



Laboratorio de Redes y Sistemas Operativos

Trabajo Práctico: API REST con acceso por JWT

05/12/2019

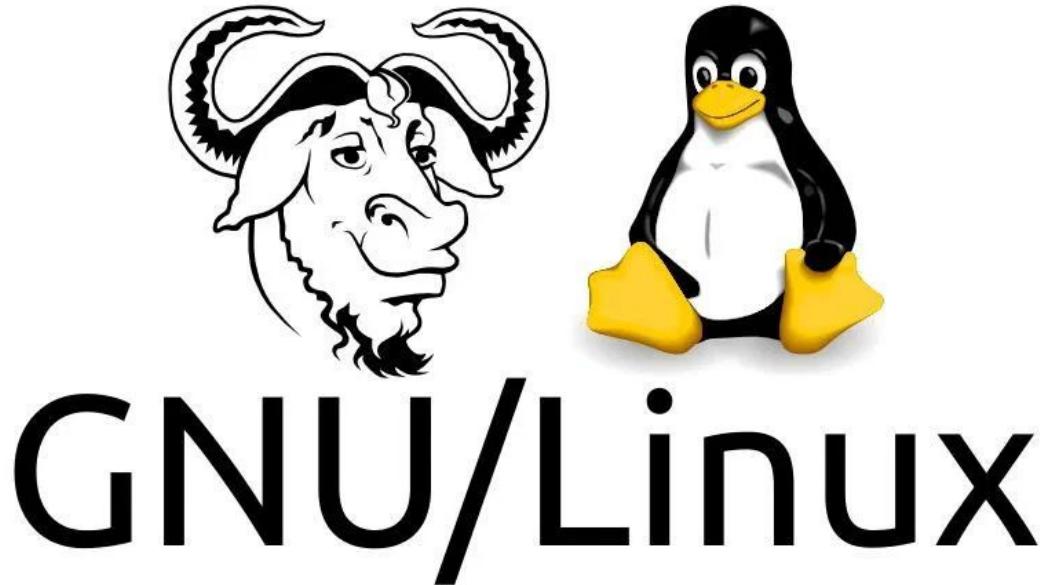
Integrantes:

Platero Leandro
Soñez Mailin

INDICE

Programas y tecnologías usados	3
GNU/Linux	3
Node.js	4
MongoDB	5
Auth0	6
¿Qué es un JSON Web Token?	7
¿Cuándo se usan los JSON Web Tokens?	7
¿Cuál es la estructura del JSON Web Token?	7
Guía para construir y probar la API REST	10
Creación del proyecto de la API con Node.js	12
Prueba de los endpoints de la API	16

Programas y tecnologías usados



GNU/Linux

es un sistema operativo de código abierto que puede ejecutarse en computadoras personales, servidores, máquinas virtuales y otros dispositivos y se utiliza para desarrollar y correr distintos tipos de software: libre, propietario, gratuito, comercial, empresarial, personal, etc. En este trabajo se utiliza y recomienda para instalar y ejecutar el servidor web de una API REST creada con Node.js y el programa Postman para probar el funcionamiento de esta API. Ambos programas son de código abierto.



Node.js

es un entorno de ejecución para programas JavaScript construido con el motor de JavaScript V8 de Chrome. Concebido como un entorno de ejecución de JavaScript orientado a eventos asíncronos, Node.js está diseñado para construir aplicaciones en red escalables.

Esto contrasta con el modelo de concurrencia más común hoy en día, donde se usan hilos del Sistema Operativo. Las operaciones de redes basadas en hilos son relativamente ineficientes y son muy difíciles de usar. Además, los usuarios de Node.js están libres de preocupaciones sobre el bloqueo del proceso, ya que no existe. Casi ninguna función en Node.js realiza I/O directamente, así que el proceso nunca se bloquea. Debido a que no hay bloqueo es muy razonable desarrollar sistemas escalables en Node.js.

