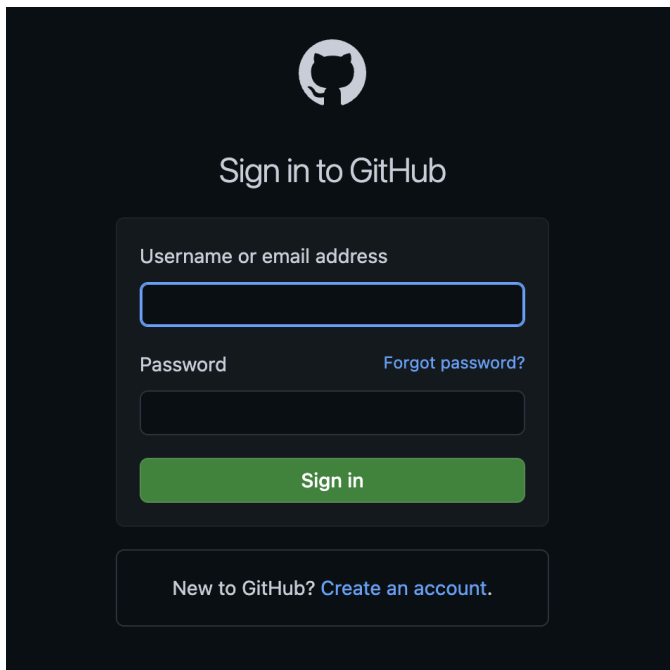


Autenticación, ¿qué es?

La autenticación en los servicios web se refiere al proceso de verificar la identidad de un usuario que está intentando acceder a un recurso protegido en un sistema web. En otras palabras, la autenticación es un mecanismo de seguridad que permite al sistema web verificar la identidad de un usuario antes de otorgarle acceso a ciertos recursos o funcionalidades.

Para autenticar a un usuario en un servicio web, el sistema generalmente solicita un nombre de usuario y una contraseña. Una vez que el usuario proporciona sus credenciales, el sistema web verifica si son válidos y si corresponden a un usuario registrado en su base de datos. Si las credenciales son correctas, el sistema web otorga al usuario acceso a los recursos protegidos.

A screenshot of the GitHub sign-in page. It features the GitHub Octocat logo at the top, followed by the text 'Sign in to GitHub'. Below this is a form with two input fields: 'Username or email address' and 'Password'. To the right of the password field is a link that says 'Forgot password?'. A green 'Sign in' button is positioned below the password field. At the bottom of the form is a link that says 'New to GitHub? Create an account.'.

Existen otros métodos de autenticación además de las credenciales de usuario y contraseña, como el uso de certificados digitales, tokens de seguridad y sistemas de autenticación basados en biometría. En cualquier caso, la autenticación es una parte fundamental de cualquier sistema web que maneje información sensible o privada.

¿Cómo se implementa en Node.js?

En Node.js, puedes utilizar la biblioteca **jsonwebtoken** para crear y verificar tokens JWT.

Primero, debemos instalar la librería sequelize a través de npm:

```
JavaScript  
npm install jsonwebtoken
```

Para crear un token JWT, puedes utilizar la siguiente sintaxis:

```
JavaScript  
const jwt = require('jsonwebtoken');  
  
const payload = { usuario: 'johnny', password: 'J0hnn1' };  
const secret = 'claveSecretaQueSoloElServerConoce';  
  
const token = jwt.sign(payload, secret, { expiresIn: '1h' });
```

En este ejemplo, se define el objeto **payload** que contiene la información que se desea transmitir. En este caso, es el nombre del **usuario** y el **password**. También se define la clave secreta **secret** que se utilizará para codificar el token. Por último, se llama al método **jwt.sign()** para crear el token.

Hay un argumento de la función `expiresIn: '1h'` que define cuánto tiempo tiene validez el token. En este caso, la librería tiene la opción de controlar el tiempo de expiración.

Este token, una vez creado, hay que validarlo, en el servidor que esté manejando los requests. Para verificar un token JWT, puedes utilizar la siguiente sintaxis:

JavaScript

```
const jwt = require('jsonwebtoken');

const token =
'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c3Vhcm1vIjoiam9obm55IiwicGFzc3dvcmQiOiJKMGhubjEifQ.KB-nZtU9M_ufMhv9PfWe4MsNqL585BXnIrH89mq94Gw';
const secret = 'claveSecretaQueSoloElServerConoce';

jwt.verify(token, secret, (err, decoded) => {
  if (err) {
    console.error(err);
    return;
  }

  console.log(decoded);
});
```

En este ejemplo, se define el token como la variable **token** que se desea verificar. También se define la clave secreta **secret** que se utilizará para descodificar el token. Por último, se llama al método **jwt.verify()** para verificar el token. En caso de que el token no sea válido, el objeto **err** tendrá un Error y podremos hacer algo al respecto.

