<codoa codo/>

Agencia de Aprendizaje a lo largo de la vida

FULL STACK FRONTEND
Clase 31

Node 7

BA

Les damos la bienvenida

Vamos a comenzar a grabar la clase

Clase 30

Node

- MULTER
  Instalación y Configuración
  diskStorage y filename
  Subiendo archivos
  Probando desde Postman

Clase 31

Node

- Autenticación
  JWT
  Librerias jsonwebtoken, y bcryptsjs
  Estructura posible del proyecto
  Probando en POSTMAN

Clase 31

Node

Autenticación con JWT

¿Qué es el proceso de Autenticación?

La **autenticación** es el proceso de verificar la identidad de un usuario, dispositivo o sistema para garantizar que solo las personas autorizadas accedan a los recursos y servicios protegidos.

Fases del proceso de autenticación

Identificación:
El usuario proporciona una identidad, como un nombre de usuario o un ID.

Verificación:
El usuario prueba su identidad mediante una contraseña, un token o biometría.

Métodos Comunes de Autenticación:

Contraseñas:
El usuario ingresa una contraseña que se compara con una almacenada.

Tokens:
Se utilizan tokens de software o hardware para la verificación.

Biometría:
Utiliza características físicas como huellas dactilares o reconocimiento facial.

¿Qué es un Token?

Un token es una cadena de texto que actúa como un sustituto de datos sensibles. Se utiliza en seguridad informática para autenticar usuarios y otorgar acceso a recursos protegidos.

Los tokens se generan de manera única para cada sesión o transacción y se pueden revocar o expirar después de un periodo de tiempo.

¿Qué es JSON Web Token?

JWT es un estándar abierto para crear tokens de acceso que permiten la comunicación segura entre dos partes.

Se compone de tres partes: Header, Payload y Signature

Estructura de un JWT

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_ad

Header: El Header típicamente consiste de dos partes: el tipo de token, que es JWT, y el algoritmo de cifrado, como HMAC SHA256 o RSA.

Payload: El Payload contiene las reclamaciones (claims). Las reclamaciones son declaraciones sobre una entidad (normalmente, el usuario) y datos adicionales. Hay tres tipos de claims: registered, public, y private claims.

Signature: Para crear la Signature, se toma el Header codificado, el Payload codificado, una clave secreta y el algoritmo especificado en el Header, y se firma.

Flujo de Autenticación con JWT

LOGIN
El usuario ingresa sus credenciales

→

GENERACION DE TOKEN
Si las credenciales son válidas se genera un JWT

→

VERIFICACION
En cada petición, el token se envía y se verifica su validez

Ventajas de Usar JWT

Sin Estado: No requiere mantener sesiones en el servidor.

Escalabilidad: Ideal para aplicaciones distribuidas.

Seguridad: Uso de firmas y cifrado.

Compacto: Eficiente en tamaño, ideal para transmisión en HTTP headers.

Uso de la librería jsonwebtoken
La librería jsonwebtoken se utiliza en Node.js para trabajar con JSON Web Tokens (JWTs). Esta librería proporciona métodos para crear, firmar, verificar y decodificar tokens JWT, facilitando la implementación de autenticación y autorización en aplicaciones web y móviles.

Uso de la librería bcryptjs

La librería bcryptjs se utiliza en Node.js para manejar la seguridad de las contraseñas. Proporciona funciones para cifrar (hash) contraseñas y compararlas de manera segura. Es especialmente útil para aplicaciones que requieren almacenar y verificar contraseñas de usuarios de manera segura.

IMPORTANTE

COMO DESARROLLADORES NUNCA DEBEMOS GUARDAR UNA CONTRASEÑA EN UNA BASE DE DATOS SIN UTILIZAR ALGÚN TIPO DE HASH.

Las contraseñas se guardan hasheadas para evitar que sean usadas si existe un ataque a la base de datos.

