

# Inicio de sesión con Google

# **Implementación en un proyecto desde cero.**

---

Franciso Javier Florencio Airado

Laura Merino Ortiz

Antonio Carlos Moruno Gómez

13 de junio del 2023



# ÍNDICE

<b>ÍNDICE</b>	<b>1</b>
<b>1. Resumen</b>	<b>2</b>
<b>2. Pasos a seguir</b>	<b>2</b>
2.1. Google API	2
2.2. Composer	7
2.3. Configuración dentro del proyecto	8

## 1. Resumen

En este documento se explicarán los pasos a seguir utilizados para implementar el inicio de sesión en un proyecto con la API que Google pone a disposición de los usuarios. Aconsejamos encarecidamente utilizar este método para controlar las sesiones y los usuarios, pues nos evitaremos guardar contraseñas y datos vulnerables de los usuarios en nuestra base de datos, cubriendonos la espalda ante posibles futuros ataques y robos de contraseñas.

También, nos gustaría especificar que la tecnología que usamos para este tutorial han sido javascript para la parte del cliente y php para la parte de servidor, ambos en su versión sin frameworks.

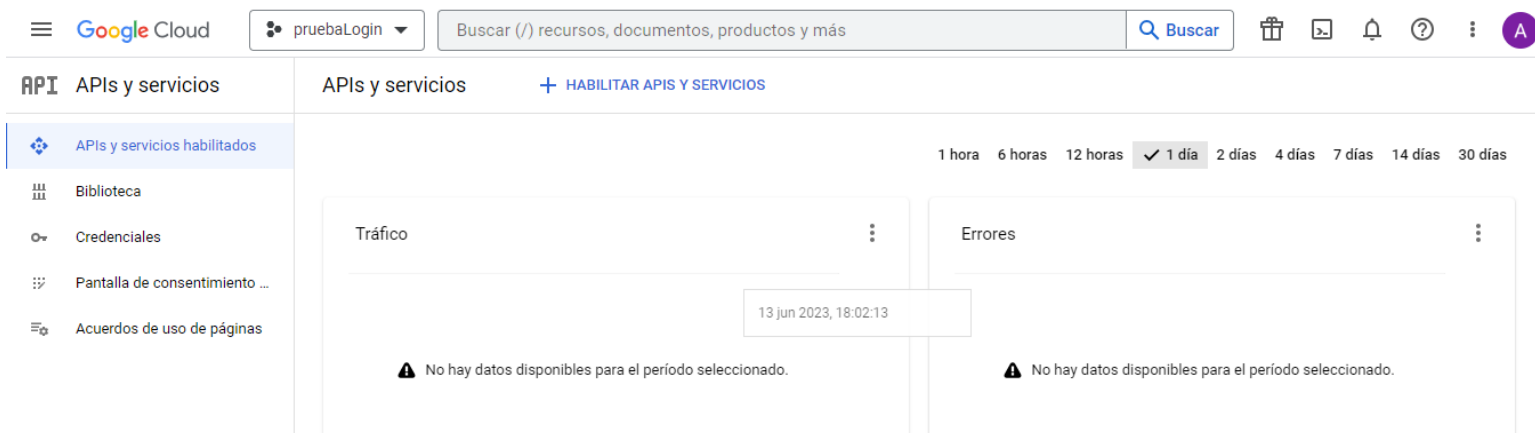
## 2. Pasos a seguir

A partir de aquí mostraremos los pasos desde cero que hemos seguido para facilitar la aplicación de este método a futuros usuarios que quieran implementarlo.