Sitio web: <https://nodejs.org>



mongoDB®

MongoDB

es una base de datos de código abierto distribuida, basada en documentos y de uso general que ha sido diseñada para desarrolladores de aplicaciones modernas y para la era de la nube. En este trabajo se usa para almacenar datos de libros, que son los modelos de objetos sobre los que trabaja la API REST.

¿Para qué más sirve MongoDB?

- Guarda los datos en documentos parecidos a JSON.
- Permite filtrar y ordenar por cualquier campo, sin importar que tan anidado esté dentro del documento.
- Las consultas también son JSON, por lo tanto es fácil componerlas. No hace falta concatenar strings para generar consultas SQL.

Sitio web: <https://www.mongodb.com>



Auth0

Auth0

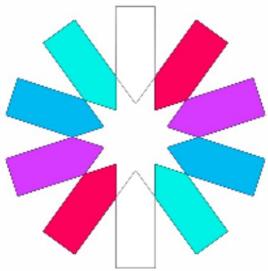
Plataforma que provee servicios de autenticación y autorización para web, mobile y aplicaciones. Cuenta con planes gratuitos y otros comerciales.

¿Para qué sirve Auth0?

- Los usuarios de una app pueden iniciar sesión con su usuario y contraseña o a través de sus redes sociales (como Facebook o Twitter).
- Para construir una API y que sea segura.
- Para implementar SSO (Single Sign-On) / “Inicio de Sesión Unificado” en el caso de tener más de una app y que el usuario pueda acceder a ambas con una sola instancia de identificación.
- Los usuarios pueden iniciar sesión con un código generado en el momento, que se les envía por email o SMS.
- Para bloquear a las IP sospechosas si fallan varias veces al intentar iniciar sesión.
- Para no tener que implementar una propia forma de manejar los usuarios. Crear usuarios, resetear contraseñas, bloquear o borrar usuarios. La API de Auth0 se encarga de todo eso.
- Para usar autenticación multifactor.

En este trabajo se usa Auth0 para generar y validar los JWT que usa la API REST.

Sitio web: <https://auth0.com>



JUNT

¿Qué es un JSON Web Token?

JSON Web Token (JWT) es un standard (RFC 7519) que define una manera compacta para transmitir información de forma segura en forma de un objeto JSON. Esta información puede ser verificada y confiable porque está firmada digitalmente. Los JWTs pueden ser firmados usando una key secreta (algoritmo HMAC) o pública/privada (RSA o ECDSA).

Aunque los JWTs pueden estar encriptados para proveer discreción entre las partes, se centra en los tokens firmados. Éstos verifican la integridad de lo que contiene, mientras que los tokens encriptados lo esconden de terceros. Cuando son asignados, la firma certifica que únicamente el grupo dueño de la key privada es quien firmó.

¿Cuándo se usan los JSON Web Tokens?

- **Autorización:** El caso más común para usar JWT. Una vez que el usuario inició sesión, cada Request que envie incluirá el JWT, permitiéndole acceder a servicios y recursos autorizados sólo para ese token.
- **Intercambio de información:** JWT es una buena manera de transmitir de forma segura información entre grupos. Por el uso de firmas y claves, se puede estar seguro de que quien envía la información es exactamente quien dice ser. También verifica que el contenido no fue manipulado, porque la firma se calcula usando el header y el payload.

¿Cuál es la estructura del JSON Web Token?

JWT consiste de tres partes separadas por puntos (.):

- Header
- Payload
- Signature

Por lo tanto, un JWT típicamente se ve así:

xxxxx.yyyyy.zzzzz

Header

El header (cabecera) típicamente consiste de dos partes: el tipo del token, que es JWT, y el algoritmo para la firma que se está usando, como HMAC, SHA256 o RSA.

Ejemplo:

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

Payload

La segunda parte del token es el payload (carga), que contiene los pedidos. Los pedidos son sentencias o declaraciones sobre una entidad (generalmente, el usuario) e información adicional. Hay tres tipos de pedidos: registrados, públicos y privados.

