

# Trabajo final 2 ° ASIR

## Sistema de Autentificación y Geolocalización para trabajo en remoto

---

Sergio Pérez Ríos - 2° ASIR

8 jun 2024

Tutor: Fernando Raya Díaz

## Índice

1. Introducción	2
2. Instalación de servidor mysql y creación de la base de datos de empleados.	3
3. Instalación de servidor apache y desarrollo de la página web.	10
4. Instalación de php y programación del backend.	16
5. Registro de empleado.	39
6. Panel administrativo.	45
7. Programación del login con reconocimiento facial para jefes.	62
8. Prueba de inicio de sesión de empleado con biometría.	72

---

Máquina	Dirección IP
MySQL	192.168.1.202
Servidor apache (frontend)	192.168.1.140
Servidor apache (backend)	192.168.1.222

## 1. Introducción

Es muy común hoy en día el trabajo en remoto, y más en el mundo de la informática. El trabajo en remoto es muy cómodo para el trabajador, ya que no tiene que abandonar su vivienda, tiene las herramientas que quiera a su disposición y nadie le vigila. Ésto siempre me había parecido muy llamativo, pero haciendo las prácticas de empresa en remoto me dí cuenta de que desde el punto de vista de la empresa, se pierde mucho control sobre sus empleados, ya sea administrativo, de eficiencia de los mismos, etc... .

Por ello decidí diseñar un sistema en el que se pudiera acercar el trabajo en remoto a estar físicamente en la empresa. Así, en las próximas páginas explico cómo he diseñado dicho sistema, que registra las ubicaciones de trabajo de los empleados al momento de fichar. Ésto evidentemente solo sería legal si los dispositivos que usan para trabajar los empleados son dispositivos de empresa.

Por ello, se va a implementar un sistema para que los empleados puedan fichar desde su casa y, de manera que la ubicación de los mismos quede registrada para su consulta en cualquier momento. El fichaje se hará desde una página web, donde se habilitará un “Área de empleados” en la misma en la que solo puedan acceder los empleados, de manera que nadie se puede registrar, sólo podrán iniciar sesión si figuran en la base de datos de la empresa como empleados.

Además, para mayor seguridad, aquí es donde entra el sistema de reconocimiento facial. Ésto es porque, como comentaré más abajo, los empleados se dividirán en jefes y no jefes; los que sí lo son tienen acceso a información delicada de los usuarios, como la ubicación de fichaje, datos personales, etc.. Y el reconocimiento facial solo sería una forma de blindar el sistema de login para complicar el acceso a la información.

## 2. Instalación de servidor mysql y creación de la base de datos de empleados

He instalado mysql 8.0 en un servidor Ubuntu. Tras asignar contraseña al usuario **root** vamos a crear un usuario para gestionar la base de datos de los empleados. Dicho usuario se llamará **empleados**.

```
mysql> CREATE USER 'empleados'@'%' IDENTIFIED BY 'Departamento1!';  
Query OK, 0 rows affected (0,01 sec)
```

Crearemos la base de datos empleados vacía a la que llamaremos **employees\_db**, y le asignaremos privilegios a **empleados** sobre la nueva base de datos.

```
mysql> CREATE DATABASE employees_db;  
Query OK, 1 row affected (0,00 sec)
```

```
ERROR 1045 (42100): No database selected  
mysql> GRANT ALL PRIVILEGES ON employees_db.* TO 'empleados'@'%';  
Query OK, 0 rows affected (0,00 sec)
```

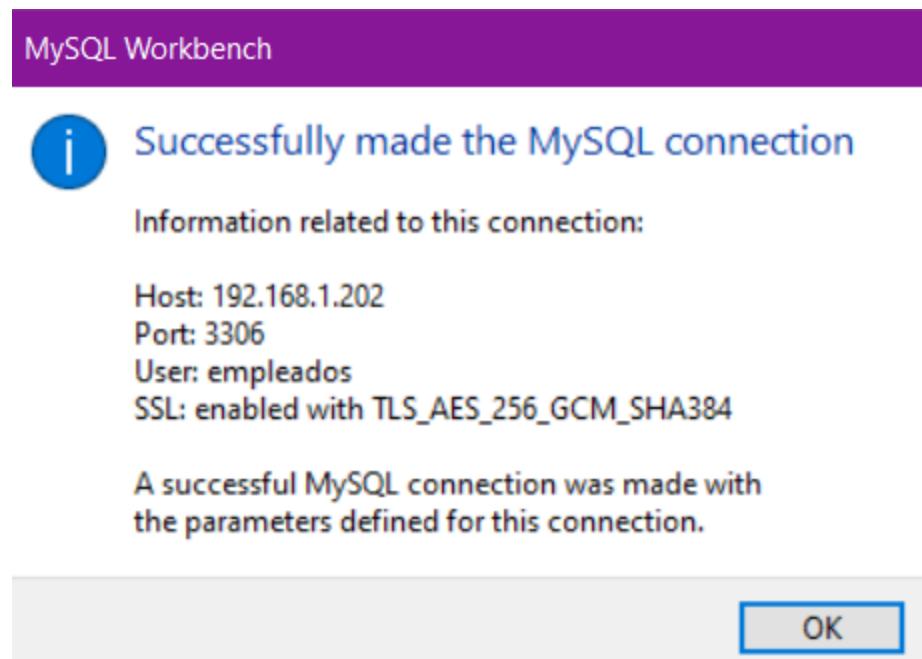
Refrescamos privilegios.

```
mysql> FLUSH PRIVILEGES;  
Query OK, 0 rows affected (0,01 sec)
```

Editamos el archivo de configuración de mysql para permitir que los usuarios puedan conectarse desde fuera. El archivo en cuestión es **/etc/mysql/mysql.conf.d/mysqld.cnf** y editamos la siguiente directiva y reiniciamos el servicio de mysql:

```
bind-address = 0.0.0.0
```

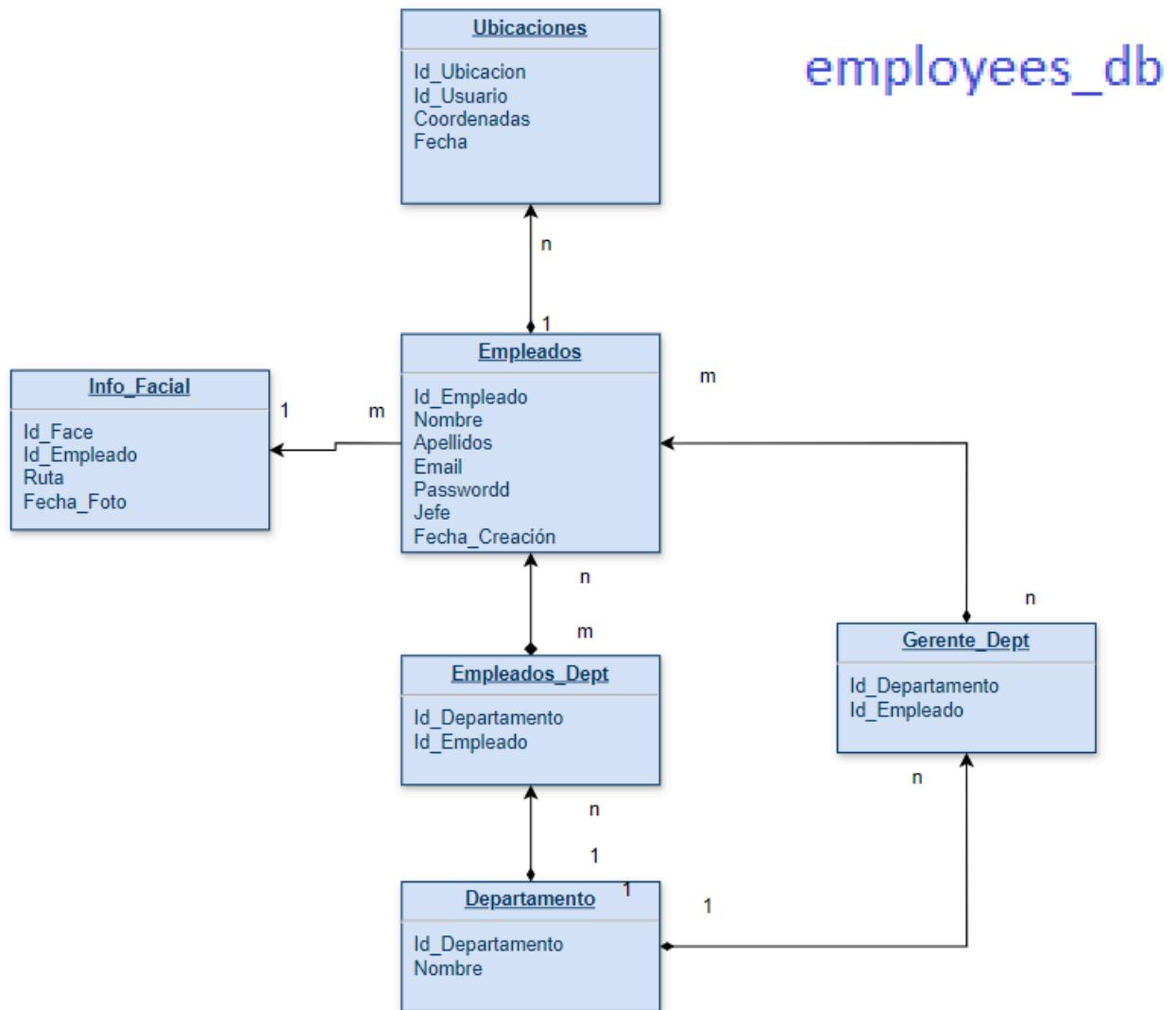
Si probamos la conexión, vemos que podemos conectarnos correctamente con el usuario **empleados**:



## Creación de la base de datos

Para el diseño de la base de datos he creado el siguiente diagrama en el que podemos discernir usuarios (empleados) y jefes, acordes a su respectivo departamento, y las **ubicaciones** diarias en el momento de fichar. Además, se almacena en dicha base de datos la información facial de los empleados (jefes).

El diagrama de entidad-relación para aproximarnos a la forma que tendrá la base de datos es el siguiente:



**ACLARACIONES::**

- El campo **Passwordd** de la tabla **Empleados** se llamará **Passwordd** debido a que la palabra Password es una palabra reservada en mysql.
- Para distinguir a los empleados de los jefes la tabla empleados contendrá un campo obligatorio de tipo carácter denominado jefe y que podrá obtener los valores “S” de Sí o “N” de No.
- La tabla **Ubicaciones** tendrá un campo **Coordenadas** de tipo **POINT** (ver más abajo), que es perfecto para guardar datos de tipo punto, ya que contiene un valor para el eje X y otro para el eje Y.
- La tabla **Info\_Facial** contendrá la información acerca de las fotos que consultará el programa de reconocimiento facial para autenticar a los jefes. Simplemente contiene el id del empleado, la ruta donde se almacenan las fotos y la fecha en la que se tomó la foto.

Para crear la base de datos **employees\_db** he usado el siguiente script:

```
use employees_db;

-- Tabla Empleados
CREATE TABLE Empleados (
    Id_Empleado CHAR(36) PRIMARY KEY,
    Nombre VARCHAR(100) NOT NULL,
    Apellidos VARCHAR(100) NOT NULL,
    Email VARCHAR(100) NOT NULL,
    Passwordd VARCHAR(100) NOT NULL,
    Jefe CHAR(1) NOT NULL CHECK (Jefe IN ('S', 'N')),
    Fecha_Creacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
    CURRENT_TIMESTAMP NOT NULL
);
```

## SISTEMA DE AUTENTIFICACIÓN Y GEOLOCALIZACIÓN PARA TRABAJO EN REMOTO

```
-- Tabla Departamentos
CREATE TABLE Departamentos (
    Id_Departamento INT AUTO_INCREMENT PRIMARY KEY,
    Nombre VARCHAR(100) NOT NULL
);

-- Tabla Ubicaciones
CREATE TABLE Ubicaciones (
    Id_Ubicacion INT AUTO_INCREMENT PRIMARY KEY,
    Id_Empleado CHAR(36) NOT NULL,
    Coordenadas POINT NOT NULL,
    Fecha TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP NOT NULL,
    FOREIGN KEY (Id_Empleado) REFERENCES Empleados(Id_Empleado)
);

-- Tabla Info_Facial
CREATE TABLE Info_Facial (
    Id_Face INT AUTO_INCREMENT PRIMARY KEY,
    Id_Empleado CHAR(36) NOT NULL,
    Ruta POINT NOT NULL,
    Fecha_Foto TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP NOT NULL,
    FOREIGN KEY (Id_Empleado) REFERENCES Empleados(Id_Empleado)
);

-- Tabla Empleados_Dept
CREATE TABLE Empleados_Dept (
    Id_Departamento INT NOT NULL,
    Id_Empleado CHAR(36) NOT NULL,
    PRIMARY KEY (Id_Departamento, Id_Empleado),
    FOREIGN KEY (Id_Departamento) REFERENCES
Departamentos(Id_Departamento),
    FOREIGN KEY (Id_Empleado) REFERENCES Empleados(Id_Empleado)
);
```

```
-- Tabla Gerente_Dept
CREATE TABLE Gerente_Dept (
    Id_Departamento INT NOT NULL,
    Id_Empleado CHAR(36) NOT NULL,
    PRIMARY KEY (Id_Departamento, Id_Empleado),
    FOREIGN KEY (Id_Departamento) REFERENCES Departamentos(Id_Departamento),
    FOREIGN KEY (Id_Empleado) REFERENCES Empleados(Id_Empleado)
);
```

Del script debo destacar que el **Id\_Empleado** se generará automática y aleatoriamente mediante la función UUID() de mysql. Además, es imposible la repetición de ID's deido a que dicha función genera números de 128 bits representado como una cadena utf-8 mediante una combinación de la MAC del equipo y un timestamp . El formato en número hexadecimal será el siguiente:

**aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeee**

Para tener acceso al backend de mi aplicación web, voy a crear un usuario para mí. Para ello usare el siguiente código. Además, como éste usuario lo crearé a mano, voy a transformar la contraseña que usaré a su hash usando el algoritmo bcrypt, que es el que usaré más tarde para hashear las contraseñas de los usuarios.

```
use employees_db;

INSERT INTO Empleados( ID_Empleado, Nombre, Apellidos, Email, Passwordd, Jefe)
VALUES (
    UUID(),
    "Sergio",
    "Pérez Ríos",
    "sperrio2706@ieszaidinvergeles.org",
    "$2a$12$wAVgRUjxHAoo18L0B6oZ2.c0.Q1/t5Q9iGE8c7vbQfouD1sNCB1gm",
```

## SISTEMA DE AUTENTIFICACIÓN Y GEOLOCALIZACIÓN PARA TRABAJO EN REMOTO

```
"S"  
);
```

Si consultamos la tabla empleados, vemos que se ha creado correctamente:

Id_Empleado	Nombre	Apellidos	Email	Passwordd	Jefe	Fecha_Creacion
da4b3362-227d-11ef-882c-0800275e5af5	Sergio	Pérez Ríos	sperrio2706@ieszaidinvergeles.org	\$2a\$12\$wAVgRUjxHAoo18L0B6oZ2.c0.Q1/t5Q...	S	2024-06-04 14:22:20
NULL	NULL	NULL	NULL	NULL	NULL	NULL

### 3. Instalación de servidor apache y desarrollo de la página web.

Para instalar apache y php solo tenemos que ejecutar los dos siguientes comandos:

```
sudo apt install apache2  
sudo apt install php libapache2-mod-php php-mysql
```

Una vez instalados apache y php vamos a configurar el virtualhost. En mi caso, tengo como virtualhost principal bettercallsergio.es, que es dominio que poseo. Hago uso del módulo de apache **rewrite** pero por defecto viene activado. Uso dicho módulo para redirigir las peticiones http a https, de manera que obligo a usar el protocolo seguro. El segundo virtualhost es equivalente al primero con la diferencia de que contiene **www** como subdominio. El tercero es el interesante, uso el subdominio **app.bettercallsergio.es** para redirigir el tráfico al servidor backend usando apache como proxy inverso.