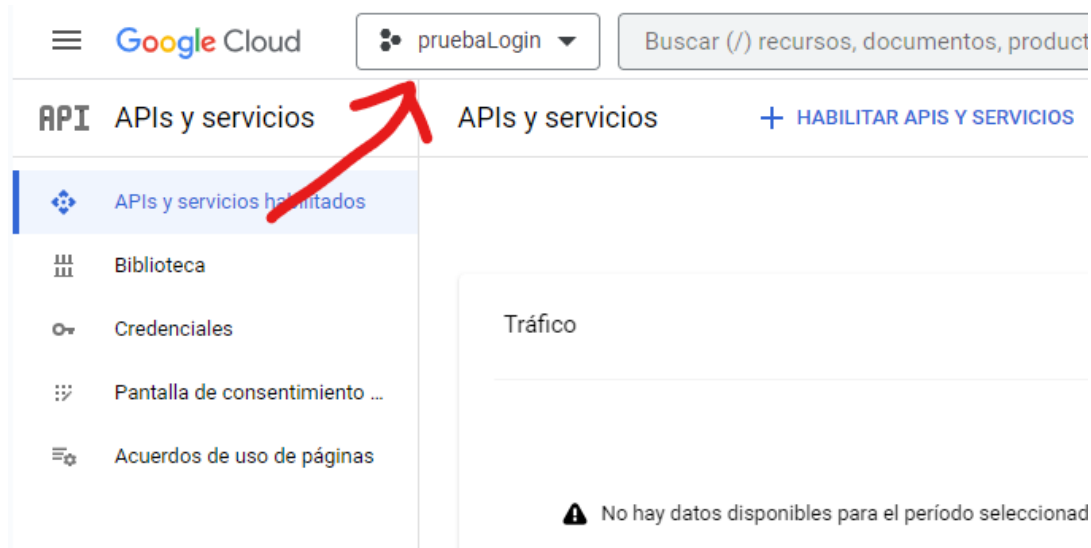
### 2.1. Google API

Lo primero de todo es dirigirnos al enlace de la API de Google que es el siguiente <https://console.developers.google.com/>. Si estáis tratando de acceder con un correo electrónico con restricciones, como puede ser el de una escuela o institución, posiblemente no os deje entrar, aconsejamos entrar con el correo personal.

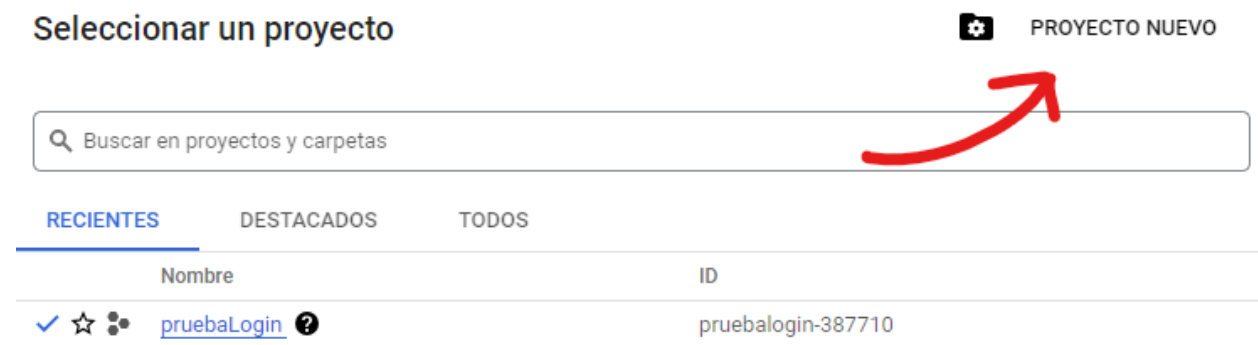
Una vez pulsado el enlace e iniciado sesión con google os aparecerá una ventana como la siguiente.



En esta ventana lo primero que debemos hacer es pulsar sobre el botón de arriba a la izquierda donde pone “pruebaLogin”, aunque si no tenéis ningún proyecto anteriormente montado puede que os aparezca “Crear proyecto” o similar.




Tras darle a este botón se nos abrirá una ventana con los proyectos que tenemos actualmente, pero para montar uno nuevo pulsaremos sobre el botón “PROYECTO NUEVO” arriba a la izquierda de la ventana.




Lo único que deberemos hacer para crear el proyecto será rellenar el nombre del proyecto con el nombre que queramos y darle a crear.