Creación del proyecto:

Para crear el proyecto utilizaremos:

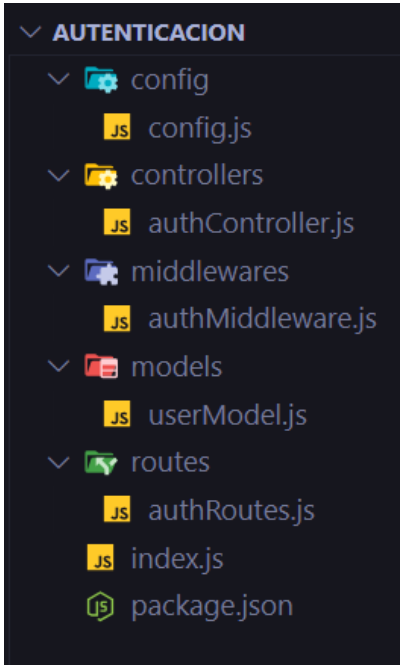
```
npm init -y
```

```
npm install express jsonwebtoken bcryptjs
```

Instalaremos las librerías **express**, **jsonwebtoken** (para manejar los tokens) y **bcryptjs** (para encriptar las contraseñas). Si bien en este ejemplo no las guardaremos en la base de datos, es una buena práctica no tener las contraseñas sin hashear.

Estructura del proyecto:

Y luego crearemos la siguiente estructura:



Crear el servidor: index.js

index.js > ...

```
1   const express = require('express');
2   const authRoutes = require('./routes/authRoutes');
3
4   const app = express();
5   const PORT = process.env.PORT || 3000;
6
7   app.use(express.json());
8
9   app.use('/auth', authRoutes);
10
11  app.listen(PORT, () => {
12      console.log(`Server is running on port ${PORT}`);
13  });
14
```

Configurar el servidor: config.js

Este archivo contiene la configuración necesaria para manejar los **tokens JWT** en la aplicación. Incluye la clave secreta utilizada para firmar los tokens y la duración de validez de los tokens.

secretKey: es una cadena de texto que se usa para firmar y verificar los tokens JWT. **Esta clave debe ser única y secreta.** En un entorno de producción, es importante mantener esta clave segura y no compartirla públicamente.

config > config.js > ...

```
1  module.exports = {
2      secretKey: 'tu_clave_secreta',
3      tokenExpiresIn: '1h'
4  };
```

La clave se usa luego en la función **jwt.sign()** para crear el token y **en jwt.verify()** para verificar la validez del token.

tokenExpiresIn: especifica cuánto tiempo es válido el token antes de expirar. Se puede utilizar 1h, 2h, 30m, etc.

Modelo de usuario:

Crearemos una estructura básica para almacenar usuarios en la aplicación. Este modelo se utiliza para almacenar y manejar los datos de los usuarios de manera temporal en la memoria durante el tiempo de ejecución de la aplicación.

models > JS userModel.js > ...

```
1  const users = [];
2
3  module.exports = users;
4
```

Nota: A los fines de centrar el ejercicio en JWT, almacenaremos los usuarios en memoria. En un proyecto real, los usuarios deben almacenarse en base de datos.

Controlador de autenticación

El controlador de autenticación maneja las solicitudes de registro e inicio de sesión de los usuarios. Realiza las siguientes funciones principales:

- **Registro de usuario (register) :**Recibe los datos del usuario, cifra la contraseña, almacena el usuario en el array de usuarios, genera un token JWT y lo envía como respuesta.

- **Inicio de Sesión (login):** Verifica las credenciales del usuario, genera un token JWT si las credenciales son correctas y lo envía como respuesta.

Controlador de autenticación

Crearemos el archivo controllers/authController.js

Antes de crear la función de **registro** y de **login** debermos importar las **librerías** y **módulos** necesarios:

controllers > authController.js > ...

```
// Importa el módulo jsonwebtoken para manejar JWT
const jwt = require('jsonwebtoken');
// Importa el módulo bcryptjs para cifrar contraseñas
const bcrypt = require('bcryptjs');
// Importa el modelo de usuarios (array de usuarios)
const users = require('../models/userModel');
// Importa la configuración (clave secreta y duración del token)
const config = require('../config/config');

```

Ahora estamos en condiciones de comenzar con la función de registro:

Controlador de autenticación

Función de registro: Creamos la función que permitirá registrar un usuario.

```
// Función para registrar un nuevo usuario
exports.register = (req, res) => {
    // Extrae el nombre de usuario y la contraseña del cuerpo de la solicitud
    const { username, password } = req.body;
    // Cifra la contraseña usando bcrypt
    const hashedPassword = bcrypt.hashSync(password, 8);

    // Crea un nuevo objeto de usuario con un ID único
    const newUser = { id: users.length + 1, username, password: hashedPassword };
    // Agrega el nuevo usuario al array de usuarios
    users.push(newUser);

    // Genera un token JWT para el nuevo usuario
     const token = jwt.sign({ id: newUser.id }, config.secretKey, { expiresIn: config.tokenExpiresIn });
     // Envía el token como respuesta al cliente
     res.status(201).send({ auth: true, token });
};
```

Resumiendo la función register:

1) Extrae el nombre de usuario y la contraseña del cuerpo de la solicitud

2) Cifra la contraseña usando bcrypt.