Quiero aclarar que como quiero que este proyecto sea posible de utilizar desde cualquier ordenador, estoy utilizando el puerto 8080 de mi router para redirigir las peticiones a mi servidor backend. De esta manera utilizo el mismo dominio para acceder a mis dos servidores.

```
<VirtualHost *:80>  
  
    ServerAdmin webmaster@localhost  
    ServerName bettercallsergio.es  
    DocumentRoot /var/www/html  
  
  
    ErrorLog ${APACHE_LOG_DIR}/error.log  
    CustomLog ${APACHE_LOG_DIR}/access.log combined
```

```
RewriteEngine on
RewriteCond %{SERVER_NAME} =bettercallsergio.es
RewriteRule ^ https:// %{SERVER_NAME} %{REQUEST_URI}
[END,NE,R=permanent]

</VirtualHost>

<VirtualHost *:80>
    ServerName www.bettercallsergio.es

    RewriteEngine on
    RewriteCond %{SERVER_NAME} =www.bettercallsergio.es
    RewriteRule ^ https:// %{SERVER_NAME} %{REQUEST_URI}
    [END,NE,R=permanent]
</VirtualHost>

<VirtualHost *:80>
    ServerName app.bettercallsergio.es

    ProxyPreserveHost On
    ProxyPass / https://bettercallsergio.es:8080/
    ProxyPassReverse / https://bettercallsergio.es:8080/

    #RewriteEngine on
    #RewriteCond %{SERVER_NAME} =app.bettercallsergio.es
    #RewriteRule ^ https://bettercallsergio.es:8080
    [END,NE,R=permanent]

</VirtualHost>
```

Nuestro servidor apache va a actuar también como proxy inverso, así que para pueda usar las instrucciones de proxy tenemos que habilitar los siguientes módulos. Para ello usaremos el programa **a2enmod**:

```
sudo a2enmod proxy proxy_http proxy_balancer lbmethod_byrequests
```

Reiniciamos apache con systemctl y lo siguiente será generar el certificado para activar https. Para ello instalamos los paquetes a continuación. Vamos a usar un bot de certificados que nos generará uno de Let's Encrypt.

```
sudo apt install certbot python3-certbot-apache
```

Una vez instalado, lo ejecutamos con **sudo certbot --apache**, a lo que nos preguntará sobre qué dominio/os.

```
usuario@ubuntu:~$ sudo certbot --apache
Saving debug log to /var/log/letsencrypt/letsencrypt.log

Which names would you like to activate HTTPS for?
-----
1: bettercallsergio.es
2: www.bettercallsergio.es
-----
Select the appropriate numbers separated by commas and/or spaces, or leave input
blank to select all options shown (Enter 'c' to cancel): 2
```

Reiniciamos apache, y ya tenemos configurado el proxy inverso. Ésto también permitirá

Para esta cuestión, vamos a editar el archivo **000-default.conf** y vamos a añadir las siguientes directivas.

Voy a mostrar las páginas del servidor.

```
> error  
> images  
<> contacto.html  
<> index.html  
<> login.html  
<> nosotros.html  
# styles.css
```

### [index.html](#)



La parte interesante de la página es el botón llamado **Área de Empleados**, que lleva al usuario a un formulario (**login.html**). El formulario se ve así:



Los usuarios loguean con el correo registrado y su contraseña. Y este formulario les lleva al servidor backend en el caso de que las credenciales introducidas sean correctas; si no lo son, son redirigidos a la página principal del frontend. Hay que tener en cuenta que el archivo php que procesa el formulario está en el backend. El formulario luce así:

```
<form action="https://app.bettercallsergio.es/login.php" method="POST">
  <h2>Inicio de sesión</h2>
  <pre>Solo para empleados</pre>
  <br>
  <label for="email">Email:</label><br>
  <input type="email" id="email" name="email" required placeholder="Introduzca su correo electrónico"><br>
  <label for="password">Contraseña:</label><br>
  <input type="password" id="password" name="password" required placeholder="Introduzca su contraseña"><br>
  <input type="submit" value="Enviar">
</form>
```

Gracias al proxy inverso (el propio frontend) nos lleva al fichero **login.php**. Mostraré dicho fichero en el punto siguiente, cuando hablemos del backend.

Las páginas contacto.html y nosotros.html son simplemente galerías de imágenes para simular una página de empresa.

Por otro lado, en la imagen del formulario se vé que no hay un botón para registrarse como usuario, esto es porque el login es privado, dirigido solo para empleados, como se indica. Para registrar un usuario, sería necesario contactar con el administrador, facilitar los datos y éste creará el usuario con un formulario solamente accesible desde el backend.

Quiero aclarar una cosa. Al principio pensé en usar OAuth 2.0 porque dicha tecnología permite iniciar sesión con las cuentas de google y trabaja bien con php, y con ello ahorrarme parte del proceso de registro, pero esa opción no servía por el hecho de que quería que mis usuarios fueran lo más privados posible.

## 4. Instalación de servidor apache y php y programación del backend.

Instalamos apache y php como en el punto anterior, y generamos el certificado para tener https también en este servidor. La diferencia es que este servidor tendrá un certificado autofirmado, lo que quiere decir que el navegador notificará que la página no es segura porque la Autoridad Certificadora no es válida, ya que es autofirmado, pero si utilizará encriptación https, por lo que los datos que se le envíen desde el formulario del servidor frontend estarán encriptados.

Hé de señalar que no he utilizado ningún framework para el desarrollo de la aplicación web, pero he tenido en cuenta algunas de las cuestiones que un framework trata, como es la conversión de contraseñas a su hash, haciendo uso de funciones que trae php por defecto. En mi caso, he usado **password\_hash()**, que usa por defecto el algoritmo **bcrypt**, el cual es bastante seguro, ya que permite rehacer ciclos de hasheo, es decir, se le puede especificar cuantas veces se va a aplicar el algoritmo de hash sobre la contraseña, lo que permite una mayor seguridad frente a los ataques de fuerza bruta para averiguar la contraseña en caso de que la información de la base de datos sea robada.

Con esto, ya puedo explicar como he desarrollado el backend. Los archivos que contiene ahora mismo son los siguientes:

```

> error
> images
< includes
  footer.php
  functions.php
  header.php
  Login.class.php
  Register.class.php
  contacto.html
  enviar_fichaje.html
  fichaje_correcto.html
  fichar.html
  guardar_fichaje.php
  index.html
  login.php
  logout.php
  mi_ubicacion_mapa.h...
  profile.php
# styles.css

```

Primero debería hablar de los archivos en la carpeta **includes**.

- **footer.php:**

```
<?php
ob_start();
session_start();

if(!isset($_SESSION['valid'])){
    header('Location: https://bettercallsergio/index.html');
}

?>

<footer>
    <div class="container">
        <p>Sergio Pérez Ríos [2024] Empresa de ejemplo</p>
    </div>
    <br><hr><br>
    <p>Bienvenido, <?php echo $_SESSION['email'];?></p>
    <a href="logout.php" id="logout-link">Cerrar sesión</a>
</footer>
```

Este archivo simplemente añade al archivo html en el que se incluya un pie de página que contiene el correo con el que se ha iniciado la sesión y un botón que permite cerrar la sesión (no hago uso de él de momento). Para que funcione primero he llamado a la función **start\_session()** que inicializa los datos de sesión, y a la función **ob\_start()**, que activará el almacenamiento en búfer de salida. Mientras el almacenamiento en búfer de salida está activo, no se envía ningún resultado desde el script, pero el resultado se almacena en un búfer interno.

Además, hago una comprobación para que si la sesión no es válida, devuelva al usuario a la página principal del frontend.

- **functions.php:**

```
<?php

# Funcion que filtra cadenas
function secure_data($data){
    # quita los espacios al principio y a final
    $data = trim($data);

    # quita las comillas simples de la cadena
    $data = stripslashes($data);

    # cambia caracteres especiales a su notación html
    $data = htmlspecialchars($data);

    return $data;
}
```

Este archivo contiene todas las funciones que voy a ir utilizando en los diferentes archivos. La primera es una función para filtrar cadenas con las funciones descritas en los comentarios, y que al final devuelve el dato pasado como parámetro filtrado.

```
# Funcion que transforma una contraseña en texto plano en su hash
function hash_password($password){
    return password_hash($password, PASSWORD_DEFAULT);
}
```

La siguiente función hashea la contraseña pasada en texto plano como argumento. Como he mencionado anteriormente, utiliza la función **password\_hash()**, con el parámetro **PASSWORD\_DEFAULT**, le indicamos que utilice como algoritmo el que usa por defecto, **bcrypt**.

```

# Funcion que se conecta a la base de datos y devuelve el objeto pdo
function connectionDB(){
    $host = '192.168.1.202:3306';
    $dbName = 'employees_db';
    $user = 'empleados';
    $pass = 'Departamento1!';
    $hostDB = 'mysql:host='.$host.';dbname='.$dbName.';';
}

try{
    $connection = new PDO($hostDB,$user,$pass);
    $connection->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    return $connection;
} catch(PDOException $e){
    die('ERROR: '.$e->getMessage());
}
}

```

La última es una función que se conecta a la base de datos. He de señalar que la contraseña se ve en texto plano, así que lo ideal sería poner ahí el hash de la contraseña y que una función se encargue de convertirlo.

#### - header.php:

Este fichero es similar a **footer.php**, pero en lugar de reemplazar el footer, reemplaza el header. Se mostrará igualmente un botón para cerrar sesión.

```

<?php
ob_start();
session_start();

# Si la sesión no es valida me redirige al frontend
if(!isset($_SESSION['valid'])){[
    header('Location: https://bettercallsergio.es/index.html');
]}

>>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Mi empresa</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <header>
        <div class="container">
            <div class="logo">
                
            </div>
            <nav>
                <ul>
                    <li><a href="index.html">Home</a></li>
                    <li><a href="fichar.html">Fichar</a></li>
                    <li><a href="contacto.html">Contacto</a></li>
                </ul>
            </nav>
            <div class="employee-area">
                <p>Bienvenido/a <?php echo $_SESSION['email'];?> !</p>
                <a href="logout.php" id="logout-link">Cerrar sesion</a>
            </div>
        </div>
    </header>

```

- **Login.class.php:**

```

<?php
require_once('functions.php');

# Clase que inicia sesion
class Login{
    private $email;
    private $password;
    private $connectionDB;

    public function __construct($email, $password)
    [
        $this->email = secure_data($email);
        $this->password = secure_data($password);
        $this->connectionDB = connectionDB();

        if($this->check_email_exists()){
            $passInDB = $this->get_pass_in_db();

            # Compara el hash de la contraseña en la base de datos
            $auth = password_verify($this->password,$passInDB);

            # Si el hash y la contraseña introducida coinciden
            if($auth){
                ob_start();
                session_start();
                $_SESSION['email'] = $this->email;
                $_SESSION['valid'] = true;

                # Redirige a fichero privado
                header('Location: index.html');
            } else {
                # Acceso denegado (contraseña no coincide)
                header('Location: https://bettercallsergio.es/index.html');
            }
        } else {
            # Acceso denegado (email no registrado)
            header('Location: https://bettercallsergio.es/index.html');
        }
    ]
}

```

Este archivo contiene incluye las funciones vistas anteriormente, de las cuales solo usaremos la de conectarnos a la bbdd.

A continuación viene la clase **Login** con los atributos **email**, **password** y **connectionDB**.

En el constructor de la clase se le pasa como argumento el email y la contraseña, y las almacena en los atributos tras haberlas filtrado con la función **secure\_data()** vista anteriormente.

En el atributo connectionDB guarda el objeto de tipo PDO (la conexión a la bbdd). El constructor se encarga de llamar al método **check\_email\_exists()**, que devolverá **true** si el email figura en la base de datos, y **false** en caso contrario. El método es el siguiente:

```
private function check_email_exists(){
    $stmt = $this->connectionDB->prepare('SELECT * FROM Empleados WHERE Email=:email');
    $stmt->bindParam(':email',$this->email);
    $stmt->execute();

    $result = $stmt->fetch();

    if(isset($result['Email'])){
        return true;
    } else {
        return false;
    }
}
```

Hace una consulta a la tabla Empleados filtrada por el email introducido en el login, y si la consulta devuelve un resultado devuelve **true**; si no, **false**.

Voy a describir el otro método que uso. Se llama **get\_pass\_in\_db()**, y consulta la contraseña perteneciente al usuario con el email con el que se loguea. Devuelve la contraseña almacenada en la base de datos (recordar que está hasheada).

```
private function get_pass_in_db(){
    $stmt = $this->connectionDB->prepare('SELECT * FROM Empleados WHERE Email=:email');
    $stmt->bindParam(':email',$this->email);
    $stmt->execute();

    $result = $stmt->fetch();

    return $result['Password'];
}
```

Ahora será mas sencillo entender lo que hace el constructor por donde íbamos:

```

if($this->check_email_exists()){
    $passInDB = $this->get_pass_in_db();

    # Compara el hash de la contraseña en la base de datos
    $auth = password_verify($this->password,$passInDB);

    # Si el hash y la contraseña introducida coinciden
    if($auth){
        ob_start();
        session_start();
        $_SESSION['email'] = $this->email;
        $_SESSION['valid'] = true;

        # Redirige a fichero privado
        header('Location: index.html');
    } else {
        # Acceso denegado (contraseña no coincide)
        header('Location: https://bettercallsergio.es/index.html');
    }
} else {
    # Acceso denegado (email no registrado)
    header('Location: https://bettercallsergio.es/index.html');
}

```

Si el email está almacenado en la base de datos, se va a guardar en la variable **passInDB** la contraseña mediante la llamada al método **get\_pass\_in\_db()**; en caso contrario devolvería al usuario a la página principal del frontend. A continuación se guarda en la variable **auth** el resultado de la función **password\_verify()**, que comparará la contraseña en texto plano introducida por el usuario en el login, y la almacenada en la base de datos (el hash). Si éstas coinciden, se almacenará en la variable **true**, y si no, **false**.

Si la variable **auth** contiene **false**, se redirigirá al usuario a la página principal del frontend. En caso contrario (**auth == true**) se van a inicializar los datos de sesión, dándole valores a los campos **email** y **valid** de **\_SESSION**, que contendrán respectivamente el email introducido por el usuario y **true**. Tras esa operación, se redirige al usuario a la página principal del frontend.

Explicaré más tarde el archivo **Register.class.php**.

- **login.php:**

```
<?php
    require_once('./includes/Login.class.php');
    if(isset($_POST['email']) && isset($_POST['password'])){
        $login = new Login($_POST['email'], $_POST['password']);
    } else {
        header('Location: https://bettercallsergio.es/index.html');
    }
?>
```

Este es el archivo que actúa como cerbero en mi aplicación web. Incluye la clase **Login**; si no se le pasa por solicitud **POST** el email y la contraseña, devuelve al usuario a la página principal. En caso afirmativo, crea un objeto de la clase **Login**, por lo que, como hemos visto anteriormente, si pasa todos los filtros de la clase, el usuario terminará en el index.php de la página principal con la sesión iniciada.

