
SENPAI
ACADEMY

Autenticación de usuarios

SENPAI
ACADEMY

Node.js - Autenticación de usuarios

Autenticación refiere a validar la identidad del usuario, por ejemplo a partir de un nombre de usuario/id y contraseña.

El sistema verifica que el usuario es quien dice ser usando las credenciales.

En una aplicación web, una vez que autenticamos al usuario, podemos mantener la información del usuario en sesión o local storage para no enviar las credenciales en cada request.

SENPAI ACADEMY

Node.js - Autenticación de usuarios

Para soportar autenticación debemos contar con:

- Funcionalidad de Registro de usuarios (o algunos usuarios hardcodeados)
- Funcionalidad de Login de usuarios (para autenticarse)
- Manejar de alguna forma los usuarios que ya se autenticaron.. para esto podemos usar **JWT (JSON Web Token)**

- Cuando un usuario envíe sus credenciales en el Login, vamos a devolverle un JWT.
- Las siguientes requests del mismo usuario van a enviar el mismo JWT (puede ser como un header, o como parte del Body de la request).
- En la request vamos a validar que el JWT sea válido, y no haya expirado.

SENPAI ACADEMY

Node.js - Registro de Usuarios

Queremos permitir que un nuevo usuario se registre en nuestro sistema.

Para esto podemos crear un router en ***'/auth'*** que contenga un manejador del verbo **POST** en la ruta ***'auth/register'***

Idealmente debemos utilizar una Base de Datos donde insertaremos el nuevo registro. Por ahora, podemos almacenarlo en un array de usuarios.

SENPAI
ACADEMY

```
// En index.js
const express = require('express');
const path = require('path');
const cors = require('cors');
const bodyParser = require('body-parser');

const authRouter = require('./routes/auth');

const app = express();
const PORT = process.env.PORT || 3000;

app.use(express.static(path.join(__dirname, "public")));
// parse application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({ extended: false }));
// parse application/json
app.use(bodyParser.json());
// Solo en desarrollo
app.use(cors());

app.use('/auth', authRouter);

app.get('/ping', function (req, res) {
  return res.send('pong');
});

app.listen(PORT, function () {
  console.log(`El servidor quedo corriendo en el puerto ${PORT}`);
});
```

```
// En routes/auth.js
const express = require('express');
const router = express.Router();

const usuarios = [];

// Registro de un usuario
router.post('/register', (req, res) => {
  // La contraseña la vamos a querer encriptar luego
  const password = req.body.password;

  const newUser = {
    name: req.body.name,
    mail: req.body.mail,
    password: password
  }
  usuarios.push(newUser);

  res.json({ success: true, newUser, usuarios });
});

module.exports = router;
```

Node.js - Registro de Usuarios

Para mejorar la seguridad, las contraseñas deben almacenarse en la base de datos encriptadas. De esta manera, si un atacante logra obtener a la base de datos, no podrá conocer las contraseñas.

Para lograr esto usaremos el módulo ***bcrypt*** para crear un hash de la password al momento de creación de usuario.

Cuando un usuario ingrese con Login, vamos a usar ***bcrypt*** para comparar la contraseña que se ingresa versus el hash almacenado en la base.

```
npm install bcrypt
```

SENPAI ACADEMY

```
// En routes/auth.js
const express = require('express');
const bcrypt = require('bcrypt');

const router = express.Router();

const usuarios = [];

// Registro de un usuario
router.post('/register', async (req, res) => {

  // hash contraseña
  const salt = await bcrypt.genSalt(10);
  const password = await bcrypt.hash(req.body.password, salt);

  const newUser = {
    name: req.body.name,
    mail: req.body.mail,
    password: password
  }
  usuarios.push(newUser);

  res.json({ success: true, newUser, usuarios });
});

module.exports = router;
```

SENPAI
ACADEMY

Node.js - Login de Usuarios

Para un Login de usuarios, vamos a seguir el mismo patrón que con el registro.