## Proyecto nuevo

 Tienes 11 projects restantes en tu cuota. Solicita un incremento o borra algunos proyectos. [Más información](#)

[MANAGE QUOTAS](#)

**Nombre del proyecto \***  
MiProyectoDePruebaTutorial 

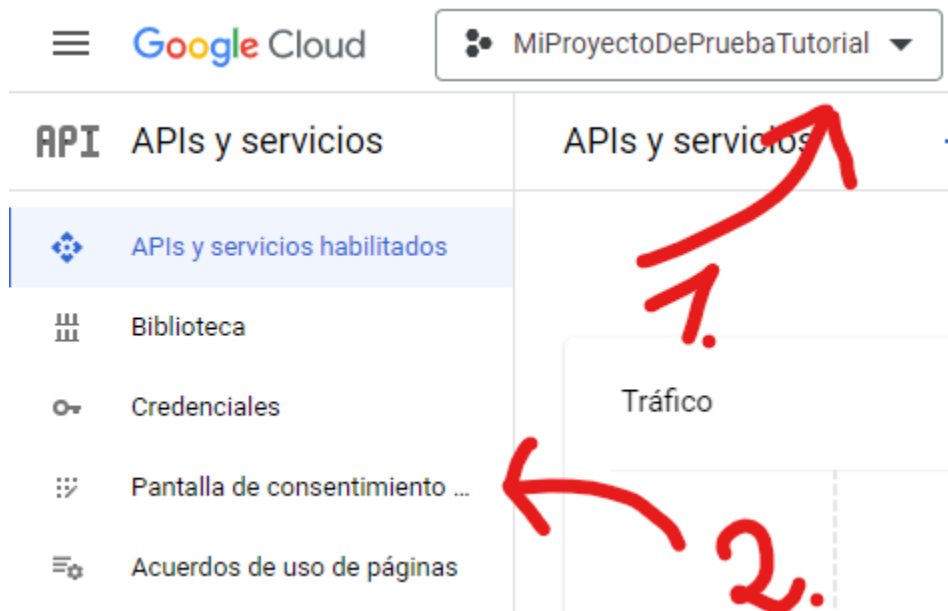
ID de proyecto: miproyectodepruebatutorial. No se podrá cambiar más tarde. [EDITAR](#)

**Ubicación \***  
 Sin organización [EXPLORAR](#)

Organización o carpeta superior

[CREAR](#) [CANCELAR](#)

Ahora nos aseguramos que esté nuestro proyecto creado seleccionado en seleccionable de arriba y posteriormente pulsaremos sobre la opción “Pantalla de consentimiento” que aparece en el menú de navegación lateral.



Llegados a este punto seleccionamos la opción que queramos para el tipo de usuario. En nuestro caso seleccionamos la opción “Externos” y le damos a crear.

## Pantalla de consentimiento de OAuth

Elige cómo deseas configurar y registrar tu app, incluidos los usuarios objetivo. Puedes asociar una sola app con tu proyecto.

### User Type

☐ Interno ?

Solo está disponible para los usuarios de tu organización. No necesitarás enviar tu app para verificarla. [Obtén más información sobre el tipo de usuario](#)

☒ Externos ?

Está disponible para cualquier usuario de prueba con una Cuenta de Google. Tu app se iniciará en modo de prueba y solo estará disponible para los usuarios que agregues a la lista de usuarios de prueba. Una vez que la app esté lista para enviarse a producción, puede que debas verificarla. [Obtén más información sobre el tipo de usuario](#)