Entonces, el usuario vería lo siguiente:



Podemos apreciar que es la misma página de inicio del frontend con la diferencia de que ahora el menú de navegación contiene un botón que lleva a la página de fichar de los empleados, y que el botón de **Área de Empleados** ahora lleva al perfil del empleado

- **perfil.php:**

```
perfil.php > main > sectionwhite > p
<?php
    require_once('./includes/header.php');
?>
<main>
    
    <hr>
    <h2 class="white">Tu información de perfil</h2>
    <hr>
    <section class="white">
        <p>Tu email es <?php echo $_SESSION['email'];?></p>
        <p>Esta información la hemos recuperado usando las sesiones</p>
    </section>
</main>
</body>
</html>
```

MODIFICAR consulta a la base de datos para saber la ruta imagen

- **logout.php:**

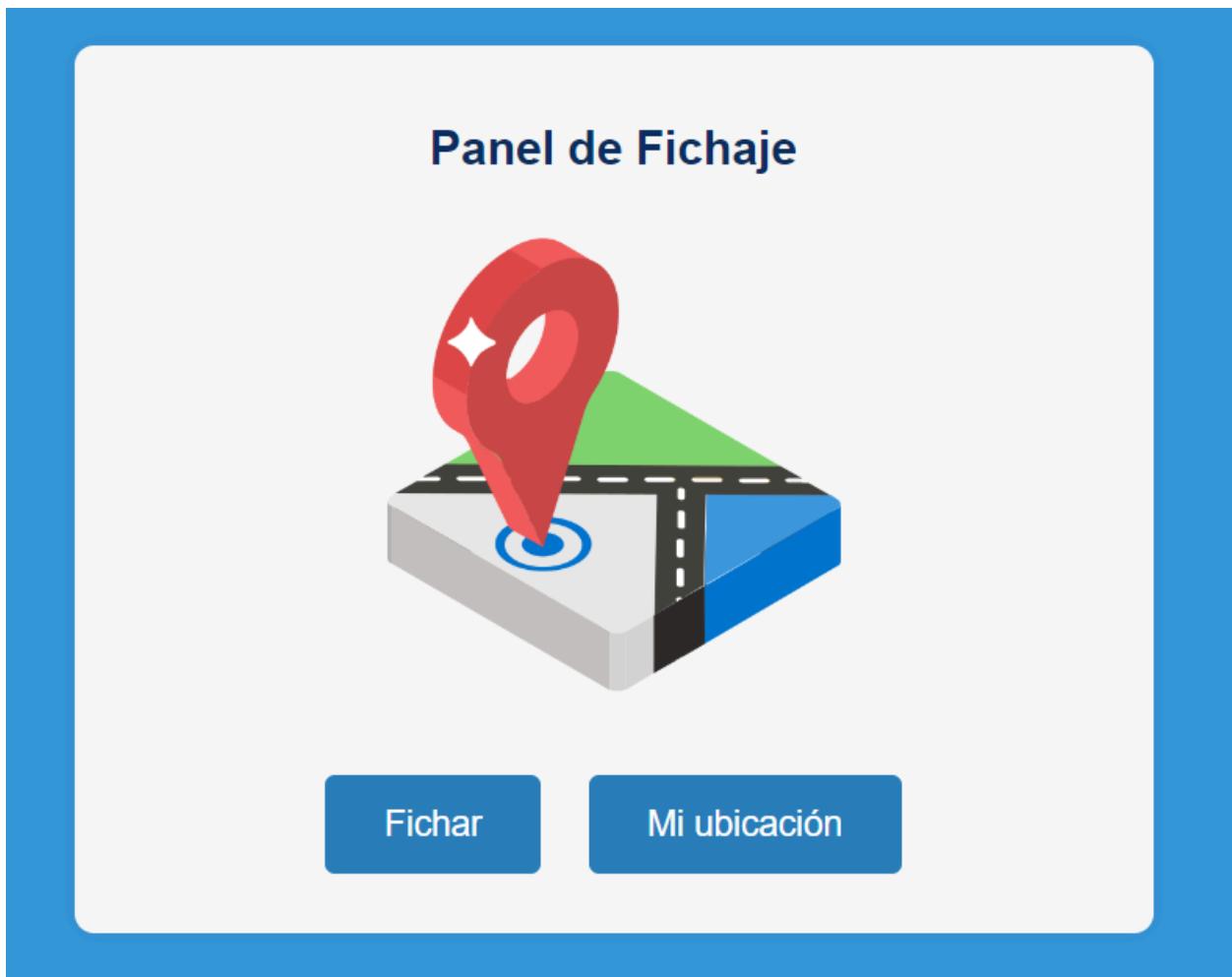
```
<?php
    session_start();
    unset($_SESSION['email']);
    unset($_SESSION['valid']);
    session_destroy();

    header('Location: https://bettercallsergio.es/index.html');

?>
```

Este archivo se encarga de eliminar todos los datos de sesión almacenados y redirigir al usuario a la página de inicio del frontend.

- **fichar.html:**

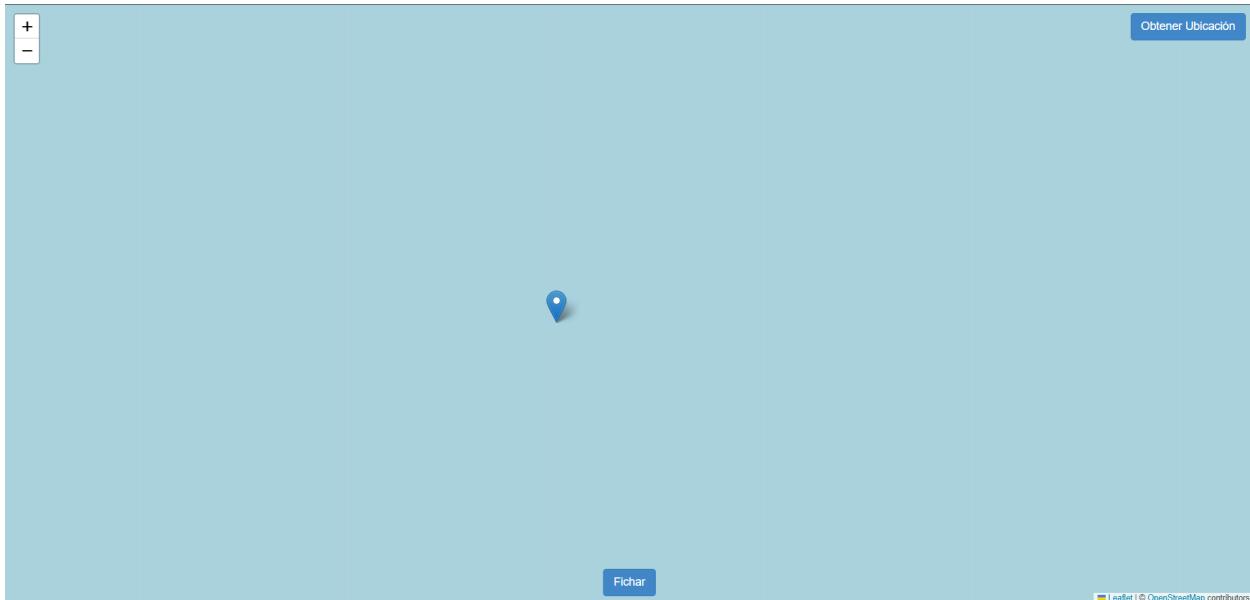


La página es un panel vistoso con 2 botones, en uno el usuario puede consultar su ubicación, y en el otro puede fichar. Los botones llevan a los siguientes archivos:

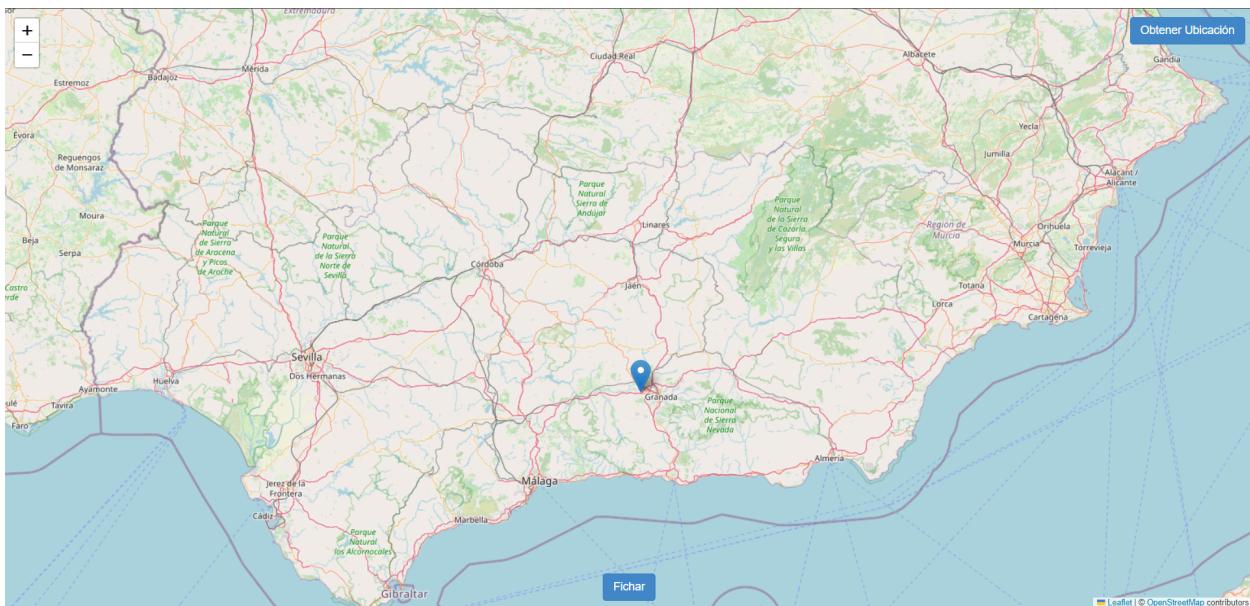
```
<button onclick="location.href='enviar_fichaje.html'">Fichar</button>
<button onclick="location.href='mi_ubicacion_mapa.html'">Mi ubicación</button>
```

## SISTEMA DE AUTENTIFICACIÓN Y GEOLOCALIZACIÓN PARA TRABAJO EN REMOTO

- enviar\_fichaje.html:



En mi caso, tras pulsar el botón se vería la siguiente ubicación. No es necesario pulsar el botón **Obtener Ubicación** pero es más orientativo para el usuario:



Aquí es donde entra la tecnología javascript. El script js se conecta a la api de **openstreetmaps** para obtener la ubicación, si se pulsa el botón obtener ubicación, se centra el cursor sobre la ubicación del usuario, a modo de consulta (no realiza ninguna acción más), pero si se pulsa el botón **fichar**, se envía una solicitud POST al archivo **guardar\_fichaje.php**, que enseñaré a continuación. El archivo **mi\_ubicacion\_mapa.html** tiene el mismo funcionamiento, exceptuando el botón de fichar, solo se puede consultar la ubicación y que se centre el cursor sobre la misma.

Hay que señalar que dependiendo de quien ejecute el script que geolocaliza el dispositivo saldrá una ubicación más o menos precisa, ya que la geolocalización se realiza mediante dirección IP. El script ofrece una ubicación lo más precisa posible con la tecnología que dispongo.

Por su envergadura, el código de éste archivo está en el siguiente enlace de mi repositorio de github:

<https://github.com/sperrio2706/TFG-ASIR>

La página web muestra un mapa interactivo y permite al usuario obtener su ubicación y "fichar" esta ubicación enviándola a un servidor.

En primer lugar, se establecen las configuraciones iniciales y se incluyen varias dependencias esenciales para la funcionalidad y el estilo de la página. Estas dependencias incluyen archivos CSS y JavaScript para **Leaflet**, que es una biblioteca de mapas interactivos; **jQuery**, para facilitar la manipulación del DOM y las operaciones AJAX; **Bootstrap**, para proporcionar estilos y componentes de interfaz de usuario responsivos; y otras bibliotecas como **Font Awesome** para íconos y **Leaflet.AwesomeMarkers** para mejorar los marcadores en el mapa. Además, se establecen algunos estilos básicos para asegurar que el mapa ocupe toda la pantalla y se ajusten adecuadamente los elementos en la página.

En el cuerpo del documento HTML, se estructura la interfaz principal con un div que tiene el ID `map`. Este div ocupa toda la pantalla y se utiliza para mostrar el mapa interactivo. Se incluyen también dos botones: uno con la clase **locate-btn**, que permite al usuario obtener su ubicación actual, y otro dentro de un formulario, con la clase **fichar-btn**, que permite al usuario "fichar" su ubicación. El formulario **fichar-form** contiene campos ocultos (`latitude-input` y `longitude-input`) que almacenarán las coordenadas de la ubicación del usuario y se enviarán al servidor (**guardar\_fichaje.php**) cuando el usuario presione el botón de fichar.

El script de JavaScript incluido en la página configura el mapa utilizando Leaflet. Se inicializa un mapa centrado inicialmente en las coordenadas [0, 0] con un nivel de zoom de 15. A este mapa se le agrega una capa de teselas de OpenStreetMap para mostrar los datos del mapa y se coloca un marcador inicial en el centro del mapa. Además, se define la función **updateMarker**, que actualiza la posición del marcador en el mapa y centra el mapa en las nuevas coordenadas. Esta función también actualiza los campos ocultos del formulario con las coordenadas actuales del usuario.

La función **getLocation** utiliza la API de geolocalización del navegador para obtener la ubicación actual del usuario. Si el navegador soporta la geolocalización y el usuario otorga permiso, esta función obtiene las coordenadas de la ubicación actual y llama a **updateMarker** para actualizar el marcador en el mapa y los campos del formulario. En caso de error o si la geolocalización no es compatible con el navegador, se muestra un mensaje de alerta.

Para gestionar los eventos de los botones, se añaden dos **event listeners**. El primero está asociado al botón **locate-btn** y llama a **getLocation** cuando se hace clic, actualizando así la ubicación del usuario en el mapa. El segundo event listener está asociado al botón **fichar-btn**. Este listener previene el envío inmediato del formulario utilizando **event.preventDefault()**, llama a **getLocation** para obtener la ubicación actual y, tras una breve espera de 500 milisegundos

## SISTEMA DE AUTENTIFICACIÓN Y GEOLOCALIZACIÓN PARA TRABAJO EN REMOTO

(para asegurar que la ubicación se haya actualizado), envía el formulario con las coordenadas actualizadas al servidor.

Lo más interesante es que al pulsar **fichar-btn** se va a mandar una solicitud post al archivo **guardar\_fichaje.php**, que procesa dicha información.

```
<?php

ob_start();
session_start();

# Si la sesión no es válida, redirige al frontend
if (!isset($_SESSION['valid'])) {
    header('Location: https://bettercallsergio.es/index.html');
    exit; // Agrega una salida para asegurarse de que se detenga la
ejecución después de la redirección
}

include 'includes/functions.php';

// Función para obtener el ID de usuario de la sesión actual
function getUserId($conexion)
{
    try {
        $consulta = "SELECT Id_Emppleado FROM Empleados WHERE Email = ?";
        $resultado = $conexion->prepare($consulta);
        $email = $_SESSION["email"];
        $resultado->bindParam(1, $email);
        $resultado->execute();
        // Devuelve el ID del empleado
        return $resultado->fetchColumn();
    } catch (PDOException $e) {
        // Manejo de errores
        echo "Error al obtener el ID del empleado: " . $e->getMessage();
        return false;
    }
}
```

```
}
```

La primera parte del script es similar a la vista en otros archivos; inicializa los datos de sesión y si ésta no es válida, redirige al usuario a la página de inicio del frontend.

Tras ésto, incluye las funciones de **funciones.php**, y añade una más abajo para consultar el Id del usuario asociado a su email.

A continuación, se muestra una función que inserta la ubicación enviada por el archivo **enviar\_fichaje.html**.

```
// Función para insertar la ubicación al fichar
function insert_ubi($conexion, $id_empleado)
{
    $lat = $_POST['latitude'];
    $lon = $_POST['longitude'];

    try {
        $consulta = "INSERT INTO Ubicaciones (Id_Empleado, Coordenadas)
VALUES (:Id, POINT(:Lat,:Lon))";
        $resultado = $conexion->prepare($consulta);
        $resultado->execute(array(
            ':Id' => $id_empleado,
            ':Lat' => $lat,
            ':Lon' => $lon
        ));
    } catch (PDOException $e) {
        // Manejo de errores
        echo "Error al insertar la ubicación: " . $e->getMessage();
        return false;
    }
}
```

Primero almacena en variables la latitud y la longitud, consultadas en la variable **\_POST**. Luego abre una cláusula **try-catch** para manejo de errores, donde, en el try, se guarda en la variable **consulta** el insert into que va a ejecutar. Hay que destacar de dicha consulta que solo vamos a insertar datos manualmente en los campos **Id\_Empleado** y **Coordenadas**, ya que el **Id\_Ubicación** y la **Fecha** son autogenerados (auto increment y timestamp).