- **Registrados:** Hay un conjunto de pedidos registrados predefinidos que no son obligatorios pero sí recomendados, para proveer pedidos útiles. Algunos de ellos son: iss (issuer), exp (expiration time), sub (subject), aud (audience) y otros.
- **Públicos:** Éstos pueden definirse a voluntad por los que usan JWTs. Pero para evitar colisiones, deberían estar definidos en IANA JSON Web Token Registry o como una URI resistente a la colisión por espacios de nombres (namespace).
- **Privados:** Son los pedidos personalizados creados para transmitir información entre los grupos que acuerden usarlos y no son ni registrados ni públicos.

Ejemplo de payload:

```
{  
  "sub": "1234567890",  
  "name": "John Doe",  
  "admin": true  
}
```

Aunque esta información esté protegida contra manipulaciones, pueden leerla todos. Por lo que no hay que poner información secreta en el payload o header, excepto que esté encriptada.

Firma

Para crear la firma hay que tomar el header y payload codificados, un secreto, el algoritmo especificado en el header y firmar combinando todo eso.

Por ejemplo, si se quiere usar el algoritmo HMAC SHA256, la firma creada se verá de la siguiente manera:

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  secret)
```

En este trabajo se utiliza JWT para Autorización: Se busca que solamente los usuarios autorizados puedan utilizar la mayoría de los endpoints de la API REST (excepto uno que es público).

Guía para construir y probar la API REST

Introducción

En esta guía se indicarán los pasos para asegurar con JWT algunos endpoints de una API REST que funciona sobre un servidor web creado con Node.js y tiene como objetivo almacenar datos de libros y poder mostrar y enviar estos datos. Para la generación y validación de los JWT Tokens se utilizará el proveedor gratuito Auth0. El servidor web de la API se corre localmente pero podría estar alojado en cualquier hosting ya que es independiente del servicio de autenticación. Como base de datos se utiliza un servidor para desarrollo de MongoDB que corre también localmente y podría reemplazarse por una instancia en otro servidor de ser necesario.

Creación del servicio de autenticación

Crear una cuenta gratuita en <https://auth0.com>

Iniciar sesión e ingresar en la sección APIs del panel izquierdo.



Dashboard

Applications

APIs

Hacer click en el botón Create API.

+ CREATE API

Completar los siguientes datos:

Name: puede ser cualquiera, se usa para identificar en el panel de control del sitio de Auth0.

Identifier: Identificador lógico con estructura de URL, p. ej. <https://laboapi>

Signing Algorithm: Dejar en RS256.

New API X

Name

A friendly name for the API.

Identifier

 →

A logical identifier for this API. We recommend using a URL but note that this doesn't have to be a publicly available URL, Auth0 will not call your API at all. **This field cannot be modified.**

Signing Algorithm

 ▼

Algorithm to sign the tokens with. When selecting RS256 the token will be signed with Auth0's private key.

CREATE CANCEL

Ingresar a la API creada en la sección APIs y en la pestaña Quickstart copiar estos datos:

```
{  
  options.Authority = "https://dev-tcqbd02.auth0.com/";  
  options.Audience = "https://laboapi";  
});
```

options.Authority -> en el archivo index.js el valor va en el atributo issuer.
options.Audience -> en el archivo index.js el valor va en el atributo audience.

Creación del proyecto de la API con Node.js

Instalar la última versión disponible de Node.js con el comando (ejemplo para Fedora):

```
sudo dnf install nodejs
```

Crear y cambiar al directorio del proyecto con los comandos:

```
mkdir laboAPI  
cd laboAPI
```

Inicializar el proyecto e instalar dependencias:

```
npm init -y  
npm install body-parser cors express helmet morgan mongodb-memory-server \  
mongodb express-jwt jwks-rsa
```

Crear el directorio para los archivos fuente:

```
mkdir src
```

Dentro del directorio src crear el archivo index.js con siguiente contenido, que establece los endpoints de la API y cuáles requieren autenticación.