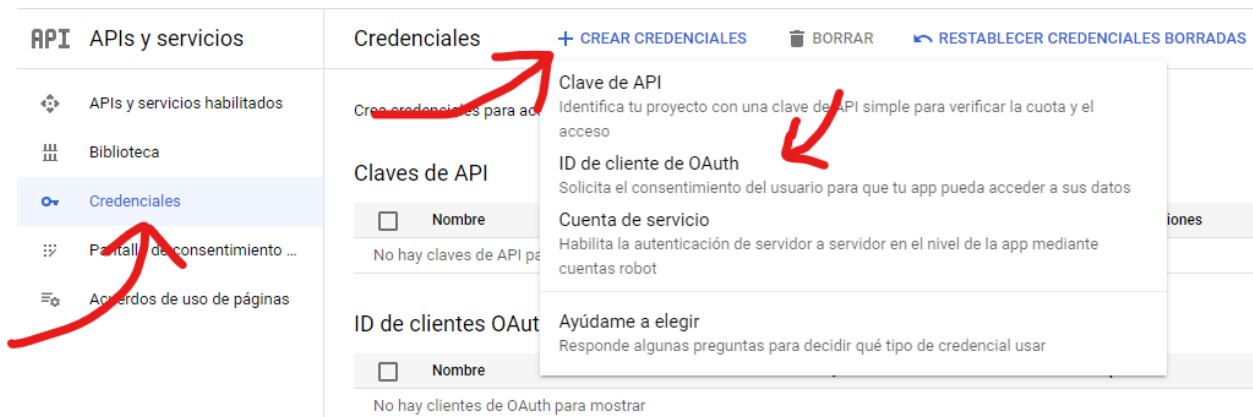
CREAR

Seguidamente rellenamos el formulario que sale a continuación, en el que solo es necesario rellenar el nombre de la aplicación, el correo electrónico de asistencia a usuario y la dirección de correo electrónico para la información de contacto del desarrollador, que para este caso de ejemplo pondremos el nuestro personal.

Para acabar con esta parte no hace falta que rellenemos nada más, le vamos dando a guardar y continuar hasta finalizar.

Lo siguiente que debemos hacer es configurar las credenciales. Desde el menú lateral de la izquierda accederemos a la parte de “Credenciales”.

Una vez accedido, le daremos al botón de “+ CREAR CREDENCIALES” y seleccionamos la opción que dice “ID de cliente de OAuth”



Una vez accedido, se nos mostrará un seleccionable para indicar el tipo de aplicación en el que deberemos seleccionar la opción de “Aplicación Web”.

Tras seleccionar dicha opción, aparecerán unas opciones adicionales. En esta parte tendréis que ir a Orígenes autorizados de JavaScript e indicar las URI que estarán autorizadas. Suponiendo que el proyecto montado sea en local o en un servidor personalizado, debería quedar algo como en la siguiente imagen.

## Orígenes autorizados de JavaScript ?

Para usar con solicitudes de un navegador

URI 1 *	<input type="text" value="https://09.2daw.esvirgua.com"/>
URI 2 *	<input type="text" value="http://localhost"/>

Finalmente guardamos los cambios y salimos de esta ventana. Se habrá creado un ID de cliente que aparecerá en la vista de Credenciales.

API

APIs y servicios

APIs y servicios habilitados

Biblioteca

Credenciales

Pantalla de consentimiento ...

Acuerdos de uso de páginas

Credenciales

+ CREAR CREDENCIALES

BORRAR

Crea credenciales para acceder a tus API habilitadas. [Más información](#)

Claves de API

Nombre

Fecha de creación ↓

No hay claves de API para mostrar

ID de clientes OAuth 2.0

Nombre

Fecha de creación ↓

Cliente web 1

24 may 2023

Si pulsamos sobre el nombre podremos editar las URI si quisiéramos junto a otros datos, pero lo importante de esta parte es que desde aquí podemos acceder al **ID del cliente** y al **número secreto del cliente**, que nos servirán más adelante.

Nombre \*

Cliente web 1

El nombre de tu cliente de OAuth 2.0. Este nombre solo se usa para identificar al cliente en la consola y no se mostrará a los usuarios finales.

Los dominios de los URI que agregues a continuación se incorporarán automáticamente a tu [pantalla de consentimiento de OAuth](#) como [dominios autorizados](#).

Orígenes autorizados de JavaScript

Para usar con solicitudes de un navegador

URI 1 \*

https://09.2daw.esvirgua.com

URI 2 \*

http://localhost

URI 3 \*

Additional information

ID de cliente

Fecha de creación

24 de mayo de 2023, 12:50:43 GMT+2

Secretos del cliente

Si estás en proceso de cambiar los secretos del cliente, puedes rotarlos de forma manual sin tiempo de inactividad. [Más información](#)

Secreto del cliente

Fecha de creación

24 de mayo de 2023, 12:50:43 GMT+2

Estado

✓ Habilitado

Con esto habríamos terminado la parte de configuración de la API de google, ahora seguiremos con los siguientes pasos.