Prepara la consulta para su ejecución (el objeto de consulta se le pasa como argumento con la variable **conexion**) y sustituye los campos por las variables con los datos que tenemos. Primero sustituye el **Id\_empleado** por el que tiene en la variable **id\_empleado** pasado como argumento. Finalmente, sustituye la latitud y la longitud y ejecuta la consulta.

Lo siguiente es el cuerpo principal del programa:

```
// Verificar si se han enviado datos por POST
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // Verificar si la variable POST 'latitude' y 'longitude' están
    configuradas
    if (isset($_POST['latitude']) && isset($_POST['longitude'])) {

        $conexion = connectionDB();

        // Consultamos el ID de usuario de nuestra sesión
        $id_empleado = get_userId($conexion);

        if ($id_empleado != null) {

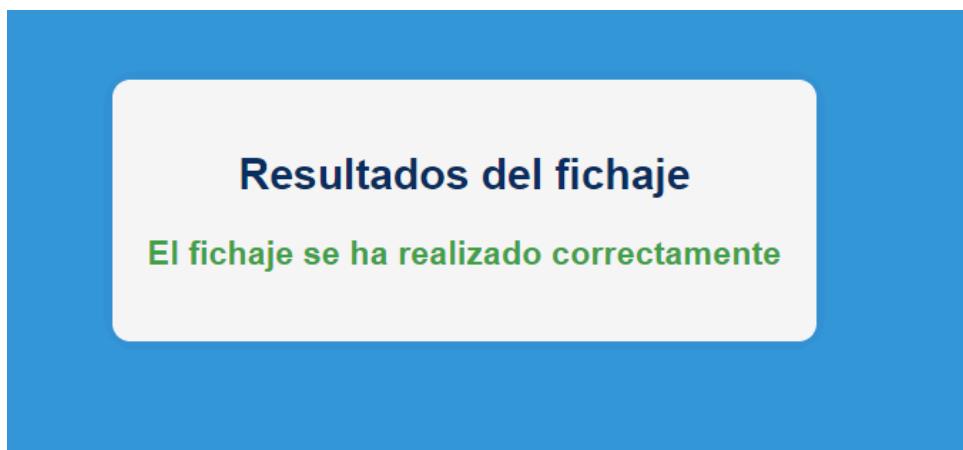
            // Insertamos la latitud y la longitud en la base de datos
            insert_ubi($conexion, $id_empleado);

            // Finalmente mostramos al usuario que el fichaje ha sido
            correcto
            header("Location: fichaje_correcto.html");
            exit; // Agrega una salida para asegurarse de que se detenga
            la ejecución después de la redirección
        } else {
            if ($_SESSION["email"] === null) {
```

```
        echo "Error: No hay un correo electrónico almacenado en  
la sesión.";  
    } else {  
        echo "Error: No se encontró ningún empleado con el  
correo electrónico '{$_SESSION["email"]}'.";  
    }  
}  
} else {  
    echo "Error: Los datos de ubicación no están configurados  
correctamente.";  
}  
}  
  
// Cerrar la conexión a la base de datos después de usarla  
$conexion = null;
```

Si el servidor usa POST como método de solicitud y se ha pasado la latitud y la longitud en la solicitud POST, se crea una conexión a la bbdd y se consulta el id del usuario de la sesión actual.

Luego hay una condición: si el **id\_empleado** devuelto por la función es **null** significa que o no hay email almacenado en la sesión, o éste no está registrado en la bbdd. En caso contrario, va a insertar la latitud y la longitud haciendo uso de la función anteriormente descrita. Tras esto, el usuario va a ser redirigido al archivo **fichaje\_correcto.html**, donde se le mostrará el siguiente mensaje:



Si consultamos la base de datos para comprobar el correcto funcionamiento de la programación, vemos que en efecto, está insertando las ubicaciones de los usuarios.

```
--  
23 •  SELECT * FROM Ubicaciones;  
24  
<[{"id": 3, "id_empleado": "da4b3362-227d-11ef-882c-0800275e5af5", "coordenadas": "BLOB", "fecha": "2024-06-04 21:31:56"}, {"id": 4, "id_empleado": "da4b3362-227d-11ef-882c-0800275e5af5", "coordenadas": "BLOB", "fecha": "2024-06-04 22:49:20"}, {"id": 5, "id_empleado": "da4b3362-227d-11ef-882c-0800275e5af5", "coordenadas": "BLOB", "fecha": "2024-06-04 22:49:23"}, {"id": 6, "id_empleado": "da4b3362-227d-11ef-882c-0800275e5af5", "coordenadas": "BLOB", "fecha": "2024-06-04 22:49:28"}, {"id": 7, "id_empleado": "da4b3362-227d-11ef-882c-0800275e5af5", "coordenadas": "BLOB", "fecha": "2024-06-04 22:55:34"}]
```

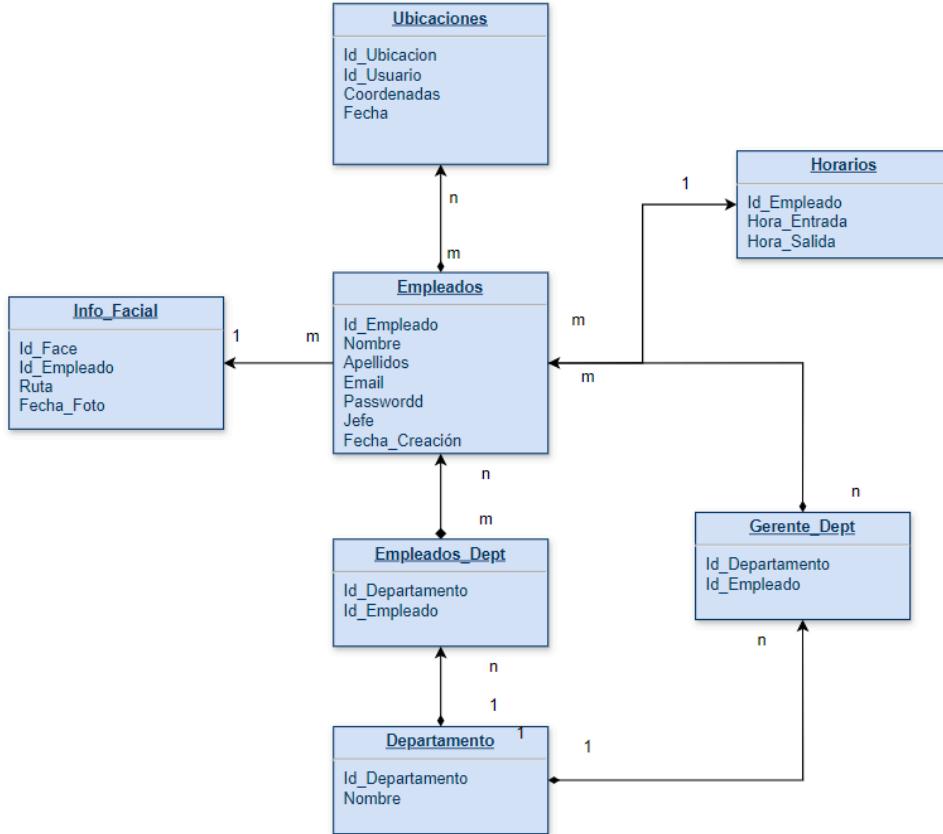
Id_Ubicacion	Id_Empleado	Coordenadas	Fecha
3	da4b3362-227d-11ef-882c-0800275e5af5	BLOB	2024-06-04 21:31:56
4	da4b3362-227d-11ef-882c-0800275e5af5	BLOB	2024-06-04 22:49:20
5	da4b3362-227d-11ef-882c-0800275e5af5	BLOB	2024-06-04 22:49:23
6	da4b3362-227d-11ef-882c-0800275e5af5	BLOB	2024-06-04 22:49:28
7	da4b3362-227d-11ef-882c-0800275e5af5	BLOB	2024-06-04 22:55:34

El funcionamiento es correcto pero hay un problema: el empleado puede fichar cuantas veces quiera. Esto lo voy a arreglar añadiendo una tabla a la base de datos que se llamará **Horarios**. Sus campos serán el **Id\_Empleado**, que será clave primaria, y **hora\_entrada** y **hora\_salida**. Así, se permitirá el fichaje a aquellos empleados que fichen en la hora de entrada con una diferencia de 30 minutos, es decir, el fichaje es posible si fichan desde 30 minutos antes de la hora de entrada hasta 30 minutos después de la misma. Ésto se controlará mediante el archivo **guardar\_fichaje.php**, donde implementaré una función que controlará esto mismo. Para esta cuestión, voy a suponer que todos los empleados trabajan de lunes a viernes de 8:00 a 15:00.

Aprovechando la modificación de la estructura de la base de datos, he añadido que el email de los empleados tenga una restricción **UNIQUE**, para que no hayan varios empleados con el mismo email.

## SISTEMA DE AUTENTIFICACIÓN Y GEOLOCALIZACIÓN PARA TRABAJO EN REMOTO

Añadiendo la nueva tabla, el diagrama de Entidad-Relación de la bbdd luce así:



El código para crearla se ve a continuación:

```

-- Tabla Horarios
CREATE TABLE Horarios (
    Id_Emppleado CHAR(36) NOT NULL,
    Hora_Entrada TIME NOT NULL,
    Hora_Salida TIME NOT NULL,
    FOREIGN KEY (Id_Emppleado) REFERENCES Empleados(Id_Emppleado)
);
  
```

Para automatizar el control del fichaje primero es necesario modificar el archivo **Register.class.php** para que al insertar un nuevo usuario, también se inserte a la tabla horarios su horario, que por defecto será de 8:00 a 15:00.

```
private function add_working_hours(){
    # Consultamos el UUID generado para el usuario
    $stmt = $this->connectionDB->prepare('SELECT * FROM Empleados WHERE Email=:email');
    $stmt->bindParam(':email',$this->email);
    $stmt->execute();

    $result = $stmt->fetch();
    $id = $result['Id_Emppleado'];

    $consulta = "
    INSERT INTO Horarios( ID_Emppleado, Hora_Entrada, Hora_Salida)
    VALUES (
        :id_empleado,
        :hora_entrada,
        :hora_salida
    );
    ";
    $stmt = $this->connectionDB->prepare($consulta);
    $stmt->bindParam(':id_empleado',$id);
    $stmt->bindParam(':hora_entrada','08:00:00');
    $stmt->bindParam(':hora_salida','15:00:00');
    $stmt->execute();
}
```

He añadido esa función para que consulte el id del empleado a registrar e inserte en la tabla horarios su id y los horarios de entrada y salida. Este sistema permite poca flexibilidad a la hora de especificar los horarios de los empleados, pero se podría solucionar si en el formulario de registro preguntamos los horarios de entrada y salida.

Ésta función es llamada después de la llamada a la función **create\_user()** en el constructor de la clase.

```
try{
    # Comprueba si existe el email
    if($this->check_email_exists()){
        $this->result_register = false;
    } else {
        # Si no existe el email crea el usuario
        $this->create_user();
        $this->add_working_hours();
        $this->result_register = true;
    }
} catch(Exception $e){
    die('ERROR: '. $e->getMessage());
}
```

Finalmente, hay que modificar el archivo **guardar\_fichaje.php** para que permita el fichaje si y sólo si al momento de pulsar el botón de fichar la hora del sistema está entre las 7:30 y las 8:30 o las 14:30 y las 15:30, y que además hay que controlar que solo se fiche una vez a la entrada y otra a la salida. Para controlar esta cuestión voy a contar el número de veces que el usuario ha fichado el día actual: si el usuario tiene un solo fichaje registrado el día de hoy, significa que ya ha fichado una vez, por la mañana, así que el nuevo fichaje será el fichaje de salida; si ha fichado 2 veces, significa que ya ha fichado en la entrada y en la salida, y no fichará; y por último, si no hay fichajes registrados el día de hoy, significa que el fichaje que va a hacer el usuario es el de entrada.

Para ésto último, utilizaré la siguiente consulta, que me devolverá el número de veces que el usuario ha fichado hoy. La consulta es la de más abajo, donde sustituiré las comillas a la derecha de **Id\_Empleado**:

```
-- Consulta numero de fichajes
SELECT COUNT(*) FROM Ubicaciones
WHERE DATE(Fecha) = CURDATE() AND Id_Empleado = "";
```

Ésto lo resuelvo con la siguiente función:

```
// Función que comprueba si el fichaje es correcto
function fichaje_posible($conexion, $id_empleado)
{
    // Variable a devolver
    $salida = false;

    // Consultamos los horarios
    $stmt = $conexion->prepare('SELECT * FROM Horarios WHERE Id_Empleado=:id_empleado');
    $stmt->bindParam(':id_empleado', $id_empleado);
    $stmt->execute();

    $result = $stmt->fetch();

    // Creamos un array con los horarios del empleado
    $horarios = [
        'hora_entrada' => $result['Hora_Entrada'],
        'hora_salida' => $result['Hora_Salida']
    ];

    // Obtener la hora actual
    $horaActual = new DateTime();
```

## SISTEMA DE AUTENTIFICACIÓN Y GEOLOCALIZACIÓN PARA TRABAJO EN REMOTO

```
// Consultamos el número de veces que el empleado ha fichado hoy
$stmt = $conexion->prepare('SELECT COUNT(*) AS Num_Fichajes FROM Ubicaciones WHERE DATE(Fecha) =
CURDATE() AND Id_Emppleado=:id_empleado');
$stmt->bindParam(':id_empleado', $id_empleado);
$stmt->execute();

$result = $stmt->fetch();

$numero_fichajes = $result['Num_Fichajes'];

// Controlamos si el usuario ficha o no

if ($numero_fichajes == 0) {
    // Convierte hora_entrada a DateTime
    $horarioEntrada = date('H:i:s', strtotime($horarios['hora_entrada']));

    // Crea intervalos de 30 minutos antes y después de la hora de entrada
    $intervaloAntes = (new DateTime($horarioEntrada))->modify('-30 minutes');
    $intervaloDespues = (new DateTime($horarioEntrada))->modify('+30 minutes');

    // Comprobar si la hora actual está dentro del intervalo
    if ($horaActual >= $intervaloAntes && $horaActual <= $intervaloDespues) {
        $salida = true;
    }
} elseif ($numero_fichajes == 1) {
    // Convierte hora_salida a DateTime
    $horarioSalida = date('H:i:s', strtotime($horarios['hora_salida']));

    // Crea intervalos de 30 minutos antes y después de la hora de salida
    $intervaloAntes = (new DateTime($horarioSalida))->modify('-30 minutes');
    $intervaloDespues = (new DateTime($horarioSalida))->modify('+30 minutes');

    // Comprobar si la hora actual está dentro del intervalo
    if ($horaActual >= $intervaloAntes && $horaActual <= $intervaloDespues) {
        $salida = true;
    }
}

return $salida;
```

Con esta función controlo el fichaje de manera que el usuario solo fichará si está en el intervalo de tiempo permitido por la mañana (media hora antes y después de las 8:00) y el número de fichajes diarios es 0, o por la tarde si se encuentra en el intervalo de tiempo de la

tarde y ha fichado una sola vez. Con cualquiera de estas dos condiciones devolverá **true** y se realizará el fichaje. En caso contrario muestra el mensaje de error siguiente:



La función que menciono se llama en una condición antes de insertar la ubicación en la tabla ubicaciones:

```
if ($id_empleado != null) {  
    if (fichaje_posible($conexion, $id_empleado)){  
        // Insertamos la latitud y la longitud en la base de datos  
        insert_ubi($conexion, $id_empleado);  
  
        // Finalmente mostramos al usuario que el fichaje ha sido correcto  
        header("Location: fichaje_correcto.html");  
    }else{  
        // Ya ha fichado dos veces  
        header('Location: https://bettercallsergio.es:8080/error\_fichaje.html');  
    }  
  
    exit; // Agrega una salida para asegurarse de que se detenga la ejecución después de la redirección  
} else {  
    if ($_SESSION["email"] === null) {  
        echo "Error: No hay un correo electrónico almacenado en la sesión.";  
    } else {  
        echo "Error: No se encontró ningún empleado con el correo electrónico '{$_SESSION["email"]}.'";  
    }  
}
```

## 5. Registro de empleado.

Para el registro del empleado haré uso de un formulario html que pedirá los datos del usuario y enviará una solicitud POST al archivo **register.php**, que hará uso de la clase **Register** del fichero **Register.class.php** que aún no he explicado.