```
// src/index.js  
// Importar dependencias  
const express = require('express');  
const bodyParser = require('body-parser');  
const cors = require('cors');  
const helmet = require('helmet');  
const morgan = require('morgan');  
  
// Definir la aplicación de Express  
const app = express();  
  
const {startDatabase} = require('./database/mongo');  
const {insertBook, getBooks} = require('./database/books');  
const {deleteBook, updateBook} = require('./database/books');
```

```

// Importar dependencias para chequear JWT
const jwt = require('express-jwt');
const jwksRsa = require('jwks-rsa');

// Agregar middlewares
app.use(helmet());
app.use(bodyParser.json());
app.use(cors());
app.use(morgan('combined'));

// Definir endpoints de acceso público
// Obtener todos los libros disponibles
app.get('/', async (req, res) => {
  res.send(await getBooks());
});

// Definir endpoints que requieren autenticación
const checkJwt = jwt({
  secret: jwksRsa.expressJwtSecret({
    cache: true,
    rateLimit: true,
    jwksRequestsPerMinute: 5,
    jwksUri: `https://dev-tcqbd02.auth0.com/.well-known/jwks.json`
  }),
  audience: 'https://laboapi',
  issuer: `https://dev-tcqbd02.auth0.com/`,
  algorithms: ['RS256']
});
app.use(checkJwt);

// Agregar un nuevo libro
app.post('/', async (req, res) => {
  const newBook = req.body;
  await insertBook(newBook);
  res.send({ message: 'New Book inserted.' });
});

```

```

// Eliminar un libro
app.delete('/:id', async (req, res) => {
  await deleteBook(req.params.id);
  res.send({ message: 'Book removed.' });
});

// Actualizar un libro
app.put('/:id', async (req, res) => {
  const updatedBook = req.body;
  await updateBook(req.params.id, updatedBook);
  res.send({ message: 'Book updated.' });
});
// Inicializar la base de datos MongoDB de desarrollo
startDatabase();

// Iniciar el servidor HTTP
app.listen(3001, async () => {
  console.log('listening on port 3001');
});

```

Crear el archivo mongo.js en el directorio src/database con el siguiente contenido utilizado para inicializar la base de datos de desarrollo.

```

// ./src/database/mongo.js
const {MongoMemoryServer} = require('mongodb-memory-server');
const {MongoClient} = require('mongodb');

let database = null;

async function startDatabase() {
  const mongo = new MongoMemoryServer();
  const mongoDBURL = await mongo.getConnectionString();
  const connection = await MongoClient.connect(mongoDBURL, {useNewUrlParser: true});
  database = connection.db();
}

async function getDatabase() {
  if (!database) await startDatabase();
  return database;
}

```

```
module.exports = {
  getDatabase,
  startDatabase,
};
```

Crear el archivo books.js js en el directorio src/database con el siguiente contenido que define las funciones para manejar los endpoints de la API.

```
// ./src/database/books.js
const {getDatabase} = require('./mongo');
const {ObjectId} = require('mongodb');
const collectionName = 'books';

async function insertBook(book) {
  const database = await getDatabase();
  const {insertedId} = await database.collection(collectionName).insertOne(book);
  return insertedId;
}

async function getBooks() {
  const database = await getDatabase();
  return await database.collection(collectionName).find({}).toArray();
}

async function deleteBook(id) {
  const database = await getDatabase();
  await database.collection(collectionName).deleteOne({
    _id: new ObjectId(id),
  });
}

async function updateBook(id, book) {
  const database = await getDatabase();
  delete book._id;
  await database.collection(collectionName).update(
    { _id: new ObjectId(id), },
    {
      $set: {
        ...book,
      },
    },
  );
}
```