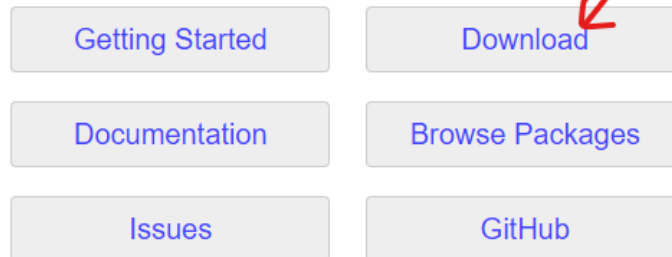
## 2.2. Composer

Para que el inicio de sesión de google funcione en nuestra aplicación, deberemos descargar Composer desde esta url <https://getcomposer.org/>



# A Dependency Manager for PHP

Latest: **2.5.8** ([changelog](#))



[Home](#) | [Getting Started](#) | [Download](#) | [Documentation](#) | [Browse Packages](#)

## Download Composer Latest: v2.5.8

### Windows Installer

The installer - which requires that you have PHP already installed - will download Composer for you and set up your PATH environment variable so you can simply call `composer` from any directory.

Download and run [Composer-Setup.exe](#) - it will install the latest composer version whenever it is executed.

Una vez descargado ejecutamos el .exe e instalamos Composer.

Tras descargar este programa, abriremos la ventana de comandos de windows en la carpeta de nuestro proyecto y ejecutamos el comando `composer require google/apiclient:"^2.0"`. Este proceso me dio problemas, pues no conseguía acabar el proceso al introducir el comando. Mi solución fue desactivar la Comprobación de aplicaciones y archivos de las opciones de Seguridad de Windows (no olvidar volver a activar la opción al finalizar el proceso).

## 2.3. Configuración dentro del proyecto

Lo primero que debemos hacer es irnos a nuestro index.html e introducir en el head la etiqueta `<script src="https://accounts.google.com/gsi/client" async defer></script>.` Aunque si por el diseño de vuestra página la vista del login no está enlazada con el index, también podéis introducir la etiqueta script dentro del `<div>` donde queráis que aparezca el botón de inicio de sesión de google.

El siguiente pasó será crear un estilo sencillo en el que irá el botón de iniciar sesión, que para probar puede ser algo como el siguiente código.

```
<div class="col-sm-6 mt-5 ps-5 pe-5">
  <h1>Login por Google</h1>
  <div id="divGoogleLogin" class="d-flex justify-content-center"></div>
</div>
```

Ahora que ya sabemos dónde va a estar nuestro botón de iniciar sesión, tenemos que montar la parte de javascript que va a obtener el Token de inicio de sesión y el que controla la vista de esta parte. En nuestro caso lo tenemos montado de la siguiente manera:

```
export class Inicio{
  constructor (controlador) {
    this.controlador=controlador
    window.setTimeout(this.iniciar.bind(this), timeout: 500)
    window.onerror = function (error : Event | string ) { console.log('Error capturado. ' + error) }
  }

  /**
   Inicia el login.
   Se llama al cargar la página
  */
  iniciar () {
    // Login con Google
    google.accounts.id.initialize({
      client_id: 'REDACTED',
      callback: this.login.bind(this)
    })
    google.accounts.id.renderButton(
      document.getElementById( elementId: 'divGoogleLogin'),
      { theme: 'outline', size: 'large' } // customization attributes
    )
  }
}
```

En la función iniciar, que se carga al iniciar esta vista, se utilizan unas clases específicas de la api de google con un método. En el atributo client\_id debéis poner el id del cliente que se describe en los pasos anteriores, y en el callback debéis poner el nombre de la función a la que llamaréis tras darle click al botón de google.

En google.accounts.id.renderButton dais estilo al botón de inicio de sesión de google. Tendréis que asegurarnos que estáis cogiendo la referencia del div con el id correcto, pues si no

dará problemas, entre los que pueden estar que el botón coja unas dimensiones incorrectas o incluso que incluso os topéis con una vista totalmente en blanco en la que solo se muestre el botón de inicio de sesión.