El login se encontrará en un directorio protegido por usuario y contraseña mediante un archivo **.htaccess**. Para restringir el acceso al recurso por usuario y contraseña debemos crear en primer lugar una base de datos en modo texto que nos guarde el usuario y la contraseña que especifiquemos; usaremos el programa htpasswd para crear el usuario admin y especificarle una contraseña:

```
root@ubuntu:/var/www/html/registro# htpasswd -c ./htpasswd admin
New password:
Re-type new password:
Adding password for user admin
```

Si queremos ver dicho archivo, nos mostrará el usuario en texto plano y la contraseña encriptada:

```
root@ubuntu:/var/www/html/registro# cat .htpasswd
admin:$apr1$RC7MNQyM$EKT0qtSv48Z6GoQ6k/8RT0
```

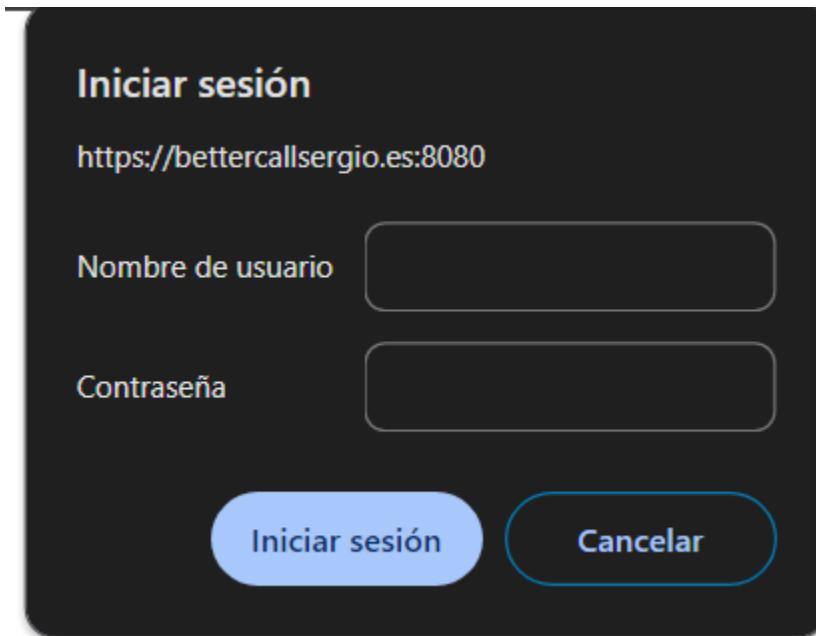
Ahora crearemos el **.htaccess** con las siguientes directivas:

```
AuthType Basic
AuthName "Se requiere usuario y contraseña"
AuthUserFile /var/www/html/registro/.htpasswd
Require user admin
```

Para que funcione el **.htaccess** es necesario cambiar la directiva **AllowOverride** a **AuthConfig** (por defecto es **None**).

```
<Directory /var/www/>
    Options Indexes FollowSymLinks
    AllowOverride AuthConfig
    Require all granted
</Directory>
```

Si accedemos al recurso registro del servidor:



El login se ubicará en el directorio `/var/www/html/registro`, al igual que `registrar.php`, que creará un objeto de la clase **Register**. El formulario se ve tal que así:

**Registro de usuarios**

Bajo autorización

Nombre:  
Introduzca su nombre

Apellidos:  
Introduzca sus apellidos

Email:  
Introduzca su correo electrónico

Contraseña:  
Introduzca su contraseña

¿Eres jefe?:  
Selecione una opción ▾

Enviar

El archivo encargado de procesar el formulario, `registrar.php`, crea un objeto de la clase **Register**, como ya había mencionado, utilizando todas las entradas de `$_POST` en el constructor de la clase. En caso de fallo de la solicitud `_POST`, muestra un mensaje de error:

```
<?php
require_once('../includes/Register.class.php');
if(isset($_POST['name']) && isset($_POST['surname']) && isset($_POST['email']) && isset($_POST['password'])){
    $register = new Register($_POST['name'], $_POST['surname'], $_POST['email'], $_POST['password'], $_POST['is_boss']);
    $resultado = $register->get_confirmation();
} else {
    echo "<h1>ERROR</h1>";
    echo "<h3>La solicitud POST no ha sido enviada correctamente</h3>";
}

# una vez registrado redirigimos a registro_correcto.html
if ($resultado){
    header('Location: https://bettercallsergio.es:8080/registro/registrar_correcto.html');
} else {
    echo "<h1>ERROR</h1>";
    echo "<h3>El usuario ya existe en el sistema</h3>";
}

?>
```

Al crearse un objeto de la clase **Register**, se ejecuta el constructor de la clase, que es el siguiente:

```
require_once('functions.php');

# Clase que registra a un usuario en la base de datos en caso de que no exista

class Register{

    private $nombre;
    private $apellidos;
    private $email;
    private $password;
    private $jefe;
    private $connectionDB;
    private $result_register;

    public function __construct($nombre, $apellidos, $email, $password, $jefe){
        $this->nombre = secure_data($nombre);
        $this->apellidos = secure_data($apellidos);
        $this->email = secure_data($email);
        $this->password = secure_data($password);
        $this->password = hash_password($this->password);
        $this->jefe = $jefe;
        $this->connectionDB = connectionDB();

        try{
            # Comprueba si existe el email
            if($this->check_email_exists()){
                $this->result_register = false;
            } else {
                # Si no existe el email crea el usuario
                $this->create_user();
                $this->result_register = true;
            }
        } catch(Exception $e){
            die('ERROR: '. $e->getMessage());
        }
    }
}
```

Tiene tantos atributos como campos tiene la tabla empleados, exceptuando la fecha de creación, que es de tipo **TIMESTAMP**, y se añade automáticamente a la fila. A constructor le pasamos por parámetro todos los campos de la solicitud POST, cuyos valores van a ser filtrados con la función **secure\_data()** vista anteriormente, antes de ser almacenados en los atributos de la clase. La contraseña introducida va a convertirse a su hash y se va a crear una conexión a la base de datos.

Luego comprueba si el email del usuario figura en la base de datos, si existe asigna el valor **false** al atributo **result\_register**, y en caso contrario, llama al método **create\_user()**, que simplemente va a insertar los datos del usuario a la tabla Empleados.

Cuando el registro se completa sin problemas, en **registrar.php** se va a hacer una comprobación llamando al método **get\_confirmation()**, que devolverá **true** en caso de que si se haya registrado correctamente. Además, en dicho caso, mediante el condicional del final, redirigirá al usuario a la página **registro\_correcto.html**.

Ya tenemos automatizado el registro de empleados en la empresa. así que vamos a probar a registrar al usuario pepe. Lo primero será entrar al recurso donde tengo el formulario de ingreso, introducir el usuario y la contraseña que solicitará el **.htaccess** y llenar los campos. El formulario llenado figura tal que así:

The screenshot shows a registration form titled "Registro de usuarios". The form is labeled "Bajo autorización". It contains four text input fields: "Nombre" (Name) with "Pepe", "Apellidos" (Last Name) with "González Adarve", "Email" with "pepe023@gmail.com", and "Contraseña" (Password) with a masked value. Below these is a dropdown menu for "¿Eres jefe?" (Are you a manager?) with "No" selected. At the bottom is a blue "Enviar" (Send) button.

<b>Registro de usuarios</b>	
Bajo autorización	
Nombre:	Pepe
Apellidos:	González Adarve
Email:	pepe023@gmail.com
Contraseña:	.....
¿Eres jefe?:	No
<b>Enviar</b>	

Si pulsamos enviar, nos redirige a la página **registro\_correcto.html**, por lo que parece que el usuario se ha insertado correctamente:



Para comprobarlo voy a conectarme a la base de datos y hacer una consulta que me saque el usuario cuyo nombre es Pepe, ya que de momento solo tengo un usuario con dicho nombre.

```
15 •  SELECT * FROM Empleados
16 WHERE Nombre = 'Pepe';
17
```

Result Grid | Filter Rows: [ ] | Edit: [ ] | Export/Import: [ ] | Wrap Cell Content: [ ]

Id_Empelado	Nombre	Apellidos	Email	Passwordd	Jefe	Fecha_Creadion
58bafae1-2334-11ef-8442-0800275e5af5	Pepe	González Adarve	pepe023@gmail.com	\$2y\$10\$oFJhOGdR4Ce1SkSUR/70g.BV.dXhPTn...	N	2024-06-05 14:08:40
*	HULL	HULL	HULL	HULL	HULL	HULL

## 6. Panel administrativo.

Los jefes van a poder consultar la lista de empleados que han fichado cada día. Para ello he creado una función que detecta si el usuario logueado consultando a la base de datos si el usuario tiene el valor **S** en el campo **Jefe** de la tabla **Empleados**. La función es la siguiente:

```
# Función que comprueba si el usuario es jefe

function es_jefe($email){
    $conexion = connectionDB();
    $stmt = $conexion->prepare('SELECT * FROM Empleados WHERE Email=:email');
    $stmt->bindParam(':email',$email);
    $stmt->execute();

    $result = $stmt->fetch();

    if($result['Jefe'] == 'S'){
        return true;
    } else {
        return false;
    }
}
```

Ahora modificaremos la clase login para que cuando loguee con éxito al usuario, compruebe si es Jefe, y si lo es, lo mandará a la página **index-jefes.html**, que será la página principal de los mismo, similar a la página **index.html** a la que ingresarán los que no sean jefes, pero con la diferencia de que éstos tendrán un botón más en el menú de navegación, que les llevará al panel administrativo. Dicha modificación es la siguiente:

```
# SI el hash y la contraseña introducida coinciden
if($auth){
    ob_start();
    session_start();
    $_SESSION['email'] = $this->email;
    $_SESSION['valid'] = true;

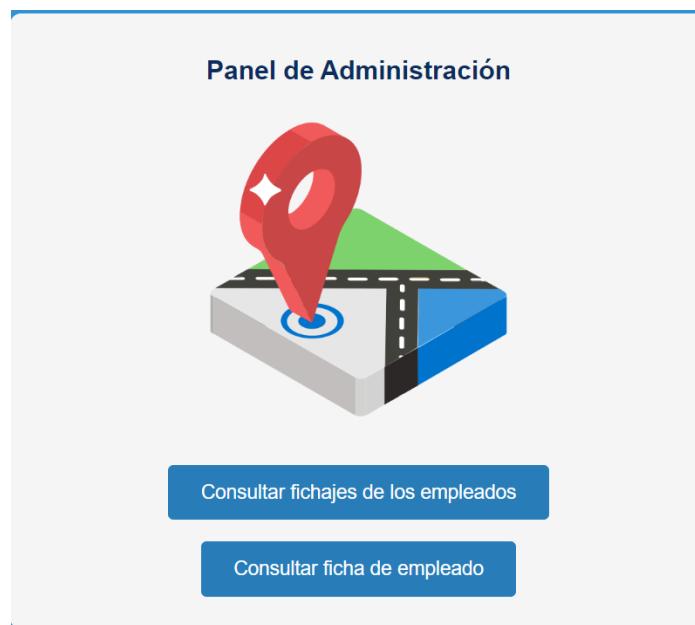
    if (es_jefe($this->email)){
        header('Location: index-jefes.html');
    }else{
        # Redirige a fichero privado
        header('Location: index.html');
    }
}
```

En el constructor de la clase he añadido que, si la autentificación es correcta, llame a la función **es\_jefe()**. Ésta, si devuelve **true**, va a redirigir a los usuarios a la página **index-jefes.html**, solo accesible por jefes, mientras que si devuelve **false**, significa que el usuario no es un jefe y será redirigido al área común. Si un jefe se loguea correctamente verá la página de más abajo, en la que estará disponible un nuevo botón en el menú de navegación: el **panel de administración**.



## SISTEMA DE AUTENTIFICACIÓN Y GEOLOCALIZACIÓN PARA TRABAJO EN REMOTO

Si el usuario pulsa sobre el panel de administración le lleva a la siguiente página:



Tiene dos botones, el primero le lleva a un archivo.php que consulta todas las fechas en las que los empleados han fichado, y donde puede ver qué empleados han fichado ese día. Se ve como a continuación:

Fecha	Nº empleados fichados	Día de la semana
2024-06-04	1	1

El código de éste archivo es el que se muestra a continuación:

```
require_once('./includes/functions.php');

function insertar_fila($fecha, $num_empleados, $dia_semana)
{
    echo '<form action="consultar_fichajes_procesador.php" method="post">';
    echo '<input type="hidden" name="fecha" value="' . $fecha . '">';
    echo '<tr><td><button type="submit">' . $fecha . '</button></td><td>' . $num_empleados . '</td><td>' . $dia_semana . '</td></tr>';
    echo '</form>';
}
```

En la primera imagen se muestra que incluimos las funciones ya utilizadas en script anteriores, y una nueva función llamada **insertar\_fila()**, cuyo único uso es facilitar la visualización del código a la hora de insertar filas a la tabla. Acepta como argumento los valores que va a introducir en los campos de la tabla. Hay que destacar que como la fecha es un botón que manda la propia fecha mediante una solicitud POST al archivo **consultar\_fichajes\_procesador.php**, está en un formulario con un **input** de tipo **hidden** de manera que la fecha se muestra en el propio botón.

```
function obtenerFechasDistintas($conexion)
{
    try {
        // Consulta las fechas distintas
        $stmt = $conexion->prepare('SELECT DISTINCT DATE(Fecha) AS Fecha FROM Ubicaciones');
        $stmt->execute();

        // Inicializa un array para almacenar los resultados
        $fechas = [];

        $tuplas = $stmt->fetchAll();

        $contador = 0;
        // Recorre los resultados y los almacena en el array
        foreach ($tuplas as $tupla) {
            $fechas[$contador] = $tupla['Fecha'];
            $contador++;
        }

        // Devuelve el array de fechas
        return $fechas;
    } catch (PDOException $e) {
        // Manejo de errores
        echo "Error al obtener fechas distintas: " . $e->getMessage();
        return false; // Otra opción: lanzar una nueva excepción o devolver un valor específico según el caso
    }
}
```

La función **obtenerFechasDistintas()** hace una consulta a la base de datos para conseguir las fechas sin repetir en que los empleados han fichado. Las filas que devuelva dicha consulta, las va a volcar en un array y va a devolver dicho array.

```
// Función que obtiene el numero de empleados que ficharon un dia concreto

function obtener_numero_empleados($conexion, $fecha)
{
    // Consulta las fechas distintas
    $sql = "SELECT COUNT(DISTINCT E.Id_Emppleado) AS Num_Employados
FROM Empleados E
INNER JOIN Ubicaciones U ON E.Id_Emppleado = U.Id_Emppleado
WHERE DATE(U.Fecha) = :fecha";
    $stmt = $conexion->prepare($sql);
    $stmt->bindParam(':fecha', $fecha);
    $stmt->execute();

    // Devuelve el array de fechas
    $result = $stmt->fetch(PDO::FETCH_ASSOC);

    return $result['Num_Employados'];
}
```

La función **obtener\_numero\_empleados()** acepta como argumento un objeto PDO una fecha. Va a consultar a la base de datos el número de empleados que ficharon en la fecha pasada como argumento, y va a devolver dicho valor.

```

$conexion = connectionDB();

// Sacamos todas las fechas sin repetir en que han habido fichajes
$fechas = obtenerFechasDistintas($conexion);

// Recorre las fechas y llama a la función insertar_fila
foreach ($fechas as $fecha) {
    // Aquí puedes calcular $num_empleados y $dia_semana si es necesario
    $num_empleados = obtener_numero_empleados($conexion, $fecha);
    $dia_semana = date('1', strtotime($fecha));

    insertar_fila($fecha, $num_empleados, $dia_semana);
}

```

Finalmente, tenemos lo que realmente va a hacer el programa. Va a crear un objeto de tipo **PDO** mediante la llamada a la función **connectionDB()**, luego va a almacenar las fechas en que ficharon los empleados en una variable, y va a recorrerlas consultando el número de empleados que ficharon cada día mediante llamadas a la función **obtener\_numero\_empleados()**. Por último, va a obtener qué día de la semana es y va a insertar la fecha, el número de empleados y el día de la semana en una fila de la tabla.