```
module.exports = {
  insertBook,
  getBooks,
  deleteBook,
  updateBook,
};
```

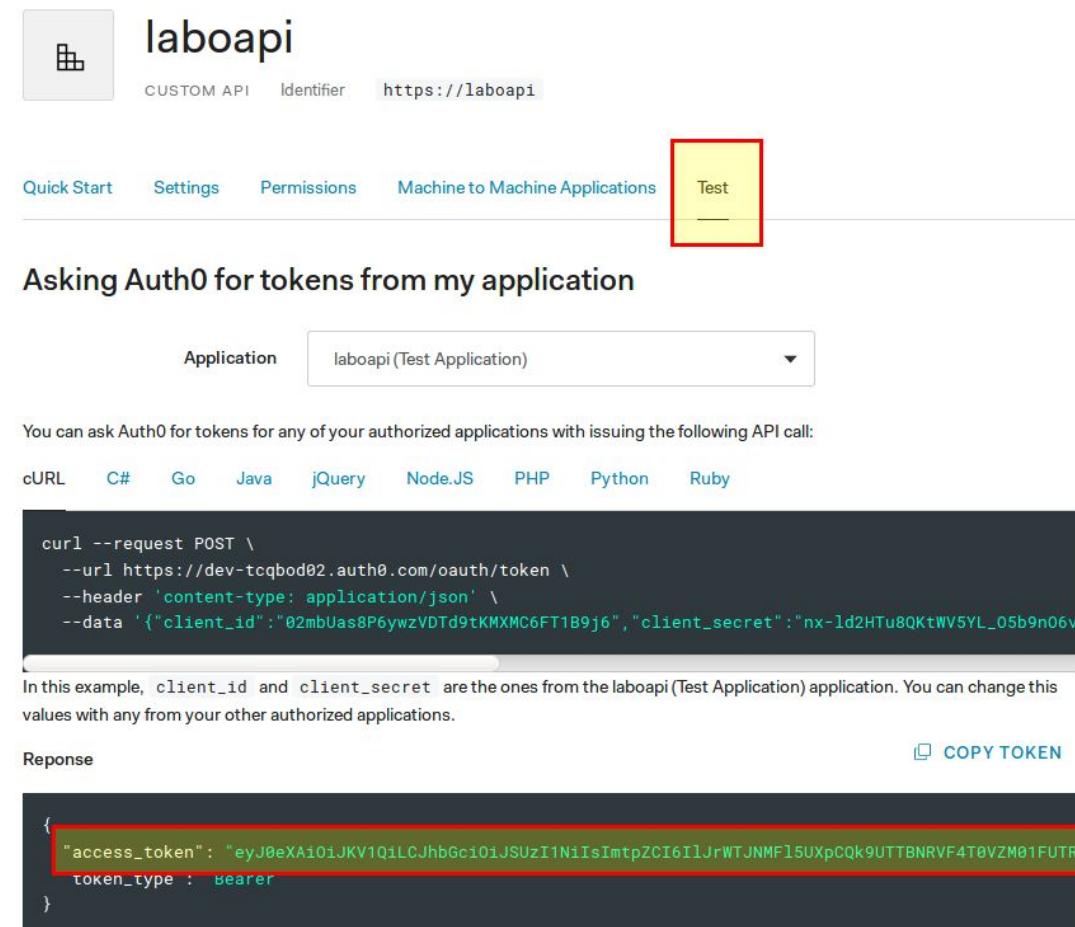
Prueba de los endpoints de la API

Iniciar el servidor de Node

```
cd laboAPI
node src
```

Obtener token para realizar pruebas

Ingresar a Auth0 y en la sección Test de la API copiar el access_token.



laboapi

CUSTOM API Identifier <https://laboapi>

Quick Start Settings Permissions Machine to Machine Applications **Test**

Asking Auth0 for tokens from my application

Application laboapi (Test Application) ▾

You can ask Auth0 for tokens for any of your authorized applications with issuing the following API call:

cURL C# Go Java jQuery Node.js PHP Python Ruby