Dentro del mismo router ***'/auth'***, agregamos otro manejador del verbo **POST** en la ruta ***'auth/login'*** (podría estar en otro router también)

El handler del **POST** de login debe autenticar al usuario, es decir verificar que las credenciales son correctas (usuario y contraseña), y brindarle una forma de que en las siguientes requests pruebe que está autorizado (JWT)

SENPAI
ACADEMY

```
// En routes/auth.js
router.post('/login', async (req, res) => {

  // Buscamos el usuario con el mismo mail
  const user = usuarios.find((u) => u.mail === req.body.mail);
  if (!user) {
    return res.status(400).json({ error: 'Usuario no encontrado' });
  }

  const validPassword = await bcrypt.compare(req.body.password, user.password);
  if (!validPassword) {
    return res.status(400).json({ error: 'Contraseña no válida' });
  }

  res.json({
    error: null,
    data: 'Login exitoso'
  });
});
```

SENPAI
ACADEMY

Autorización de usuarios

SENPAI
ACADEMY

Node.js - Autorización de usuarios

La **autorización** ocurre luego de que la identidad de un usuario haya sido validada. La autorización verifica que el usuario actual tenga permisos sobre el recurso que intenta acceder.

Como ejemplo, si bien podemos autenticar a usuario en un sistema, podemos restringir el acceso a secciones de administración solo para usuarios que sean administradores del sistema.

En las requests, vamos a poder saber que usuario es (porque ya se autenticaron), y decidir si tiene permisos o no para la ruta que accede.

Saber que usuario es, lo podemos validar con JWT por ejemplo, porque el token puede contener la información que necesitemos sobre el usuario.

SENPAI
ACADEMY

Node.js - JSON Web Token

JWT (JSON Web Token): https://jwt.io/

1. Es un token de seguridad que nosotros creamos al momento que el usuario se registra con sus credenciales.
2. Este token se devuelve al cliente el cual tendrá que enviar cada vez que solicita información al servidor.
3. Se divide en 3 partes: **Header**, **Payload** y **Verify Signature**

En la práctica, se trata de una cadena de texto que tiene tres partes codificadas en Base64, cada una de ellas separadas por un punto:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKx
wRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

SENPAI ACADEMY

Node.js - JSON Web Token

Header: encabezado donde se indica, al menos, el algoritmo y el tipo de token, que en el caso del ejemplo anterior era el algoritmo HS256 y un token JWT.

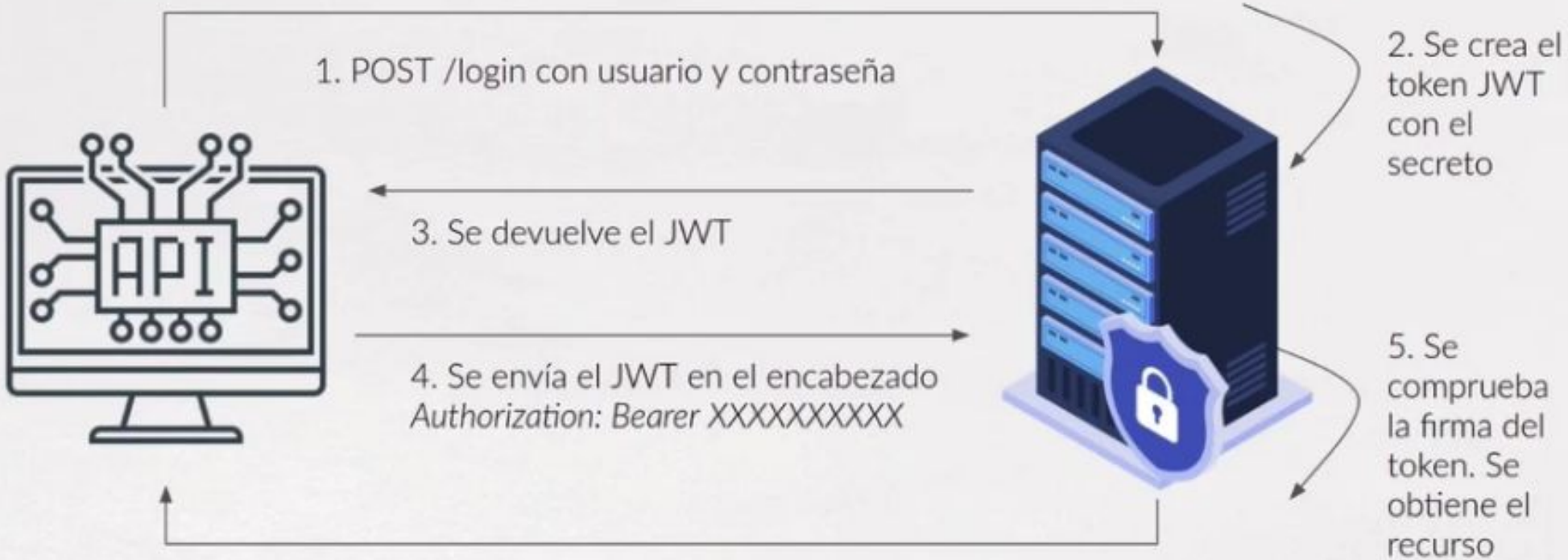
Payload: donde aparecen los datos de usuario y privilegios, así como toda la información que quieramos añadir, todos los datos que creamos convenientes.