3) Crea un nuevo usuario con un ID único y la contraseña cifrada.

4) Agrega el nuevo usuario al array de usuarios.

5) Genera un token JWT para el nuevo usuario.

6) Envía el token JWT al cliente.

Controlador de autenticación

Función de inicio de sesión (login):

```
28  // Función para iniciar sesión de un usuario
29  exports.login = (req, res) => {
30      // Extrae el nombre de usuario y la contraseña del cuerpo de la solicitud
31      const { username, password } = req.body;
32      // Busca el usuario en el array de usuarios por nombre de usuario
33      const user = users.find(u => u.username === username);
34      // Si el usuario no se encuentra, devuelve un error 404
35      if (!user) return res.status(404).send('User not found.');
36      // Compara la contraseña proporcionada con la contraseña cifrada almacenada
37      const passwordIsValid = bcrypt.compareSync(password, user.password);
38      // Si la contraseña no es válida, devuelve un error 401
39      if (!passwordIsValid) return res.status(401).send({ auth: false, token: null });
40      // Genera un token JWT usando el ID del usuario
41      const token = jwt.sign({ id: user.id }, config.secretKey, { expiresIn: config.tokenExpiresIn });
42      // Envía el token JWT al cliente con el estado 200 (OK)
43      res.status(200).send({ auth: true, token });
44  };
```

Resumiendo la función login:

1) Extrae el nombre de usuario y la contraseña del cuerpo de la solicitud.

2) Busca el usuario en el array de usuarios por nombre de usuario.

3) Verifica si el usuario existe.

4) Compara la contraseña proporcionada con la contraseña cifrada almacenada.

5) Genera un token JWT si las credenciales son correctas.

6) Envía el token JWT al cliente.

Middleware de Autenticación

El middleware de autenticación verifica el token JWT incluido en las solicitudes a rutas protegidas. Si el token es válido, permite que la solicitud continúe, si no, bloquea el acceso.

middlewares > authMiddleware.js > ...

```
// Importa el módulo jsonwebtoken para manejar JWT
const jwt = require('jsonwebtoken');
// Importa la configuración (clave secreta y duración del token)
const config = require('../config/config');

// Middleware de autenticación
module.exports = (req, res, next) => {
    // Obtiene el token del encabezado de autorización
    const authHeader = req.headers['authorization'];
    // Si no hay token en el encabezado
    if (!authHeader) return res.status(403).send({ auth: false, message: 'No se proveyó un token' });

    // Extrae el token del encabezado (formato "Bearer <token>")
    const token = authHeader.split(' ')[1];
    // Si el token no está bien formado envía una respuest ade erro 403
    if (!token) return res.status(403).send({ auth: false, message: 'Malformed token.' });
    // Verifica el token usando la clave secreta
    jwt.verify(token, config.secretKey, (err, decoded) => {
        // Si hay un error en la verificación envía una respuesta de error 500
        if (err) return res.status(500).send({ auth: false, message: 'Failed to authenticate token.' });

        // Si el token es válido, almacena el ID del usuario decodificado en la solicitud
        req.userId = decoded.id;
        // Llama a la siguiente función de middleware o controlador
        next();
    });
};
```

Resumiendo la función del middleware:

Comprueba si el token está presente. Verifica la validez del token utilizando la clave secreta.

Manejo de Errores:
- Si no se proporciona un token, devuelve un error 403.
- Si el token es inválido, devuelve un error 500.

Almacenamiento de Datos:
Si el token es válido, almacena el ID del usuario decodificado en la solicitud.

Continuación del Proceso:
Permite que la solicitud continúe hacia el siguiente middleware o controlador.

Este middleware asegura que solo las solicitudes con un token JWT válido puedan acceder a las rutas protegidas de la aplicación.

