Autenticación y Autorización

...

JSON Web Token

Autenticación

La autenticación se refiere al proceso de verificar la identidad de un usuario o sistema para asegurarse de que son quienes dicen ser. En el contexto del desarrollo de software, esto generalmente implica la verificación de las credenciales, como un nombre de usuario y una contraseña, un token de autenticación, una huella digital o una tarjeta inteligente.

Autenticación. Funcionamiento

1. Ingreso de Credenciales: El usuario o sistema proporciona sus credenciales al sistema. Esto puede incluir un nombre de usuario y una contraseña o algún otro tipo de identificación.
2. Verificación de Credenciales: El sistema verifica las credenciales proporcionadas por el usuario comparándolas con las credenciales almacenadas en una base de datos segura. Si las credenciales coinciden, se considera que la autenticación es exitosa.
3. Generación de Tokens (Opcional): En algunos casos, después de la autenticación exitosa, se genera un token de autenticación. Este token se utiliza para identificar al usuario en futuras solicitudes sin que el usuario tenga que volver a proporcionar sus credenciales. Los tokens pueden ser JWT (JSON Web Tokens), cookies seguras, o cualquier otro mecanismo seguro.

Autorización

La autorización se refiere al proceso de determinar si un usuario autenticado tiene permiso para realizar una acción o acceder a un recurso específico. Esto implica definir qué acciones pueden realizar los usuarios autenticados en función de sus roles y permisos.

Autorización. Funcionamiento

1. Roles y Permisos: El sistema define roles de usuario y asigna permisos específicos a estos roles. Por ejemplo, un sistema puede tener roles como "Usuario estándar" y "Administrador", con permisos asociados a cada uno.
2. Asignación de Roles: Cuando un usuario se autentica, se le asigna un rol específico basado en su identidad. Por ejemplo, un usuario puede ser asignado al rol "Usuario estándar".
3. Verificación de Permiso: Cuando un usuario intenta realizar una acción o acceder a un recurso, el sistema verifica si el usuario tiene los permisos adecuados para realizar esa acción. Esto se hace comprobando el rol del usuario y los permisos asociados a ese rol.
4. Control de Acceso: Si el usuario tiene los permisos necesarios, se le permite realizar la acción o acceder al recurso. De lo contrario, se le niega el acceso y se le muestra un mensaje de error o se redirige a una página de acceso denegado.

JSON Web Token

JSON Web Token (abreviado JWT) es un estándar abierto basado en JSON propuesto por IETF (RFC 7519) para la creación de tokens de acceso que permiten la propagación de identidad y privilegios o claims en inglés. Por ejemplo, un servidor podría generar un token indicando que el usuario tiene privilegios de administrador y proporcionarlo a un cliente. El cliente entonces podría utilizar el token para probar que está actuando como un administrador en el cliente o en otro sistema. El token está firmado por la clave del servidor, así que el cliente y el servidor son ambos capaces de verificar que el token es legítimo. Los JSON Web Tokens están diseñados para ser compactos, poder ser enviados en las URLs -URL-safe- y ser utilizados en escenarios de Single Sign-On (SSO). Los privilegios de los JSON Web Tokens puede ser utilizados para propagar la identidad de usuarios como parte del proceso de autenticación entre un proveedor de identidad y un proveedor de servicio, o cualquiera otro tipo de privilegios requeridos por procesos empresariales.

En base a https://es.wikipedia.org/wiki/JSON_Web_Token

Estructura

Los JSON Web Tokens generalmente están formados por tres partes: un encabezado o header, un contenido o payload, y una firma o signature5. El encabezado identifica qué algoritmo fue usado para generar la firma y normalmente se ve de la siguiente forma:

header = '{"alg":"HS256","typ":"JWT"}'

HS256 indica que este token está firmado utilizando HMAC-SHA256.

El contenido contiene la información de los privilegios o claims del token:

payload = '{"loggedInAs":"admin","iat":1422779638}'

El estándar sugiere incluir una marca temporal o timestamp en inglés, llamado iat para indicar el momento en el que el token fue creado.