Con los datos que actualmente tiene mi base de datos, solo me muestra una fila, de un solo fichaje de un solo empleado. Las fechas son un botón que lleva a una nueva página que consulta todos los empleados que ficharon en dicha fecha. Ésto es posible porque al pulsar el botón de la fecha, se manda una solicitud post al archivo **consultar\_fichajes\_procesador.php** con la fecha como información, y en dicho archivo se consultan los empleados sin repetir que ficharon ese día. La nueva página se ve así:

Nombre	Apellidos	Email	Jefe	Ubicacion
Sergio	Pérez Ríos	sperrio2706@ieszaidinvergeles.org	S	Lat: -3.7413464, Lon: 37.1965381 <a href="#">Ver Ubicacion</a>

Y el código es el siguiente:

```
require_once('./includes/functions.php');

function insertar_fila($nombre, $apellidos, $email, $jefe, $longitudes, $latitudes)
{
    $longitudes = explode(' ', $longitudes);
    $latitudes = explode(' ', $latitudes);

    echo '<tr>';
    echo '<td>' . htmlspecialchars($nombre) . '</td>';
    echo '<td>' . htmlspecialchars($apellidos) . '</td>';
    echo '<td>' . htmlspecialchars($email) . '</td>';
    echo '<td>' . htmlspecialchars($jefe) . '</td>';
    echo '<td>';
    echo '<form action="get_user_ubi.php" method="post">';
    echo '<input type="hidden" name="email" value="" . htmlspecialchars($email) . '"';
    echo '<select name="ubicacion">';
    for ($i = 0; $i < count($longitudes); $i++) {
        $latitud = htmlspecialchars($latitudes[$i]);
        $longitud = htmlspecialchars($longitudes[$i]);
        echo "<option value='$latitud,$longitud'>Lat: $latitud, Lon: $longitud</option>";
    }
    echo '</select>';
    echo '<button type="submit">Ver Ubicacion</button>';
    echo '</form>';
    echo '</td>';
    echo '</tr>';
}
```

Es la función **insertar\_fila()** usada anteriormente con ligeras modificaciones. Ahora acepta como argumento el nombre, los apellidos, el email, si es jefe o no, y las coordenadas separadas en latitud y longitud. Su funcionamiento es igual que la del anterior script, va a insertar filas en la tabla según los campos de ésta. La gran diferencia es que el formulario ahora manda el email y la ubicación, mediante un desplegable en el podemos elegir las ubicaciones de los fichajes hechos ese día (serán 2 máx).

```

function obtener_datos_empleado($conexion)
{
    $fecha = $_POST['fecha']; // Obtener la fecha de la solicitud POST
    $sql = "
SELECT E.Nombre, E.Apellidos, E.Email, E.Jefe,
       GROUP_CONCAT(ST_X(U.Coordenadas) SEPARATOR ';' ) AS Longitudes,
       GROUP_CONCAT(ST_Y(U.Coordenadas) SEPARATOR ';' ) AS Latitudes
FROM Empleados E
INNER JOIN Ubicaciones U ON E.Id_Empleado = U.Id_Empleado
WHERE DATE(U.Fecha) = ?
GROUP BY E.Id_Empleado, E.Nombre, E.Apellidos, E.Email, E.Jefe;
";
}

$stmt = $conexion->prepare($sql);
$stmt->execute([$fecha]);
$resultado = $stmt->fetchAll(PDO::FETCH_ASSOC);
return $resultado;
}

```

La función **obtener\_datos\_empleado()** es similar a la del script anterior, con la diferencia de que la consulta es más compleja; consulta (sin repetir) nombre, apellidos, email, si es jefe, y todas las latitudes y la longitudes de todos los fichajes hechos en esa fecha concreta concatenadas por el carácter “;”. Luego, devuelve los resultados de la consulta.

```

$conexion = connectionDB();
$datos_empleado = obtener_datos_empleado($conexion);

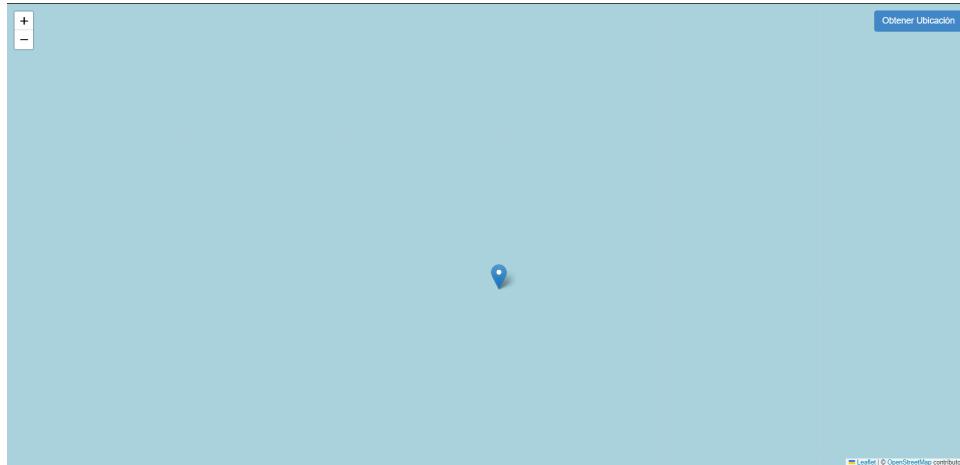
foreach ($datos_empleado as $datos) {
    insertar_fila($datos['Nombre'], $datos['Apellidos'], $datos['Email'], $datos['Jefe'], $datos['Longitudes'], $datos['Latitudes']);
}
?>

```

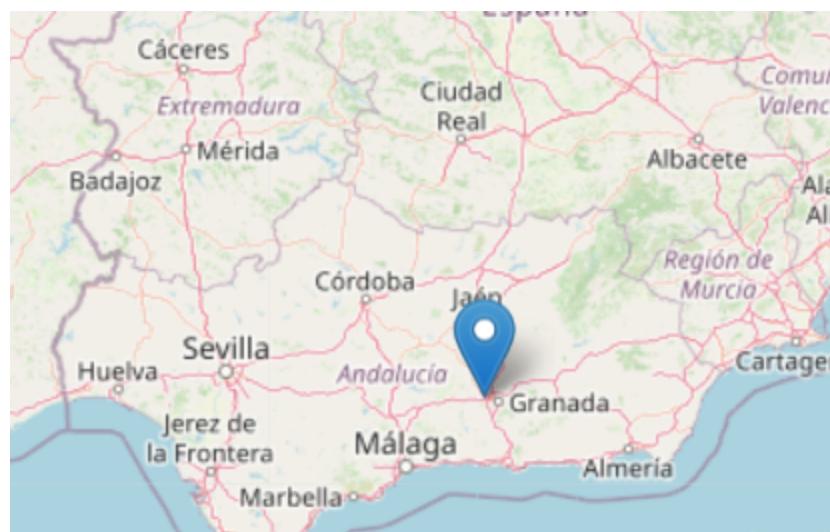
Finalmente, en el código principal, se crea la conexión a la base de datos y se almacenan los datos de los empleados mediante una llamada a la función **obtener\_datos\_empleados()**. Tras esto, recorre las filas devueltas por la consulta de los datos de los empleados y hace llamadas a la función **insertar\_fila()** explicada anteriormente.

## SISTEMA DE AUTENTIFICACIÓN Y GEOLOCALIZACIÓN PARA TRABAJO EN REMOTO

Si le damos a **Ver ubicación** nos mandará a una nueva página similar a la que nos lleva el panel de fichaje, un mapa con un botón para obtener la ubicación, pero con la diferencia de que al pulsar dicho botón nos centrará el cursor sobre la ubicación enviada por el formulario.



Si pulsamos el botón **Obtener ubicación** podemos observar que nos centra en la ubicación en la que el usuario hizo el fichaje:



## SISTEMA DE AUTENTIFICACIÓN Y GEOLOCALIZACIÓN PARA TRABAJO EN REMOTO

El código de la página-mapa es el siguiente, el archivo **get\_user\_ubi.php**:

```
<script>
    var map = L.map("map", {
        center: [0, 0],
        zoom: 15
    });

    L.tileLayer('https://(s).tile.openstreetmap.org/{z}/{x}/{y}.png', {
        attribution: '&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a>
contributors'
    }).addTo(map);

    // Agregar un marcador predeterminado
    var marker = L.marker([0, 0]).addTo(map);

    // Función para actualizar la ubicación del marcador
    function updateMarker(lat, lon) {
        marker.setLatLng([lat, lon]);
        map.setView([lat, lon], 15);
        document.getElementById("latitude-input").value = lat;
        document.getElementById("longitude-input").value = lon;
    }

    // Función para obtener la ubicación actual del usuario
    function getLocation() {
        var ubicacion = "<?php echo isset($_POST['ubicacion']) ? $_POST['ubicacion'] : ''; ?>";

        if (ubicacion) {
            var coordenadas = ubicacion.split(",");
            var longitude = parseFloat(coordenadas[0]); // Longitud primero
            var latitude = parseFloat(coordenadas[1]); // Latitud luego

            if (!isNaN(latitude) && !isNaN(longitude)) {
                updateMarker(latitude, longitude);
            } else {
                alert("No se pudo obtener la ubicación.");
            }
        } else {
            alert("No se recibieron coordenadas de ubicación.");
        }
    }

    document.querySelector('.locate-btn').addEventListener('click', getLocation);
    document.querySelector('.fichar-btn').addEventListener('click', function() {

        // Enseña la ubicación
        getLocation();

    });
</script>
```

El código es similar al usado para el fichaje, con la diferencia de que la función **GetLocation()** se almacena la ubicación enviada desde el formulario del fichero **consultar\_fichajes\_procesador.php**. Si la variable **ubicación** está vacía se alerta sobre ello; en caso contrario va a separar la latitud y la longitud (que venían en **\$\_POST** en formato “latitud,longitud”) y las va a almacenar en variables. Si éstas nuevas variables no son nulas, va a actualizar el cursor sobre la nueva ubicación.

Volviendo al otro botón del panel administrativo (consultar ficha de empleado), al pulsarlo llevaría al usuario a la visualización de una consulta como la siguiente:

Lista de empleados					
Nombre	Apellidos	Email	Jefe	Ver perfil	
Pepe	González Adarve	pepe023@gmail.com	N	<a href="#">Pulsa aquí</a>	
Manolo	Fernandez	manolofndez@gmail.com	N	<a href="#">Pulsa aquí</a>	
Vinicio	Junior	viniciusjunior@gmail.com	N	<a href="#">Pulsa aquí</a>	
Sergio	Pérez Ríos	sperrio2706@ieszaidinvergeles.org	S	<a href="#">Pulsa aquí</a>	
Juan	Izquierdo	juanizquierdo@gmail.com	N	<a href="#">Pulsa aquí</a>	
Ismael	Peregrina	ismaelperegrina@gmail.com	N	<a href="#">Pulsa aquí</a>	
Francisco	Verde	franciscoverde@gmail.com	N	<a href="#">Pulsa aquí</a>	

Pulsar en cualquiera de los diferentes botones de la columna **Ver perfil** llevaría al usuario a una página en la que puede ver información detallada del usuario, como su foto de perfil. El código de ésta página es el siguiente:

```
<?php
    require_once('../includes/functions.php');

    function insertar_fila($nombre, $apellidos, $email, $jefe)
    {
```

```

        echo '<tr>';
        echo '<td>' . htmlspecialchars($nombre) . '</td>';
        echo '<td>' . htmlspecialchars($apellidos) . '</td>';
        echo '<td>' . htmlspecialchars($email) . '</td>';
        echo '<td>' . htmlspecialchars($jefe) . '</td>';
        echo '<td>';
        echo '<form action="get_user_info.php" method="post">';
        echo '<input type="hidden" name="email" value="" .';
        htmlspecialchars($email) . '">';
        echo '<button type="submit">Pulsa aquí</button>';
        echo '</form>';
        echo '</td>';
        echo '</tr>';
    }

    function obtener_datos_empleado($conexion)
    {
        $sql = "
SELECT E.Nombre, E.Apellidos, E.Email, E.Jefe
FROM Empleados E
GROUP BY E.Id_Empleado, E.Nombre, E.Apellidos, E.Email, E.Jefe;
";

        $stmt = $conexion->prepare($sql);
        $stmt->execute();
        $resultado = $stmt->fetchAll(PDO::FETCH_ASSOC);
        return $resultado;
    }

    $conexion = connectionDB();
    $datos_empleado = obtener_datos_empleado($conexion);

    foreach ($datos_empleado as $datos) {
        insertar_fila($datos['Nombre'], $datos['Apellidos'], $datos['Email'],
$datos['Jefe']);
    }
    ?>

```

El código va insertando filas en la tabla de la página mediante la función **insertar\_fila()**, que es similar a las versiones anteriores utilizadas de la misma, aunque ahora está modificada para que en la última columna mande el email de usuario sobre el que se pulsa el botón del formulario. Dicho formulario lleva la información al archivo **get\_user\_info.php**.

El archivo al que redirige **consultar\_ficha\_empleados.php** trabaja con fotos de los empleados, así que para poder acceder a ellas en cualquier momento he almacenado en la tabla **Info\_Facial** de la base de datos la ruta en que se encuentran las fotos de los empleados. Ésta cuestión la he hecho con el siguiente script php:

```
<?php
require('./includes/functions.php');

function removeAccents($string) {
    $tildes = [
        'á' => 'a', 'é' => 'e', 'í' => 'i', 'ó' => 'o', 'ú' => 'u',
        'Á' => 'A', 'É' => 'E', 'Í' => 'I', 'Ó' => 'O', 'Ú' => 'U',
        'ñ' => 'n', 'Ñ' => 'N'
    ];
    // strtr cambia carateres con tilde almacenados en el array
    return strtr($string, $tildes);
}

function generateImagePath($nombre, $apellidos) {
    $nombre = removeAccents(strtolower(str_replace(' ', '', $nombre)));
    $apellidos = removeAccents(strtolower(str_replace(' ', '', $apellidos)));
    return "/var/www/html/images/user-images/{$nombre}{$apellidos}/perfil.jpg";
}

function actualizarInfoFacial($conexion) {
    try {
        // Consulta los datos de la tabla Empleados
        $sql = 'SELECT Id_Emppleado, Nombre, Apellidos FROM Empleados';
        $stmt = $conexion->prepare($sql);
        $stmt->execute();

        // Recorre los resultados
        while ($fila = $stmt->fetch(PDO::FETCH_ASSOC)) {
            $id_empleado = $fila['Id_Emppleado'];
            $nombre = $fila['Nombre'];
            $apellidos = $fila['Apellidos'];

            // Generar la ruta de la imagen
            $ruta_imagen = generateImagePath($nombre, $apellidos);

            // Insertar en la tabla Info_Facial
            $sql_insert = 'INSERT INTO Info_Facial (Id_Emppleado, Ruta) VALUES (:id_empleado,
:ruta_imagen)';
    }
}
```

## SISTEMA DE AUTENTIFICACIÓN Y GEOLOCALIZACIÓN PARA TRABAJO EN REMOTO

```
$stmt_insert = $conexion->prepare($sql_insert);
$stmt_insert->bindParam(':id_empleado', $id_empleado);
$stmt_insert->bindParam(':ruta_imagen', $ruta_imagen);
$stmt_insert->execute();
}
} catch (PDOException $e) {
    // Manejo de errores
    echo "Error al insertar datos: " . $e->getMessage();
}
}

$conexion = connectionDB();
actualizarInfoFacial($conexion);
?>
```

El script recorre los resultados de la consulta a la tabla empleados; para cada fila va a llamar a la función **generateImagePath()**, que devuelve la ruta donde está la foto de perfil del usuario en formato cadena, sustituyendo los caracteres con acento con los mismos caracteres sin acento, y eliminando espacios y mayúsculas. Además, va a insertar en la tabla **Info\_Facial** los datos **Id\_Emppleado** y **Ruta** únicamente, ya que **Id\_Face** y **Fecha\_Foto** son autogenerados.