Signature: una firma que nos permite verificar si el token es válido.

Node.js - JSON Web Token

OpenWebinars

Ciclo de vida de un token JWT



SENPAI ACADEMY

Node.js - Autorización de usuarios

Primero instalamos la librería de ***jsonwebtoken*** para poder crear y validar JWT desde nuestro código:

```
npm install jsonwebtoken
```

Luego, en el manejador del Login, si validamos el mail y la contraseña, le devolvemos el Token JWT en la respuesta

SENPAI ACADEMY

```
// En routes/auth.js
const express = require('express');
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');
const { verifyToken, TOKEN_SECRET } = require('../middlewares/validate-jwt');
const router = express.Router();

// En routes/auth.js
router.post('/login', async (req, res) => {

  // Buscamos el usuario con el mismo mail
  const user = usuarios.find((u) => u.mail === req.body.mail);
  if (!user) {
    return res.status(400).json({ error: 'Usuario no encontrado' });
  }
  const validPassword = await bcrypt.compare(req.body.password, user.password);
  if (!validPassword) {
    return res.status(400).json({ error: 'Contraseña no válida' });
  }

  // Crear el token
  const token = jwt.sign({
      name: user.name,
      id: user.id
    }, TOKEN_SECRET);

  res.json({ error: null, data: 'Login exitoso', token });
});
```

SENPAI ACADEMY

Node.js - Autorización de usuarios

Luego vamos a crear un Middleware (para poder reutilizarlo) que valide el Token en las requests que llegan, para determinar si es una request que viene de un usuario autorizado.

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

const TOKEN_SECRET = 'UnaClaveParaFirmarelToken';

// middleware to validate token (rutas protegidas)
const verifyToken = (req, res, next) => {
  const token = req.header('auth-token')
  if (!token) {
    return res.status(401).json({ error: 'Acceso denegado' })
  }

  try {
    const verified = jwt.verify(token, TOKEN_SECRET)
    req.user = verified
    next() // continuamos
  } catch (error) {
    res.status(400).json({error: 'El Token no es válido'})
  }
}

module.exports = {
  verifyToken,
  TOKEN_SECRET
};
```

SENPAI ACADEMY

Node.js - Autorización de usuarios

Ahora agregamos el middleware a las rutas que queramos requerir que sólo accedan usuarios autentificados.

Por el middleware de ***verifyToken*** que creamos, en ***req.user*** tendremos los datos de que usuario realiza la request, y podemos decidir si está autorizado o no.

```
// En routes/auth.js
const express = require('express');
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');
const { verifyToken, TOKEN_SECRET } = require('../middlewares/validate-jwt');
const router = express.Router();

//Listar usuarios solo puede ser consumida por alguien autorizado
router.get('/usuarios', verifyToken, async (req, res) => {

  // Podemos acceder a los datos del usuario que hizo la request
  // Segun el JWT que envio en los headers de la request
  console.log(req.user);

  res.json({ error: null, usuarios });
});
```

SENPAI ACADEMY

SENPAI
ACADEMY

gustavguez

gustavguez

GUSTAVO RODRIGUEZ

FULL STACK DEVELOPER
SOLCRE