Como iba diciendo anteriormente, cuando se pulse sobre el botón de iniciar sesión, se llamará a la función que indiquemos en el callback, que en nuestro ejemplo el nombre de dicha función es login.

```
login (token : Object ) {
  this.controlador.loginGoogle(token.credential) Promise<void>
    .then(tokenSession => {
      sessionStorage.setItem('token', tokenSession)
      let datosToken= JSON.parse (atob (tokenSession.split('.')[1]));
      console.log(datosToken);

      Swal.fire({
        title: 'Login realizado con éxito',
        text: 'Bienvenido, ' + datosToken.nombre,
        icon: 'success',
        confirmButtonText: 'Aceptar'
      }).then((result) => {
        this.controlador.mostrarHome();
      })
    }) Promise<void>
    .catch(e => {
      Swal.fire({
        title: 'Usuario no válido',
        text: 'Usuario sin acceso a la aplicación.',
        icon: 'error',
        confirmButtonText: 'Aceptar'
      });
    })
}
```

En este método como podemos comprobar le llega una variable, que es un token que crea de forma automática la api de Google, el cual es un objeto en formato json con información. Dentro de este token hay una parte que nos interesa, “credentials”, que es más específicamente el token en sí. Es una cadena de caracteres alfanumérica codificada con información dentro y es lo que debemos enviar al servidor. A continuación se mostrará la forma en la que lo enviamos nosotros: → controlador.js → modelo.js → controlador.php → modelo.php.

- Page 10 of 10

```
async loginGoogle(credenciales){
    return await this.modelo.doLogin(credenciales)
}
```

- modelo.js

```

async doLogin(token) {
  return new Promise( {executor: resolve => {
    $.post({
      url: `${this.base_url}login/comprobarUsuario`,
      data: {token: token},
      success: (data) => {
        resolve(data);
        console.log(data)
      },
      error: (error) => {
        console.log('Error en la solicitud:', error.responseText);
        resolve(error);
      }
    });
  }});
}

```

- controlador.php

Aquí lo primero que debemos hacer es declarar las variables que vamos a usar más adelante, sobre todo el id del cliente, que más arriba se muestra dónde obtenerlo.

```
<?php
use Firebase\JWT\JWT;

require_once('modelos/modeloLogin.php');
require_once ('config/login/google/vendor/autoload.php');
require_once ('config/config.php');

/**
 * Clase Controlador de Login
 */
class LoginController{
    private $modelo;
    public static $iv = '16bits aleatorio';
    //Id de cliente de Google.
    private static $ID_CLIENTE = " ";
    function __construct(){
        $this->modelo=new ModeloLogin();
    }
}
```

Una vez hemos declarado las variables debemos crear el método al que le llega el token. Vamos a dividir este método en dos partes.

La primera parte se encargará de recoger la variable que ha sido mandada por el método POST. Se recogerá el cliente de google en otra variable (\$client) y se verificará si y decodificará el token, guardando la información del payload del token decodificado en otra variable (\$payload).

Tras esto, se comprueba si el payload existe para comprobar si hay autorización por parte de google y es un correo válido para esta aplicación, y finalmente se envía al modelo.php al método comprobarUsuario la variable de dentro del payload llamada 'email', que contendrá el email del usuario que trata de iniciar sesión

```
/**
 * Método para comprobar el usuario
 */
public function comprobarUsuario(){

    $token = $_POST["token"];

    $client = new Google_Client(['client_id' => self::$ID_CLIENTE]);
    $payload = $client->verifyIdToken($token);

    if (!$payload) {
        // Invalid ID token
        header( header: 'HTTP/1.1 401 Unauthorized');
        die();
    }

    //Primero se consiguen los datos del usuario que trata de iniciar sesión con google
    $usuario = $this->modelo->comprobarUsuario($payload['email']);

    //Si el email está en la bbdd, se sacan sus roles y se devuelve un token nuevo
```

- modelo.php

En el método comprobarUsuario del modelo.php, se comprueba si existe en la base de datos el email del usuario que trata de iniciar sesión. Si existe devuelve al controlador los datos de la base de datos de ese usuario, si no existe retorna un null.

```

public function comprobarUsuario($email) {
    $this->conectar();

    $resultado = $this->conexion->prepare("SELECT * FROM colaboradores t1 WHERE t1.correo = ?;");
    $resultado->bind_param('s', $email);
    $resultado->execute();

    // Obtener el resultado de la consulta
    $resultados = $resultado->get_result();

    // Verificar si se encontró un usuario
    if ($resultados->num_rows > 0) {
        // Obtener el primer resultado como un array asociativo
        $usuario = $resultados->fetch_assoc();

        return $usuario;
    } else {
        // Cerrar la conexión y liberar los recursos
        $resultado->close();
        $this->conexion->close();

        // Devolver null si no se encontró un usuario
        return null;
    }
}

