Autenticación por Token

Importante

Un token viene a ser como un cifrado de una firma digital pero ¿qué es una firma digital?
Pues vendría siendo algo así como una cadena larga de texto donde está encriptada cierta información

Dentro de un token vamos a guardar o recibir información de un usuario, un token nunca se almacena en un servidor, por lo que no ocupa espacios con sesiones activas

Usualmente en la autenticación normal se consume memoria en el servidor pero si existen muchos usuarios a la vez ingresando pues se cae el servidor , se podría pensar en incrementar la memoria del mismo pero esto es costoso y dificilmente escalable.

La autenticación por token surge como una solución puesto que al logearse lo que se genera es un token, y una vez obtenido el mismo pues es enviado al servidor donde ahí es descifrado y una vez descifrado sabemos a qué usuario pertenece el mismo.

Pero, ¿qué sucede con el usuario?

Pues al momento que se requiere hacer consultas, listar tareas, personas ,etc pues la URL va a tene el token y el servidor recibirá la URL con el token y únicamente para la consulta que sea requerida va a transformar el token en los datos del usuario.
Entonces, como para esa consulta ya tenemos el usuario en base al token que nos enviaron, es bastante sencillo tomar ese valor y luego hacer el filtro

Lo que quiere decir que ahora para cada consulta que necesitemos verificación de usuario, necesitamos obligatoriamente enviar un token, pero esto es trabajo en sí del cliente/navegador

Comenzaríamos desde el cliente, haciendo una petición POST para enviar el usuario y contraseña, y realizar el proceso de login.

Se comprobaría que ese usuario y su contraseña son correctos, y de serlos, generar el token JWT para devolverlo al usuario.

A partir de ahí la aplicación cliente, con ese token, haría peticiones solicitando recursos, siempre con ese token JWT dentro de un encabezado, que sería Authorization: Bearer XXXXXXX, siendo Bearer el tipo de prefijo seguido de todo el contenido del token.

En el servidor se comprobaría el token mediante la firma, para verificar que el token es seguro, y, por tanto podemos confiar en el usuario.

Dentro del cuerpo del token, además, tenemos los datos de quién es el usuario que ha realizado esa petición, porque podemos contener en el payload todos los datos de usuario que queramos.

Tras verificar que el token es correcto y saber quién es el que ha hecho la petición, podemos aplicar entonces el mecanismo de control de acceso, saber si puede acceder o no, y si es así, responder con el recurso protegido, de manera que lo podría recibir de una forma correcta.

De esta forma podríamos implementar el proceso de autenticación, y hacerlo, además, con estos JSON Web Token.

La autenticación basada en tokens es el proceso de verificar la identidad mediante la comprobación de un token. En la gestión de acceso, los servidores utilizan la autenticación por token para comprobar la identidad de un usuario, una API, un ordenador u otro servidor. Un token es un elemento simbólico que expide una fuente de confianza. Pensemos en cómo los policías llevan consigo una insignia expedida por las autoridades que legitima su autoridad. Las fichas pueden ser físicas (como una llave USB) o digitales (un mensaje generado por ordenador o una firma digital).

¿Cómo funciona la autenticación mediante token web?

Un token web es digital, no un objeto físico. Es un mensaje enviado desde un servidor a un cliente y que este almacena temporalmente. El cliente incluye una copia del token en las siguientes solicitudes enviadas al servidor para confirmar el estado de autenticación del cliente.

Mientras que la autenticación con tokens físicos verifica la identidad durante el proceso de inicio de sesión, los tokens web se emiten como resultado de un inicio de sesión con éxito. Mantienen activa la sesión iniciada.

Sin embargo, utilizar tokens web para las sesiones de usuario no siempre es lo mejor. Muchos desarrolladores son partidarios de utilizar cookies en su lugar. Los tokens web se pueden utilizar mejor para la autenticación de puntos finales de la API o para validar una conexión entre servidores, en lugar de entre el servidor y el cliente.

Ojo que nos estamos apoyando en Google Firebasepara poder generar los Token

Creación de Proyecto Firebase - Autentication

En firebase se creó el proyecto con el nombre Django-pro , se configuró que la auntenticación sea por medio del correo, se creo un template basico con un botón para el login....

Lo que se va a hacer ahora es utilizar la documentación que nos ofrece Firebase e implementar la funcionalidad en este botón, de tal forma que cuando alguien le dé clic en el mismo, este nos abre el formulario para poner nuestros datos de Google y que este a su vez recupere un token.

Ese token es el que debemos enviar hacia nuestro servidor y hacer el proceso respectivo

Integrando Firebase en un template Django

Los servicios de Firebase (como Cloud Firestore, Authentication, Realtime Database, Remote Config y muchos más) están disponibles para importarse en subpaquetes individuales.

https://www.youtube.com/watch?v=tn5n8zOHHN8 <= Actualización

Nota: Se recomienda usar la versión 9 del SDK, en especial para las apps de producción. Si necesitas compatibilidad con otras opciones de administración de SDK, como `window.firebase`, consulta Actualiza de la versión 8 al SDK web modular o Formas alternativas de agregar Firebase.

En éste caso sería la versión 8 puesto que la 9 está mas orientada a trabajar con JS modular, Vue, React, entonces vamos a actualizar de la versión 8 al SDK web modular.

Entonces se añaden los scripts en el html junto con la configuracion/keys del proyecto,
con ésto ya estaría solucionado el primer error debido a las actualizaciones. ya está instalado correctamente el SDK de firebase.

Sumado a éso hay otro problema de actualización con la función de login, entonces para ello creamos una función y dentro de ésta función irá todo lo relacionado al
Manejo del flujo de acceso con el SDK de Firebase, codigo que ya nos facilita firebase

https://firebase.google.com/docs/auth/web/google-signin?authuser=2#web-version-8_1

```
firebase.auth()
  .signInWithPopup(provider)
  .then((result) => {
    /** @type {firebase.auth.OAuthCredential} */
    var credential = result.credential;

    // This gives you a Google Access Token. You can use it to access the Google API.
    var token = credential.accessToken;
    // The signed-in user info.
    var user = result.user;
    // ...
  }).catch((error) => {
    // Handle Errors here.
    var errorCode = error.code;
    var errorMessage = error.message;
    // The email of the user's account used.
    var email = error.email;
    // The firebase.auth.AuthCredential type that was used.
    var credential = error.credential;
    // ...
  });
```

Una vez hecho ésto ya nos cargará el login de Google, pero es necesario mencionar que firebase únicamente trabaja con localhost en lugar de 127.0.0.1 , es decir así http://localhost:8000/login/

Actualización Login con Google <== Ésta clase ya se la realizó pero en youtube

Token de Firebase en un template Django

Para que funcione correctamente se cambió el signInWithPopup por el signInWithRedirect debido a que arrojaba error

ApiView Google Login View

Una vez obtenido un token necesitamos crear un servicio que reciba el mismo, que como se vió en la parte teórica lo descifre para indetificar al usuario según lo que nos está enviando Google y en caso de que no exista registrarlo y en caso de que no exista recuperarlo.

Para ello se crea un serializador donde solamente se reciba un token y también se crea una vista a la cual voy a llamar normalmente a mi serializador, sin embargo que tipo de vista uso?

Pues para éste caso usaremos la APIView , vista de DRF

Ahora dentro de ésta vista nos tocaría escribir una función de tal manera que se procesa toda la información para el propósito que queremos que es descencriptar el token para saber qué usuario está encriptado