Si es correcto, entonces en el objeto **decoded** tendremos la información que se encuentra en el **payload** que creamos. En nuestro caso, un objeto con el **usuario** y el **password**.

Qué es Passport?

Passport es una biblioteca de autenticación para Node.js que se utiliza para autenticar a los usuarios en una aplicación web. Soporta varios tipos de autenticación, incluyendo nombre de usuario y contraseña, autenticación de terceros como Google o Facebook, y tokens como JWT.

En el contexto de JWT, Passport puede utilizarse para validar tokens JWT y autenticar a los usuarios. Para hacer esto, se utiliza un strategy (estrategia) de Passport para JWT.

¿Cómo se implementa en Node.Js?

Primero, instalamos las dependencias con NPM

JavaScript

```
npm install passport passport-jwt jsonwebtoken
```

Luego, se define la estrategia de Passport para JWT en el archivo de configuración de Passport. En este ejemplo, se busca el token JWT en la cabecera Authorization:

JavaScript

```
const passport = require('passport');
const passportJWT = require('passport-jwt');
const JWTStrategy = passportJWT.Strategy;
const ExtractJWT = passportJWT.ExtractJwt;
const secret = 'claveSecretaQueSoloElServerConoce';

passport.use(new JWTStrategy({
  jwtFromRequest: ExtractJWT.fromAuthHeaderAsBearerToken(),
  secretOrKey: secret
}, (jwtPayload, done) => {
  if (tokenNoEsValido) {
    return done(null, false, { message: 'El token no es válido'
  });
})
```



```

if (usuarioNoExiste) {
  return done(null, false, { message: 'El usuario no existe'
});
}
if (passwordIncorrecto) {
  return done(null, false, { message: 'Autenticacion invalida'
});
}
return done(null, usuario);
});

```

En este ejemplo, se utiliza el método **fromAuthHeaderAsBearerToken()** de **ExtractJWT** para buscar el token JWT en la cabecera Authorization. También se define la clave secreta **secret** que se utilizará para validar el token. En la función de verificación, se debe validar el JWT y buscar el usuario en la base de datos. Dentro de esta función tendremos el **payload** decodificado que en nuestro caso es un objeto que tiene **usuario** y **password**.

Podemos ver también otros tipos de validación, como si el usuario existe o el token no es válido.

Si la combinación usuario y contraseña son correctas, se llama a **done(null, usuario)**; de lo contrario, se llama a **done(null, false)**.

Ahora necesitamos un endpoint que nos devuelva el JWT para que los usuarios se puedan autenticar en el sistema

JavaScript

```

const express = require('express');
const jwt = require('jsonwebtoken');
const router = express.Router();

router.post('/login', (req, res) => {
  const { usuario, password } = req.body;

  // Validar el usuario y la contraseña
  if (usuario === 'usuario' && password === 'password') {
    // Crear el token JWT
    const token = jwt.sign({ usuario: usuario }, secret);

```

```
res.json({ token: token });  
} else {  
  res.status(401).json({ mensaje: 'Autenticación fallida' });  
}  
});  
  
module.exports = router;
```

En este ejemplo, creamos un endpoint POST /login que recibe las credenciales del usuario en el cuerpo de la solicitud (usuario y contraseña). Luego, validamos las credenciales y, si son válidas, creamos un token JWT utilizando el método **jwt.sign** de la biblioteca **jsonwebtoken** y lo enviamos como respuesta en un objeto JSON. Si las credenciales no son válidas, devolvemos una respuesta con un estado de 401 no autorizado y un mensaje de error. El token se firma utilizando la clave secreta definida anteriormente.

Ahora, nos queda usar el middleware proporcionado por Passport: **passport.authenticate** para proteger la ruta deseada:

```
JavaScript  
// Ruta protegida  
app.get('/rutaProtegida', passport.authenticate('jwt', { session:  
false })), (req, res) => {  
  res.send('Esta es una ruta protegida');  
});
```

En este ejemplo, la ruta /rutaProtegida está protegida y solo se puede acceder si el usuario proporciona un token JWT válido en la cabecera Authorization. La ruta /login es la encargada de autenticar al usuario y devolverle el token JWT para que lo use en las solicitudes posteriores.