```
curl --request POST \
--url https://dev-tcqbd02.auth0.com/oauth/token \
--header 'content-type: application/json' \
--data '{"client_id": "02mbUas8P6ywzVDTd9tKMXMC6FT1B9j6", "client_secret": "nx-1d2HTu8QKtWV5YL_05b9n06vN", "grant_type": "client_credentials"}
```

In this example, `client_id` and `client_secret` are the ones from the laboapi (Test Application) application. You can change this values with any from your other authorized applications.

Reponse

COPY TOKEN

```
{ "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6IlJrWTJNMF15UXpCQk9UTTBNRVF4T0VZM01FUTRF", "token_type": "Bearer" }
```

Probar endpoint privado para agregar un nuevo libro con Postman sin usar token

The screenshot shows a Postman request configuration and its response. The request is a POST to `localhost:3001`. The 'Body' tab is selected, showing an empty JSON object `{}`. The response status is `401 Unauthorized`, and the response body is an HTML error page with the message `UnauthorizedError: No authorization token was found`.

POST localhost:3001

Params Authorization Headers (9) Body Pre-request Script Tests Settings

Body (1) `{}`

Body Cookies Headers (12) Test Results Status: 401 Unauthorized Time: 9ms

Pretty Raw Preview Visualize BETA HTML

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="utf-8">
6   <title>Error</title>
7 </head>
8
9 <body>
10  <pre>UnauthorizedError: No authorization token was found<br> &nbsp;at middle...
```

Resultado: Se recibe el error “No authorization token was found” y no se agrega el libro.

Probar endpoint privado para agregar un nuevo libro añadiendo el JWT token a la request

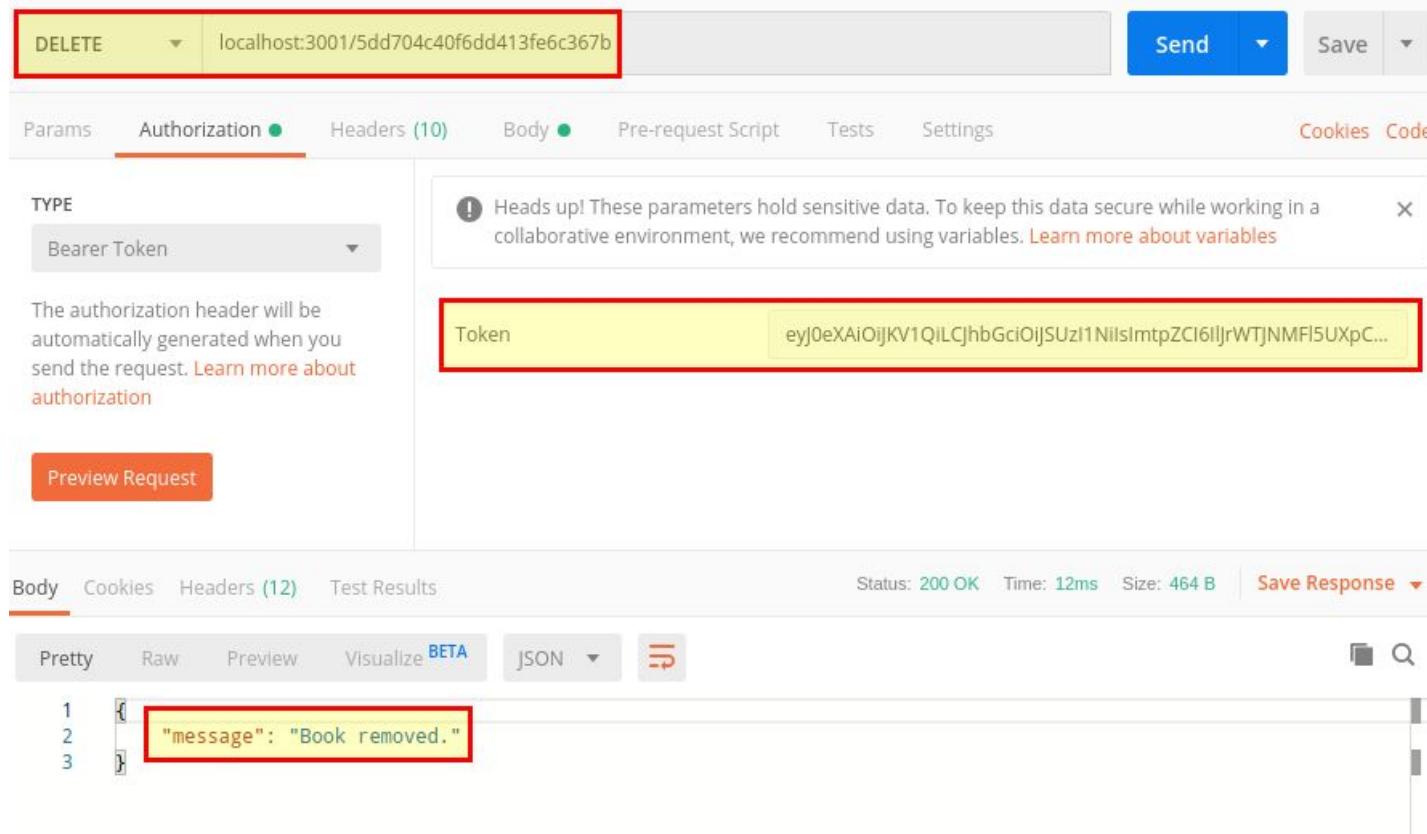
Se utiliza el endpoint / con el método POST y se envían los atributos del libro en el Body de la Request. El Token se setea en la sección Authorization.