```

- De vuelta al controlador

Se comprueba si existe la variable \$usuario que lleva los datos de lo que haya retornado el modelo.php. Si la variable \$usuario no es null, se le añaden los roles que posee dicho usuario (esta parte es opcional, si en la aplicación que estéis haciendo todos los usuarios poseen los mismos beneficios no es necesario añadirle los roles).

Tras añadirle los roles, se le da valor a la variable \$payload, que llevará varios atributos dentro, como el nombre, el email y los roles.

Para finalizar se crearemos un código secreto (en el ejemplo se ha utilizado un método que genera una cadena de 32 caracteres aleatoria, pero este código secreto lo podéis inventar vosotros), y para finalizar, le damos valor a la variable \$token, que va a ser la que devolvamos al cliente. Esta variable \$token se codificará como un Json Web Token, en el que le tenemos que introducir el payload (datos del usuario que nos pueden ser útiles), el código secreto y un método de codificación, que hemos utilizado el HS256. Tenéis que tener en cuenta que antes de

realizar este proceso para generar el token, debéis descargaros la librería que se llama Firebase-JWT e incluirla al inicio del controlador, como se puede apreciar en la primera vista del controlador.

Este token será el que se devuelva al cliente.

```
//Primero se consiguen los datos del usuario que trata de iniciar sesión con google
$usuario = $this->modelo->comprobarUsuario($payload['email']);

//Si el email está en la bbdd, se sacan sus roles y se devuelve un token nuevo
if ($usuario != null){
    $roles = $this->modelo->getRoles($usuario['id_colaborador']);

    $payload = [
        'nombre' => $usuario['nombre'],
        'email' => $usuario['correo'],
        'roles' => $roles
    ];

    $secretKey = bin2hex(random_bytes( 32));
    $token = JWT::encode($payload, $secretKey, 'alg: HS256');
    echo $token;
    /*$tokenJson = json_encode($token);
    echo $tokenJson;*/
}else{
    http_response_code( response_code: 401);
    echo 'Usuario no permitido';
}
}
```

De vuelta al cliente, manejaremos la respuesta devuelta por el controlador. Nuestra opción ha sido guardar el token devuelto en el sessionStorage, para que así sea accesible desde cualquier vista de la aplicación, teniendo a disposición los datos del usuario que se quieran utilizar.

También realizamos una demostración de cómo decodificar el token y poder acceder a los datos del payload, pues sino lo único que tendremos es una cadena de caracteres alfanuméricos, y eso sin decodificar puede que no nos sirva de mucho.

Por otro lado, aunque en el ejemplo se utiliza una librería externa para controlar la respuesta (SweetAlert) cada uno puede manejar la respuesta como quiera. En este caso, tras

comprobar que el usuario tiene acceso a la aplicación, se le da la bienvenida con un mensaje y se cambia la vista al home de la aplicación web, y si no tiene acceso o algo ha salido mal, se mostrará un mensaje avisando al usuario de que no tiene permisos.

```
login (token : Object ) {  
  this.controlador.loginGoogle(token.credential) Promise<void>  
    .then(tokenSesion => {  
      sessionStorage.setItem('token', tokenSesion)  
      let datosToken= JSON.parse (atob (tokenSesion.split('.')[1]));  
      console.log(datosToken);  
  
      Swal.fire({  
        title: 'Login realizado con éxito',  
        text: 'Bienvenido, ' + datosToken.nombre,  
        icon: 'success',  
        confirmButtonText: 'Aceptar'  
      }).then((result) => {  
        this.controlador.mostrarHome();  
      })  
    })  
    .catch(e => {  
      Swal.fire({  
        title: 'Usuario no válido',  
        text: 'Usuario sin acceso a la aplicación.',  
        icon: 'error',  
        confirmButtonText: 'Aceptar'  
      });  
    })  
}
```