Rutas de autenticación

routes > authRoutes.js > ...

```
// Importa el módulo express
const express = require('express');
// Importa el controlador de autenticación
const authController = require('../controllers/authController');
// Importa el middleware de autenticación
const authMiddleware = require('../middlewares/authMiddleware');

// Crea un nuevo enrutador de Express
const router = express.Router();

// Ruta para registrar un nuevo usuario
router.post('/register', authController.register);
/* Cuando se realiza una solicitud POST a /auth/register, se ejecuta
 la función register del controlador de autenticación */

// Ruta para iniciar sesión de un usuario
router.post('/login', authController.login);
/* Cuando se realiza una solicitud POST a /auth/login,
se ejecuta la función login del controlador de autenticación */

// Ruta protegida de ejemplo
router.get('/protected', authMiddleware, (req, res) => {
    res.status(200).send(`Hello user ${req.userId}`);
});
/* Cuando se realiza una solicitud GET a /auth/protected,
se ejecuta el middleware de autenticación primero Si la autenticación es exitosa,
se ejecuta la función que envía un mensaje de saludo con el ID del usuario */

// Exporta el enrutador para que pueda ser utilizado en otros archivos
module.exports = router;
```

Finalmente, creamos el archivo routes/**authRoutes.js**

Las rutas de autenticación definen cómo se manejan las solicitudes de registro, inicio de sesión y acceso a recursos protegidos.

Resumen de cada ruta:

/register (POST):
Maneja las solicitudes de registro de nuevos usuarios.
Utiliza la función register del authController para registrar usuarios y generar tokens JWT.

/login (POST):
Maneja las solicitudes de inicio de sesión de usuarios.
Utiliza la función login del authController para autenticar usuarios y generar tokens JWT.

/protected (GET):
Una ruta protegida que requiere autenticación.
Utiliza el authMiddleware para verificar el token JWT.
Si el token es válido, envía un mensaje de saludo con el ID del usuario.

Estas rutas permiten a los usuarios registrarse, iniciar sesión y acceder a recursos protegidos de manera segura utilizando JWT para la autenticación.

Levantando el servidor:

Ha llegado el momento de probar nuestro servidor.

En la terminal ejecutaremos:

```
node index.js
```

Y si todo salió bien, veremos:

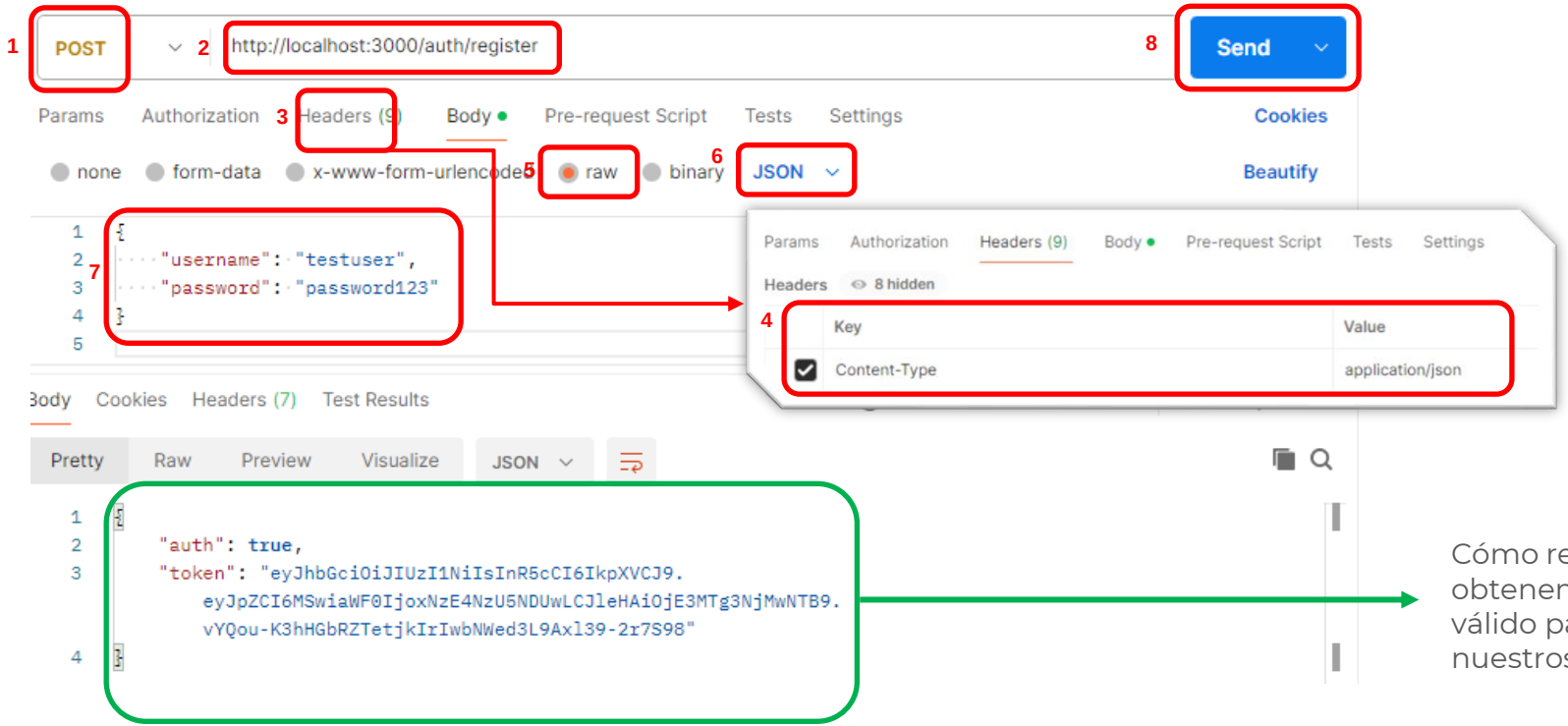
```
Server is running on port 3000
```

Probando con POSTMAN:

Deberemos probar desde el front:

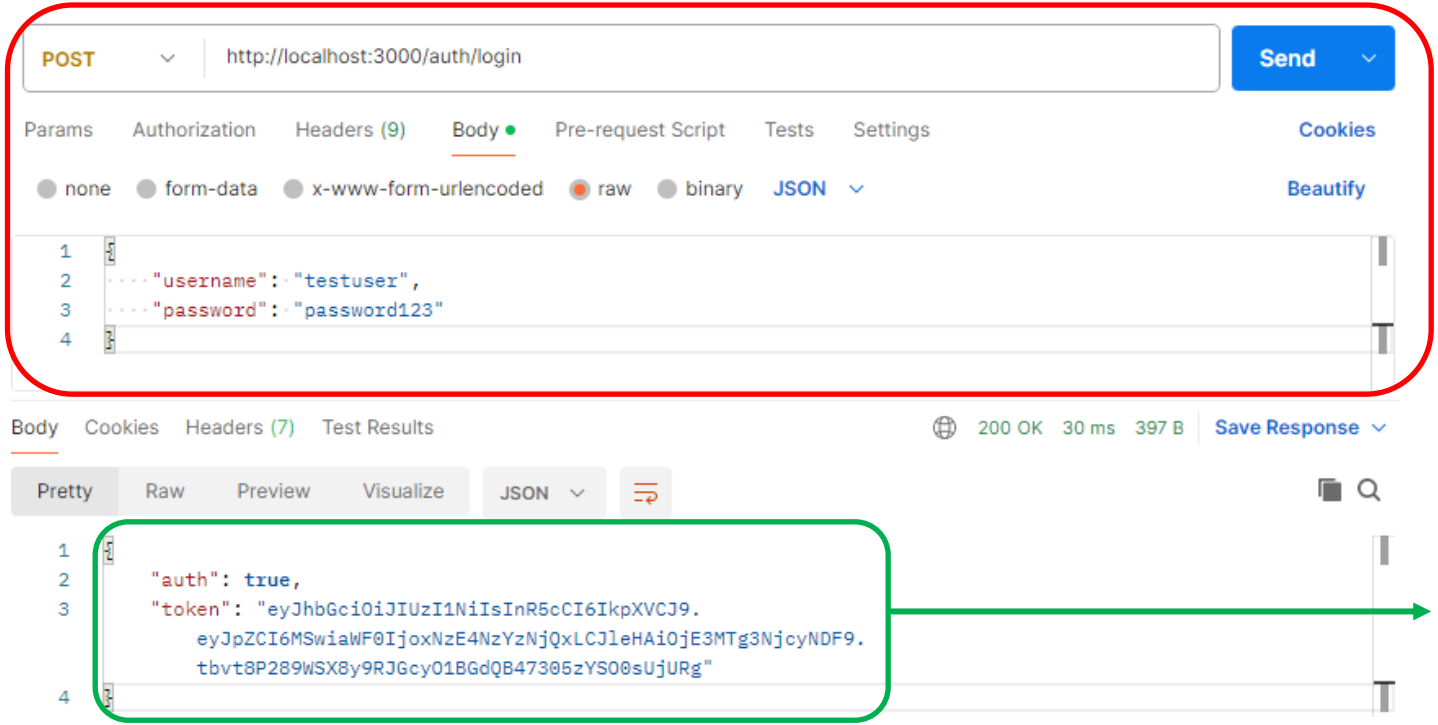
- **Registrar un nuevo usuario**
- **Iniciar una sesión**
- **Acceder a una ruta protegida**