The screenshot shows the Postman interface for a POST request to `localhost:3001`. The `Authorization` tab in the header section is highlighted with a red box. The `Body` tab is also highlighted with a red box. The `Body` section contains a `message` field with the value `"New Book inserted."`. The `Authorization` section shows a `TYPE` of `Bearer Token` and a `Token` field containing a long JWT token. The status bar at the bottom shows `Status: 200 OK`, `Time: 958ms`, and `Size: 469 B`.

Resultado: Se agrega el libro.

Probar endpoint privado para eliminar un libro usando token

Primero agregar un libro y luego copiar su ID usando los endpoints para agregar y consultar los libros respectivamente. Después utilizar el endpoint /ID con el método DELETE para eliminarlo.



The screenshot shows a Postman request configuration for a DELETE method. The URL is set to `localhost:3001/5dd704c40f6dd413fe6c367b`. The Authorization tab is selected, showing a Bearer Token type with a placeholder for the token value. A note in the token field area cautions about sensitive data. The response body is displayed in JSON format, showing a message: "Book removed.".

DELETE localhost:3001/5dd704c40f6dd413fe6c367b

Authorization

Bearer Token

Token

eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6IjJrWTJNMFI5UXpC...

Body

1 "message": "Book removed."

Resultado: Se elimina el libro indicado. Al consultar nuevamente los libros ya no aparece.