Si consultamos la tabla **Info\_Facial**, vemos que las rutas de los perfiles de los usuarios se han añadido correctamente:

14 • SELECT * FROM Info_Facial;				
<a href="#">Result Grid</a>   <a href="#">Filter Rows</a> : <input type="text"/>   Edit: <a href="#">Edit</a> <a href="#">Insert</a> <a href="#">Update</a> <a href="#">Delete</a>   Export/Import: <a href="#">CSV</a> <a href="#">Excel</a>   Wrap Cell Content: <a href="#">A</a>				
	<a href="#">Id_Face</a>	<a href="#">Id_Emppleado</a>	<a href="#">Ruta</a>	<a href="#">Fecha_Foto</a>
▶	1	58bafae1-2334-11ef-8442-0800275e5af5	/var/www/html/images/user-images/pepegonzalezadarve/perfil.jpg	2024-06-06 20:35:46
	2	c7a454a4-235d-11ef-8442-0800275e5af5	/var/www/html/images/user-images/manolofernandez/perfil.jpg	2024-06-06 20:35:46
	3	c7caba35-2401-11ef-a629-0800275e5af5	/var/www/html/images/user-images/viniciusjunior/perfil.jpg	2024-06-06 20:35:46
	4	da4b3362-227d-11ef-882c-0800275e5af5	/var/www/html/images/user-images/sergioperezrios/perfil.jpg	2024-06-06 20:35:46
	5	f0474898-23fd-11ef-a629-0800275e5af5	/var/www/html/images/user-images/juanizquierdo/perfil.jpg	2024-06-06 20:35:46

Volviendo al archivo **get\_user\_info.php**, con el que podemos consultar información detallada del empleado. Si pulsamos en ver el usuario Sergio, nos lleva a la página que figura más abajo:

The screenshot shows a web page titled "Perfil detallado" (Detailed Profile) with the subtitle "Página de empresa para TFG de ASIR". Below this, there is a table with the following data:

Foto de perfil	Nombre	Apellidos	Email	Jefe
	Sergio	Pérez Ríos	sperrio2706@ieszaidinvergeles.org	S

El código del archivo es el siguiente:

```
require_once('../includes/functions.php');

function insertar_fila($ruta_foto, $nombre, $apellidos, $email, $jefe)
{
    // Transformamos la ruta absoluta a relativa
    $ruta_relativa = str_replace("/var/www/html",".",$ruta_foto);

    echo '<tr>';
    echo '<td>';
    echo '';
    echo '</td>';
    echo '<td>' . htmlspecialchars($nombre) . '</td>';
    echo '<td>' . htmlspecialchars($apellidos) . '</td>';
    echo '<td>' . htmlspecialchars($email) . '</td>';
    echo '<td>' . htmlspecialchars($jefe) . '</td>';
    echo '</tr>';
}
```

```

function obtener_datos_empleado($conexion)
{
    $sql = "
SELECT E.Id_Emppleado, E.Nombre, E.Apellidos, E.Email, E.Jefe
FROM Empleados E
WHERE E.Email=:email
GROUP BY E.Id_Emppleado, E.Nombre, E.Apellidos, E.Email, E.Jefe;
";

$stmt = $conexion->prepare($sql);
$stmt->bindParam(':email', $_POST['email']);
$stmt->execute();
$resultado = $stmt->fetchAll(PDO::FETCH_ASSOC);
return $resultado;
}

function obtenerRutaImagen($conexion, $id)
{
    try {
        // Consulta la ruta de la imagen para el ID dado
        $sql = 'SELECT Ruta FROM Info_Facial WHERE Id_Emppleado = :id';
        $stmt = $conexion->prepare($sql);
        $stmt->bindParam(':id', $id);
        $stmt->execute();

        // Obtiene la ruta (si existe)
        $ruta = $stmt->fetchColumn();

        // Devuelve la ruta
        return $ruta;
    } catch (PDOException $e) {
        // Manejo de errores
        echo "Error al consultar la ruta de la imagen: " . $e->getMessage();
        return null; // Devuelve null en caso de error
    }
}

$conexion = connectionDB();
$datos_empleado = obtener_datos_empleado($conexion);

foreach ($datos_empleado as $datos) {
    // asignamos la ruta de la foto de perfil
    $ruta_foto = obtenerRutaImagen($conexion, $datos["Id_Emppleado"]);
    insertar_fila($ruta_foto, $datos['Nombre'], $datos['Apellidos'],

```

```
$datos['Email'], $datos['Jefe']);  
}  
?>
```

Podemos ver que el código es similar al anterior. Pero a la hora de insertar filas en la función **insertar\_filas()**, que ahora pide como argumento la ruta en que se encuentra la foto de perfil, modifica la ruta (que al consultarla en la base de datos viene como ruta absoluta), a relativa, ya que hay algún problema al poner rutas relativas en el atributo **src** de la etiqueta **img**, y no se muestran. Por ello, la función **str\_replace()**, predeterminada de php, reemplaza la cadena “/var/www/html” por “.”, de manera que la ruta resultante sería “./images/user-images/nombreyapellidos/perfil.jpg”.

## 7. Programación del login con reconocimiento facial para jefes.

Los jefes tienen acceso a cierta información delicada de los usuarios, así que es indispensable aumentar la dificultad con la que éstos pueden loguear. Si alguien accediera a la tabla **Empleados** de la base de datos, con el tiempo podrían obtener la contraseña de los mismos mediante pruebas de fuerza bruta con un diccionario. No obstante, esta cuestión es una de las cosas por las que he implementado el hash de las contraseñas con el algoritmo bcrypt, ya que, recordemos que éste permite dar varios ciclos de encriptación. Aun así, es inquietante el hecho de que tarde o temprano puedan llegar a obtener las contraseñas, y aunque una de las soluciones a este problema es el cambio regular de la contraseña, voy a implementar otro sistema más difícil de eludir: autentificación en 2 pasos con reconocimiento facial.

Para introducir este sistema en mi aplicación web, simplemente habría que modificar el constructor de la clase **Login** en **Login.class.php**, de manera que si es jefe, se tomaría una foto, que se mandaría a un directorio temporal, **./images/user-images/tmp**, y se el programa de **face\_recognition** compararía la foto temporal con la del perfil del usuario. El código del constructor de clase quedaría de la siguiente manera:

```
# Si el hash y la contraseña introducida coinciden
if($auth){
    ob_start();
    session_start();
    $_SESSION['email'] = $this->email;
    $_SESSION['valid'] = true;

    if (es_jefe($this->email)){
        header('Location: hacer-foto.html');
        # header('Location: index-jefes.html');
    }else{
        # Redirige a fichero privado
        header('Location: index.html');
    }
}
```

Si el usuario se autentifica correctamente con correo y contraseña, y es jefe, va a ser redirigido a la página **hacer-foto.html**, donde se podrá comprobar a tiempo real si la cara del usuario es la que figura en su foto de perfil.

Para implementar el reconocimiento facial, usaré el código de [philippmeisberger](#), y lo primero de todo es instalar las librerías necesarias:

```
sudo apt-get install cmake
sudo pip3 install face_recognition
sudo pip3 install flask
sudo pip install -U flask-cors
sudo pip install pillow
```

Para gestionar las similitudes faciales entre el usuario (jefe) que se intenta loguear y su foto de perfil, haré uso de javascript, ya que la página **hacer-foto.html** tendrá un script js incorporado que ejecutará una cámara de vídeo en streaming. Además, cada 3 segundos va a capturar y enviar una foto a una **rest api** implementada en python mediante una solicitud POST. Dicha rest api va a ejecutar el programa de reconocimiento facial, y enviará una respuesta comparando si la persona que aparece en la foto de perfil del usuario, y la que está capturando la cámara en directo, son la misma. Cuando las interprete, generará una solicitud POST de respuesta afirmativa o negativa, y un mensaje correspondiente. El js recibirá la respuesta y la interpretará; habrá 3 posibilidades, autenticación exitosa, lo que llevaría al usuario a **index-jefes.html**, autenticación fallida, en cuyo caso la cámara en directo seguiría capturando y enviando solicitudes, y error, caso en que la cámara también seguiría capturando.

Para ver ésto en más detalle, veamos primero el archivo html.

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="Content-Security-Policy" content="upgrade-insecure-requests">
    <title>Face Authentication</title>
    <link rel="stylesheet" href="../styles.css">
    <style>
```

## SISTEMA DE AUTENTIFICACIÓN Y GEOLOCALIZACIÓN PARA TRABAJO EN REMOTO

```
#video {
    border: 1px solid black;
    width: 700px;
    height: 500px;
}

</style>
</head>

<body>
<div class="main-content">
    <h1>Face Authentication</h1>
    <video id="video" autoplay></video>
    <p id="status"></p>
</div>
<script>
    // Get the video element
    const video = document.getElementById('video');
    const status = document.getElementById('status');

    // Get access to the camera
    if (navigator.mediaDevices && navigator.mediaDevices.getUserMedia) {
        navigator.mediaDevices.getUserMedia({ video: true }).then(function (stream) {
            video.srcObject = stream;
            video.play();
        });
    }

    // Capture an image every few seconds and send it to the server
    setInterval(function () {
        captureAndCompare();
    }, 3000);

    // Function to capture an image and send it to the Flask API
    function captureAndCompare() {
        const canvas = document.createElement('canvas');
        canvas.width = video.videoWidth;
        canvas.height = video.videoHeight;
        const context = canvas.getContext('2d');
        context.drawImage(video, 0, 0, canvas.width, canvas.height);
        const imageData = canvas.toDataURL('image/jpeg');

        fetch('get_user_data.php')
            .then(response => response.json())
            .then(data => {
                if (data.error) {
                    status.textContent = `Error: ${data.error}`;
                }
            });
    }
</script>

```

## SISTEMA DE AUTENTIFICACIÓN Y GEOLOCALIZACIÓN PARA TRABAJO EN REMOTO

```
    } else {
        // Asegúrate de que userImagePath esté definida aquí
        let userImagePath = data.user_image_path;

        // Eliminar las barras invertidas de la ruta de la imagen
        userImagePath = userImagePath.replace(/\\/g, '');

        userImagePath = userImagePath.replace('/var/www/html', '.');

        console.log(userImagePath);

        // Log the received JSON before parsing
        console.log('Received JSON:', data);

        fetch('https://bettercallsergio.es:5000/compare', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({
                user_image_path: userImagePath,
                temp_image_data: imageData
            })
        })
        .then(response => response.json())
        .then(data => {
            console.log('API Response:', data);
            if (data.result === 'success') {
                if (data.match) {
                    status.textContent = 'Authentication successful!
Redirecting...';
                    // Esperar 3 segundos antes de redirigir
                    setTimeout(function () {
                        window.location.href = '../index-jefes.html';
                    }, 3000); // 3000 milisegundos = 3 segundos
                } else {
                    status.textContent = 'Authentication failed. Please
try again.';
                }
            } else {
                status.textContent = `Error: ${data.message}`;
            }
        })
        .catch(error => {
            console.error('Error:', error);
            status.textContent = 'Error connecting to the server. Please
try again.';
        })
    }
}
```

```

try again.';
                })
            }
        })
        .catch(error => {
            console.error('Error:', error);
            status.textContent = 'Error connecting to the server. Please try
again.';
        });
    }
</script>
</body>

</html>

```

De dicho archivo he de destacar primero de todo la siguiente línea:

```
<meta http-equiv="Content-Security-Policy" content="upgrade-insecure-requests">
```

Es muy importante para el funcionamiento del reconocimiento facial, ya que lo que hace es permitir mandar solicitudes a url's que no sean https. Por defecto, los navegadores no permiten mandar solicitudes a páginas http desde una página https, y activar ésto es básico para poder contactar con la rest api, que estará escuchando peticiones en la siguiente url:

<http://192.168.1.222:5000>

Siguiendo con éste archivo, lo primero que hace el script js es activar la cámara del dispositivo del usuario y establecerla en modo **streaming**. Luego establece cada cuánto tiempo va a ejecutarse la función que envía la solicitud a la rest api, en este caso, cada 3 seg. Finalmente, define el código de la función **captureAndCompare()**. Lo primero que va a hacer dicha función es hacer una foto en formato jpeg. A continuación va a llamar al archivo **get\_user\_data.php**, que se va a conectar a la base de datos para consultar el nombre y apellidos del usuario cuyo email está en la variable **\_SESSION**, y va a devolver la ruta de la foto de perfil del usuario en cuestión en formato json. El código de dicho archivo está disponible más abajo para su visualización.

Conociendo la ruta del usuario, va a eliminar los caracteres “\\”, que son añadidos por la función que devuelve los datos en formato json en el php. Además, va a cambiar la ruta absoluta por la relativa, ya que anteriormente tuve problemas al usar rutas absolutas. A continuación, va a mandar una solicitud POST a la rest api, con la ruta de la foto de perfil y la imagen capturada como información. Nótese que la url en la que escucha dicha api es '<https://bettercallsergio.es:5000/compare>', así que he abierto un puerto en el router para que sea posible. A continuación, va a manejar la respuesta de la api como he mencionado anteriormente: éxito, fallo o error. En caso de éxito, va a mostrar un mensaje, esperará 3 segundos para que el usuario pueda visualizarlo, y lo redirigirá a la página **index-jefes.html**.

El código de **get\_user\_data.php** es:

```
<?php
session_start();
require_once('../includes/functions.php');

if (isset($_SESSION['email'])) {
    // $email = $_SESSION['email'];
    $email = $_SESSION['email'];
    $connection = connectionDB();

    $sql = "SELECT Nombre, Apellidos FROM Empleados WHERE Email = :email";
    $stmt = $connection->prepare($sql);
    $stmt->bindParam(':email', $email);
    $stmt->execute();
    $user = $stmt->fetch(PDO::FETCH_ASSOC);

    if ($user) {
        $name = $user['Nombre'];
        $surname = $user['Apellidos'];
        $ruta = generateImagePath($name, $surname);
        $ruta_sin_barras = str_replace('\\', '/', $ruta);

        // Crear el array asociativo con la ruta de la imagen
        $response = ['user_image_path' => $ruta_sin_barras];

        // Enviar el JSON como respuesta
        echo json_encode($response);
    } else {
        echo json_encode(['error' => 'User not found']);
    }
} else {
    echo json_encode(['error' => 'Session not set']);
}
```

```
?>
```

Va a consultar el nombre y los apellidos del usuario cuyo email figura en la variable de sesión, y va a devolver la ruta de su foto de perfil en formato json. Recuerdo que las fotos de perfil de los usuarios se encontrarán en:

**/var/www/html/images/user-images/nombreapellidos/perfil.jpg**

La api rest se ejecuta en el siguiente script de python