Probando con POSTMAN: Registrar un usuario



Cómo resultado obtenemos un token válido para utilizar en nuestros endpoints

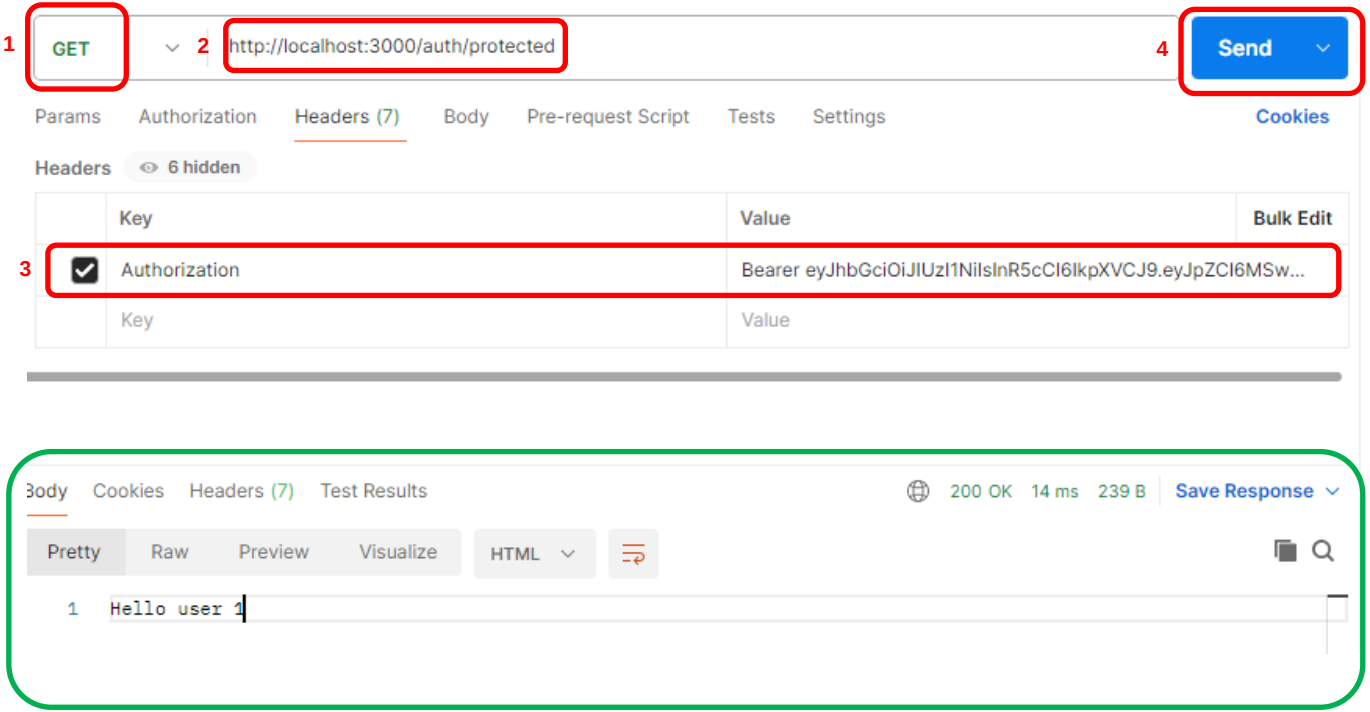
Probando con POSTMAN: Loguear un usuario



Para que el login devuelva un auth = true, deberemos primero ejecutar el register. Recordemos que en este ejemplo, los usuarios se guardan en memoria, por lo tanto deben ser cargados cada vez que se levante el servidor.

Cómo resultado obtenemos un token válido para utilizar en nuestros endpoints

Probando con POSTMAN: Validar una ruta protegida



No te olvides de dar el presente

Recordá:

- Revisar la Cartelera de Novedades.
- Hacer tus consultas en el Foro.
- **Realizá los ejercicios obligatorios.**

Todo en el Aula Virtual.