Probar endpoint privado para actualizar un libro usando token

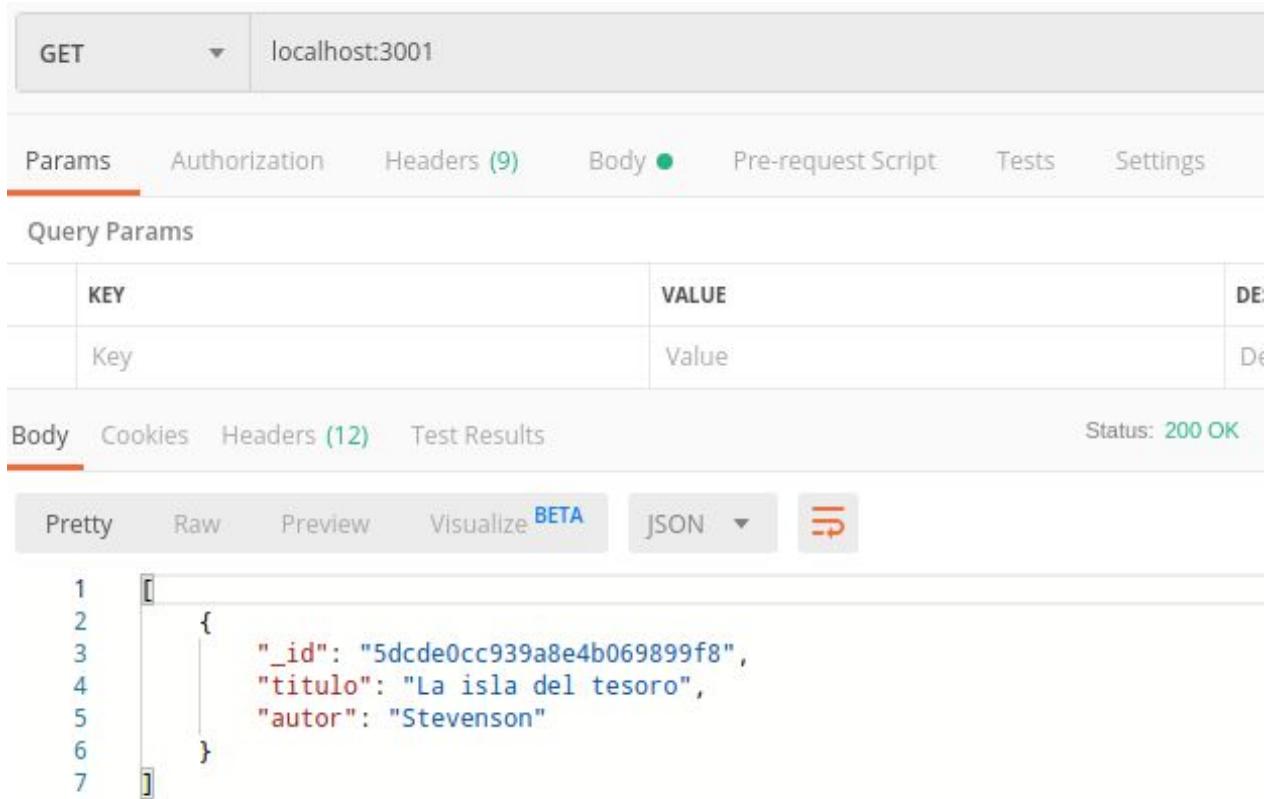
Primero agregar un libro y luego copiar su ID usando los endpoints para agregar y consultar los libros respectivamente. Después utilizar el endpoint /ID con el método PUT y el libro actualizado en el body de la Request para actualizarlo.

The screenshot shows the Postman interface with a PUT request to `localhost:3001/5dd7064e0f6dd413fe6c367c`. The Authorization tab is selected, showing a Bearer Token. The token value is `eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6IjJrWTJNMFI5UXpC...`. The response status is 200 OK, and the JSON body shows the message "Book updated."

Resultado: Se actualizan los atributos del libro indicado. Aparecen los nuevos valores seteados al consultar nuevamente los libros.

Probar el endpoint público para consultar los libros sin usar token

Se utiliza el endpoint / con el método GET.



The screenshot shows the Postman interface with the following details:

- Method: GET
- URL: localhost:3001
- Params tab is selected, showing a table with one row: Key (Value) and Value (Value).
- Body tab is selected, showing the response body in Pretty, Raw, Preview, and Visualize (BETA) formats. The JSON response is:

```
1 [ { 2   "_id": "5dcde0cc939a8e4b069899f8", 3   "titulo": "La isla del tesoro", 4   "autor": "Stevenson" 5 } ] 6 7
```

The status bar at the bottom right indicates Status: 200 OK.

Resultado: Como este endpoint se definió como público se recibe la información sin necesidad de agregar el token a la Request.