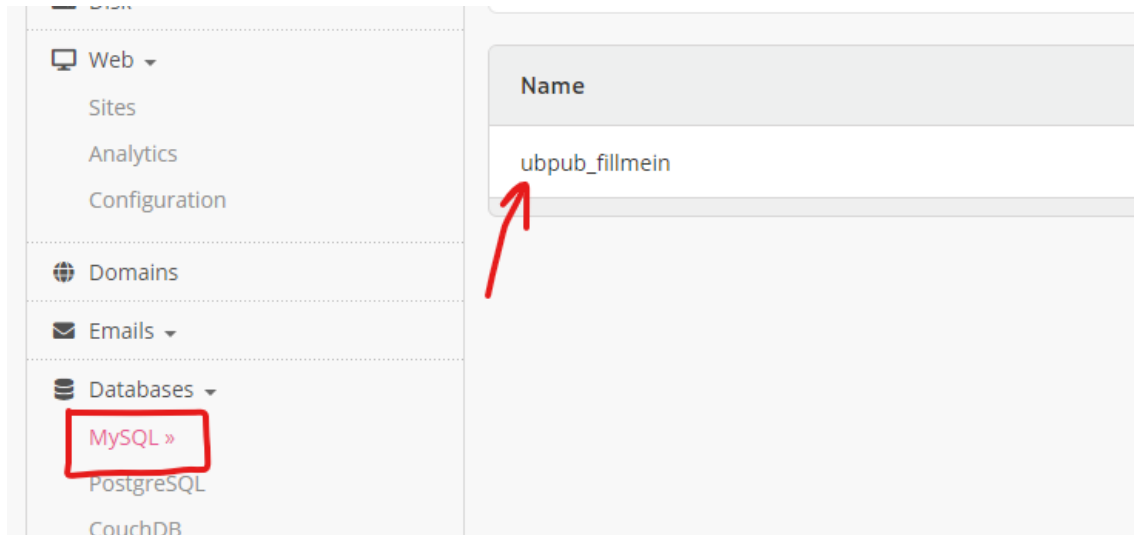


Figura 52.

Creamos una base de datos con el nombre que queramos y accedemos a PHPMyAdmin desde la propia página, te marcará un enlace en la parte superior. Desde ahí se accede con el nombre de usuario de tu cuenta y tu contraseña. Exportaremos nuestra base de datos ahí mismo.

Pasamos a la parte de SSH, que se encuentra más abajo. Ejecutaremos el enlace que aparece encima del usuario SSH creado por defecto e iniciaremos sesión en la consola que se abra. Ahí podemos eliminar el index.html que viene dentro de la carpeta www y clonar nuestro repositorio (Si hemos ido subiendo los cambios a GitHub) en esa misma carpeta. Una vez hecho esto, al abrir la página de la **Figura 51**, aparecerá nuestra página web.

7. Referencias bibliográficas

- 1- [Documentación de Always Data.](#)
- 2- [Pasar datos de PHP a LocalStorage.](#)
- 3- [Qué es un JWT - KeepCoding.](#)
- 4- [JavaScript evento onClick – Developer Mozilla.](#)
- 5- [Cambiar ScrollBar por defecto – CSS-TRICKS.](#)
- 6- [Obtener la URL actual con JavaScript – CYBMETA.](#)
- 7- [Obtener la URL actual con PHP – Bufo.](#)
- 8- [Iconos de Bootstrap.](#)
- 9- [Obtener fecha y hora actual en PHP – Código Root.](#)

8. URL del proyecto

ubpub.alwaysdata.net