Estructura

La firma está calculada codificando el encabezamiento y el contenido en base64url, concatenándose ambas partes con un punto como separador:

key = 'secretkey'

unsignedToken = encodeBase64Url(header) + '.' + encodeBase64Url(payload)

signature = HMAC-SHA256(key, unsignedToken)

En el token, las tres partes -encabezado, contenido y firma- están concatenadas utilizando puntos de la siguiente forma:

token = encodeBase64Url(header) + '.' + encodeBase64Url(payload) + '.' + encodeBase64Url(signature)
# token es:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJsb2dnZWRJbkFzIjoiYWRtaW4iLCJpYXQiOjE0MjI3Nzk2Mzh9.gzSraSYS8EXBxLN_oWnFSRgCzcmJmMjLiuyu5CSpyHI

Ejemplo de implementación en backend node.js

Instlación, en base a https://www.npmjs.com/package/jsonwebtoken

npm install jsonwebtoken

Para importarlo al proyecto:

import jsonwebtoken from 'jsonwebtoken';

Ejemplo autenticación en memoria en backend

```
const usuarios = [{email:'admin@test.com',password:'1234',rol:'admin'},
                  {email:'user@test.com',password:'1234',rol:'user'} ]
app.post('/login', async (req,res) => {
  console.log(req.body);
  if( req.body ) {
    const indice = usuarios.findIndex(e=>e.id==req.params.id)
    if(indice >= 0) {
      const usuario = usuarios[indice];
      if ( req.body.password == usuario.password) {
          const token =
          jsonwebtoken.sign({email:usuario.usuario,rol:'adm'},'clave_secreta')
          res.json({token:token})
      } else { res.sendStatus(401);  }
    } else { res.sendStatus(401);  }
  } else { res.sendStatus(400);  }
})
```

Login en front-end - Store

```
import { defineStore } from 'pinia'
import axios from 'axios'

export const loginStore = defineStore('login', {
    state: () => {
        return {
            usuario: {},
            estaLogeado : false
        }
    },
```

Login en front-end - Store

```
actions: {
    async login(usuario) {
        try {
            const datos = await axios.post("http://localhost:3000/login",usuario);
            console.log(datos);
            if(datos.status == 200) {
                this.estaLogeado = true;
                this.usuario.email = usuario.email;
                localStorage.setItem('usuario',JSON.stringify(
                    {email:usuario.email, token: datos.data.token}) )
            } else {
                this.estaLogeado = false;
            }
        } catch(e) {
            console.log(e);
        }
    },
```

Ocultando funciones en el menu

```
const store1 = loginStore();
 const { usuario, estaLogeado } = storeToRefs(store1);
 const {logout} = store1;


  <RouterLink to="/system" v-if="estaLogeado" >System  |</RouterLink>
   <RouterLink to="/login" v-if="!estaLogeado" >Login  |</RouterLink>
```

Login en front-end - Vista

1- Crear una vista Login.vue, llamando a los métodos de loginStore.js

Autorizando rutas, en routes/index.js

```
    meta: { RequireAuth: true}

router.beforeEach( (to,from,next) => {
  const usuarioLog = localStorage.getItem('usuario')
  if( to.matched.some(r => r.meta.RequireAuth) && !usuarioLog ) {
    next('/')
  }
  next()
})
```

Enviando token en las peticiones

En el archivo listaService.js, agregamos:

```
apiClient.defaults.headers.common['authorization'] =
`Bearer ${ JSON.parse(localStorage.getItem('usuario')).token }`
```

En el backend, para verificar el envio del token en la cabecera:

```
console.log(req.headers['authorization']);
```

Verificando el token en backend

```
if(req.headers['authorization']!==undefined) {
    const bearerToken = req.headers['authorization'];
    const bearer = bearerToken.split(' ');
    const token = bearer[1];
    jsonwebtoken.verify(token,'clave_secreta',(err:any, payload:any) => {
        if(err) {
            res.sendStatus(401);
        } else {
            res.json(lista);
        }
    })
 } else {
    res.sendStatus(401);
 }
```