```
from flask import Flask, request, jsonify
from flask_cors import CORS
import face_recognition
import os
import ssl
import base64
from io import BytesIO
from PIL import Image

app = Flask(__name__)
CORS(app, resources={r"/compare": {"origins": "https://bettercallsergio.es:8080"}})

def compare_faces(user_image_path, temp_image_data):
    try:
        # Cargar la imagen del usuario desde la ruta proporcionada
        if not os.path.exists(user_image_path):
            return {'result': 'error', 'message': 'User image path does not exist.'}

        user_image = face_recognition.load_image_file(user_image_path)
        user_face_encoding = face_recognition.face_encodings(user_image)

        if len(user_face_encoding) == 0:
            return {'result': 'error', 'message': 'No face found in user image.'}

        # Decodificar la imagen temporal de la cámara
        temp_image = face_recognition.load_image_file(temp_image_data)
        temp_face_encoding = face_recognition.face_encodings(temp_image)

        if len(temp_face_encoding) == 0:
            return {'result': 'error', 'message': 'No face found in captured image.'}

        # Comparar las caras
        match = face_recognition.compare_faces([user_face_encoding[0]],
temp_face_encoding[0])[0]
```

## SISTEMA DE AUTENTIFICACIÓN Y GEOLOCALIZACIÓN PARA TRABAJO EN REMOTO

```
if match:
    return {'result': 'success', 'match': str(match)}
else:
    return {'result': 'error', 'match': str(match)} # Convertir el booleano a
cadena de texto
except Exception as e:
    return {'result': 'error', 'message': str(e)}

@app.route('/compare', methods=['POST'])
def compare_faces_route():
    try:
        data = request.get_json()
        print("Received data:", data)

        user_image_path = data.get('user_image_path')
        temp_image_data = data.get('temp_image_data')

        if not user_image_path or not temp_image_data:
            return jsonify({'result': 'error', 'message': 'User image path and temp image
data are required.'})

        #path_bueno = str(user_image_path).replace("/var/www/html", ".") # No es necesario

        # Decodificar la imagen temporal desde base64
        temp_image_data = temp_image_data.split(",") [1]
        temp_image_data = BytesIO(base64.b64decode(temp_image_data))
        temp_image = Image.open(temp_image_data)
        temp_image.save("./temp_image.jpg") # Guardar la imagen temporal para su uso en
comparación

        result = compare_faces(user_image_path, "./temp_image.jpg")
        return jsonify(result)
    except Exception as e:
        return jsonify({'result': 'error', 'message': str(e)})

if __name__ == '__main__':
    # Configura el contexto SSL
    ssl_context = ssl.create_default_context(ssl.Purpose.CLIENT_AUTH)
    ssl_context.load_cert_chain(certfile='/etc/ssl/certs/bettercallsergio.ddns.net.crt',
keyfile='/etc/ssl/private/bettercallsergio.ddns.net.key')

    # Ejecuta la aplicación Flask con HTTPS
    app.run(host='0.0.0.0', port=5000, ssl_context=ssl_context)
```

Primeramente va a importar las librerías necesarias, y va a crear el objeto api, de manera que ejecute la función main. En la línea justo de debajo se establece para el objeto **app** que se permiten solicitudes de un dominio distinto al de la api.

Luego se declara la función **compare\_faces()**, que acepta como argumento las rutas de las dos imágenes a comparar. Va a comprobar si existen las fotos en las rutas pasadas como argumento. A continuación, va a cargar las dos imágenes con el programa de reconocimiento facial y va a comprobar si hay alguna cara en la foto mediante el método **face\_encoding()**, ya que si éste es igual a 0, no se encuentran caras. Por último va a comparar las caras de las 2 fotos y guardar el resultado en la variable **match**, que devolverá el json con un success si contiene **true**.

Con la siguiente línea la api quedará configurada para escuchar solicitudes POST en el recurso **/compare**.

```
@app.route('/compare', methods=['POST'])
```

A continuación se define la función **compare\_faces\_route()**, que va a desglosar la solicitud POST y guardar las dos rutas en variables. Si no existen las imágenes de las rutas, va a devolver un json con una respuesta de error. A continuación, se va a procesar la imagen de la manera que se ve en el script para decodificar la imagen que se envía desde el navegador en formato base64, convertirla en un formato que pueda ser manejado por PIL, y luego guardarla como un archivo de imagen temporal en el servidor.

Luego, va a llamar a la función explicada anteriormente, que devolverá un json con un success o un error, y mandará la respuesta de vuelta.

En la función main creamos un contexto ssl para que la api pueda utilizar https para comunicarse con el propio backend, por ello, debemos especificarle (línea de abajo) que cargue el certificado ssl y su clave correspondiente. Finalmente especificamos que escuche a todas las direcciones que le contacten por el puerto 5000.

En mi caso, la api va a estar integrada en el propio servidor backend, pero se podría colocar perfectamente en cualquier otro, siempre y cuando la dirección en que se encontrara se especificara correctamente en el **port forwarding** del router.

Para ejecutar la api es recomendable tener una consola específica para dicha función, ya que una vez la ejecutemos, perderemos el control de la consola hasta que paremos la ejecución de la api. La ejecutamos con:

```
usuario@ubuntu:/var/www/html$ sudo python3 api_rest/api_rest.py
[sudo] password for usuario:
 * Serving Flask app 'api_rest'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on https://127.0.0.1:5000
 * Running on https://192.168.1.222:5000
Press CTRL+C to quit
```

Sin embargo, la consola en la que ejecutamos la api, sigue siendo útil, ya que en ella podremos ver las diferentes peticiones que le llegan

En el siguiente y último punto muestro visualmente el funcionamiento del sistema de reconocimiento facial.

## 8. Prueba de inicio de sesión de empleado con biometría.

Lo primero es saber cual es el usuario que vamos a loguear, ya que tiene que ser uno registrado como jefe. Haciendo una consulta a la base de datos filtrada por jefes, obtenemos que actualmente mi único usuario logueado es el siguiente:

Result Grid   Filter Rows: [ ]   Edit: [ ]   Export/Import: [ ]   Wrap Cell Content: [ ]						
Id_Emppleado	Nombre	Apellidos	Email	Password	Jefe	Fecha_Creacion
da4b3362-227d-11ef-882c-0800275e5af5	Sergio	Pérez Ríos	sperrio2706@ieszaidinvergeles.org	\$2a\$12\$wAVgRUjxHAoo18L0B6oZ2.c0.Q1/t5Q...	S	2024-06-04 16:22:20
HULL	HULL	HULL	HULL	HULL	HULL	HULL

Mi propio usuario, Sergio, es el único jefe registrado, así que haremos el login con él. Accedemos a la web e introducimos el usuario y la contraseña:

Inicio de sesión

Solo para empleados

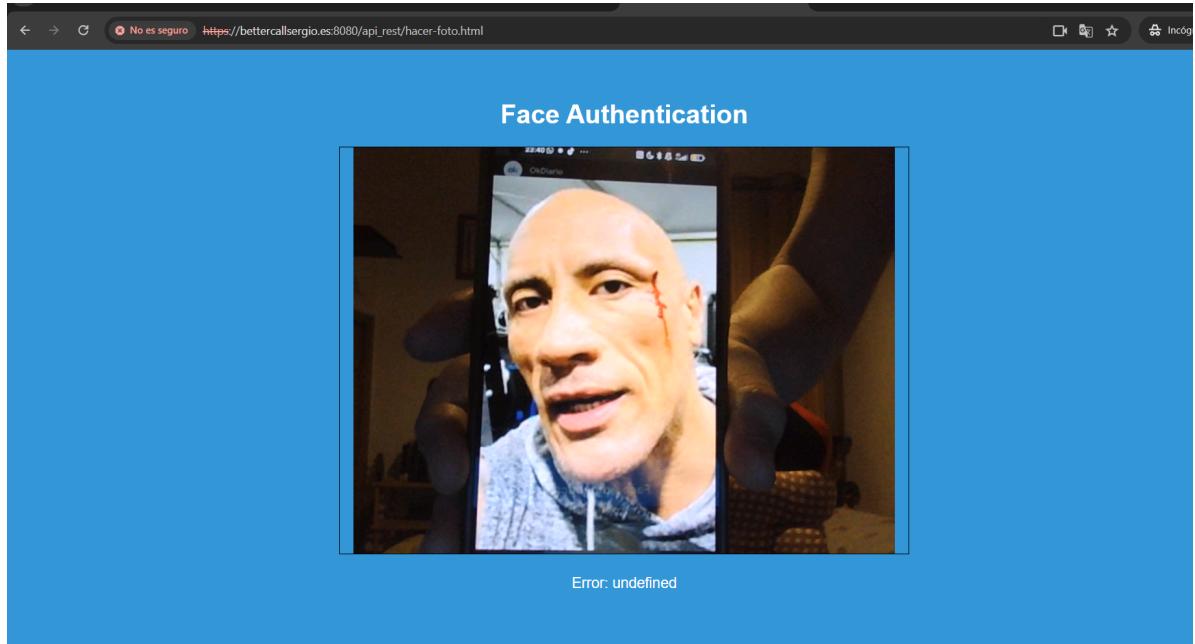
Email:

Contraseña:

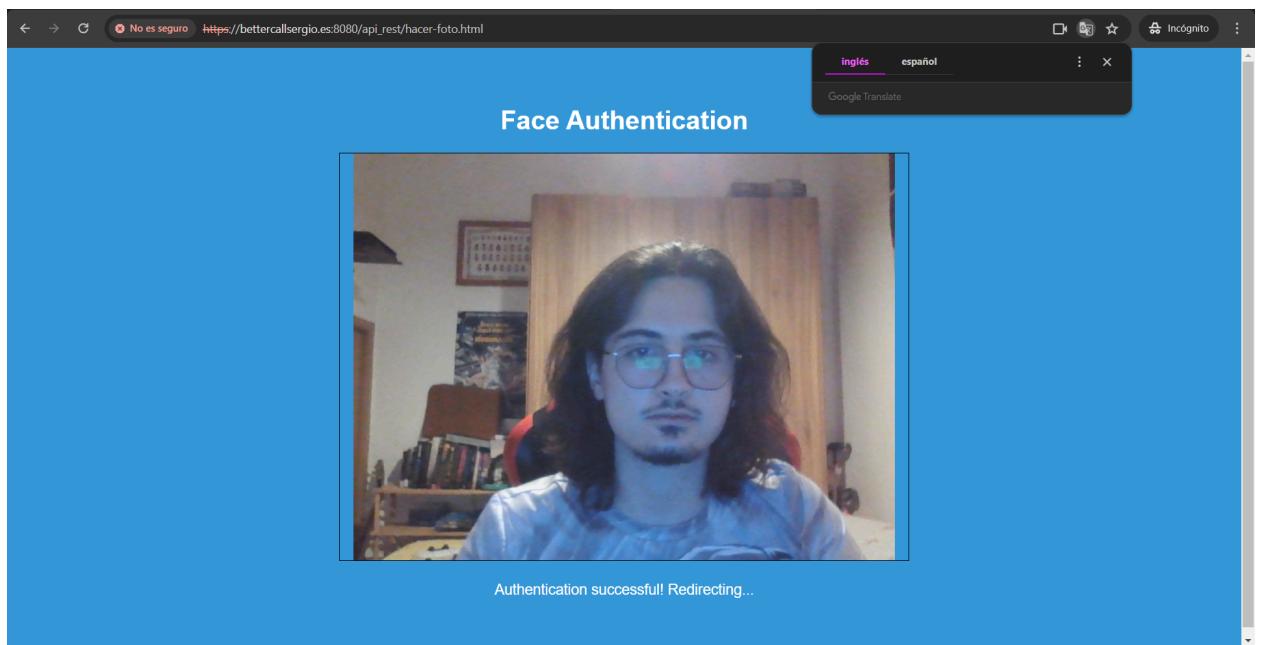
Enviar

## SISTEMA DE AUTENTIFICACIÓN Y GEOLOCALIZACIÓN PARA TRABAJO EN REMOTO

Cuando pulsemos enviar nos llevará a la página **hacer-foto.html**, en la cual, para hacer la prueba, voy a mostrar una foto de Dwayne Johnson:



Se puede ver que no la reconoce. Ahora voy a mostrar mi cara, así que tendré acceso. La siguiente es una captura de pantalla que me ha dado tiempo a hacer antes de que me dirigiera a la página de los jefes:



En la foto podemos leer el mensaje **Authentication succesful! Redirecting**, que se muestra 3 segundos antes de dirigirnos a la página de más abajo:



## Conclusión

### ¿Qué he aprendido?

Del proyecto puedo sacar un conjunto de cosas que he aprendido conforme iba implementando el sistema descrito en las anteriores páginas. Lo primero que me gustaría destacar es el concepto de rest api. Nunca antes había utilizado una para mis sistemas, y me ha parecido de lo más útil. El hecho de que puedas configurar a tu gusto un servicio que escuche las peticiones que le ordenes, y devuelva otras, permite una gran infinidad de posibilidades. Además, me ha gustado aprender cómo se envían y manejan las solicitudes POST, sabiendo que están en formato JSON.

Por otro lado, me he sentido muy cómodo desarrollando la página en general, haciendo hincapié en la integración de la base de datos para darle interactividad a la misma. Además, me llevo también como lección aprendida el saber cómo funcionan los logins de una aplicación web con php y los registros de usuarios, haciendo uso de clases y sesiones para estas cuestiones.

Una de las cosas que más me han llamado la atención es el darse cuenta de errores lógicos al implementar la página, en los que no habría caído de no ser por haber estado desarrollándola. Por ejemplo, el error que traté debido a que los usuarios podían fichar todas las veces que quisieran, rompiendo completamente el sistema con el simple hecho de pulsar el botón de fichar repetidas veces.

Finalmente, me ha parecido interesante el hecho de usar la geolocalización en mi proyecto, y cómo es posible su integración en los diferentes aspectos del mismo (tipo POINT en mysql, latitud y longitud para envío de solicitudes POST, ect...).

## Anexo

Fuentes consultadas:

<a href="https://www.youtube.com/watch?v=lBCd31uzzaM">https://www.youtube.com/watch?v=lBCd31uzzaM</a>
<a href="https://dev.mysql.com/blog-archive/geography-in-mysql-8-0/">https://dev.mysql.com/blog-archive/geography-in-mysql-8-0/</a>
<a href="https://dev.mysql.com/blog-archive/storing-uuid-values-in-mysql-tables/">https://dev.mysql.com/blog-archive/storing-uuid-values-in-mysql-tables/</a>
<a href="https://gridscale.io/en/community/tutorials/apache-server-reverse-proxy-ubuntu/">https://gridscale.io/en/community/tutorials/apache-server-reverse-proxy-ubuntu/</a>
<a href="https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia">https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia</a>
<a href="https://www.freecodecamp.org/news/javascript-post-request-how-to-send-an-http-post-request-in-js/">https://www.freecodecamp.org/news/javascript-post-request-how-to-send-an-http-post-request-in-js/</a>
<a href="https://stackoverflow.com/questions/11525726/hiding-an-html-forms-submit-button">https://stackoverflow.com/questions/11525726/hiding-an-html-forms-submit-button</a>
<a href="https://github.com/ageitgey/face_recognition">https://github.com/ageitgey/face_recognition</a>
<a href="https://j2logo.com/flask/tutorial-como-crear-api-rest-python-con-flask/">https://j2logo.com/flask/tutorial-como-crear-api-rest-python-con-flask/</a>
<a href="https://www.php.net/manual/es/pdo.query.php">https://www.php.net/manual/es/pdo.query.php</a>
<a href="https://www.php.net/manual/es/class.datetime.php">https://www.php.net/manual/es/class.datetime.php</a>
<a href="https://www.php.net/manual/es/function.password-hash.php">https://www.php.net/manual/es/function.password-hash.php</a>

## Anexo

Branches => backend | frontend:

<https://github.com/sperrio2706/TFG-ASIR>

■ ■ ■

Sergio Pérez Ríos

Datos de contacto:

[sperrio2706@ieszaidinvergeles.org](mailto:sperrio2706@ieszaidinvergeles.org)  
[sergiopereriosxd@gmail.com](mailto:sergiopereriosxd@gmail.com)  
<https://bettercallsergio.es>

Web del proyecto:

<https://bettercallsergio.es>