INFORME DE GUÍA PRÁCTICA

I. PORTADA

Tema: Seguridad en Sistemas Distribuidos

Unidad de Organización Curricular: PROFESIONAL

Nivel y Paralelo: 6to "A"

Alumnos participantes: Albán Chávez Melanie Elizabeth
Alvarez Coello Freddy Guillermo
Salas Acosta Alison Micaela
Soriano Proaño Rafael Andrés

Asignatura: Aplicaciones Distribuidas

Docente: Ing. José Caiza, Mg.

II. INFORME DE GUÍA PRÁCTICA

1.1 Objetivos

General:

Comprender e implementar mecanismos esenciales de seguridad en un sistema distribuido hospitalario, enfocándose en tres pilares fundamentales: autenticación con JWT, resiliencia del sistema mediante patrones como *Retry* y *Circuit Breaker*, y la protección de la comunicación entre servicios mediante TLS y VPN simulada.

Específicos:

- Diseñar e implementar una solución segura de autenticación basada en JSON Web Tokens (JWT), que garantice la verificación fiable de identidad tanto para usuarios como servicios dentro del ecosistema hospitalario distribuido.
- Incorporar estrategias de resiliencia, como los patrones de reintento (Retry) y disyuntor (Circuit Breaker), para mantener la disponibilidad y robustez en la interacción entre microservicios esenciales del sistema.
- Implementar canales de comunicación seguros entre los distintos componentes del sistema, utilizando cifrado TLS y una red privada virtual simulada, a fin de asegurar la confidencialidad e integridad de los datos transferidos.

1.2 Modalidad

Presencial.

1.3 Tiempo de duración

Presenciales: 3

No presenciales: 0

1.4 Instrucciones

- El trabajo se realizará en grupos, como parte del sistema distribuido hospitalario simulado (MedCity).
- Desarrollar y demostrar la implementación práctica de tres mecanismos de seguridad clave: autenticación JWT, patrones de resiliencia, y cifrado en tránsito (TLS/VPN).
- Participar en actividades guiadas en clase como codificación, análisis de arquitectura y discusión técnica.
- Completar una evaluación teórica integradora.
- Documentar brevemente los hallazgos y entregarlos al finalizar (opcional, si lo requiere el docente).

1.5 Listado de equipos, materiales y recursos

- Computadora personal con Docker, Postman, VS Code instalados
- Acceso a Internet y GitHub
- Plantilla base de microservicio con autenticación
- Material de clase y bibliografía técnica
- Herramientas de evaluación (quiz digital o papel)
- Aula virtual para referencias

TAC (Tecnologías para el Aprendizaje y Conocimiento) empleados en la guía práctica:

☒Plataformas educativas

☒Simuladores y laboratorios virtuales

☒Aplicaciones educativas

☒Recursos audiovisuales

☐Gamificación

☒Inteligencia Artificial

1.6 Actividades por desarrollar

#	Objetivo	S (Situación)	M (Medición)	A (Actividades)	R (Relevancia)	T (Tiempo)
1	Implementar autenticación JWT para un microservicio de MedCity	"Codificar la emisión y validación de JWT en el microservicio de autenticación de MedCity"	Token generado correctamente + acceso autorizado a endpoint protegido	Codificación guiada con plantilla base y prueba práctica con Postman o curl	JWT es clave en la autenticación distribuida en MedCity	45 min
2	Analizar patrones de resiliencia: Circuit Breaker y Retry	"Identificar dónde y cómo aplicar Circuit Breaker y Retry entre microservicios de MedCity"	Análisis en grupos sobre escenarios de fallo + explicación oral o escrita	Análisis grupal con plantilla arquitectónica + discusión de casos	Refuerza la disponibilidad y tolerancia a fallos del sistema	25 min
3	Explicar uso de cifrado en tránsito (TLS) y VPNs	"Explicar el uso de TLS y VPNs para proteger la comunicación entre servicios de MedCity"	Aportaciones grupales a una pregunta guía + validación en quiz final	Debate guiado en clase + ejemplo de arquitectura protegida	Esencial para garantizar la seguridad de la información en tránsito	20 min
4	Evaluación integradora (teoría) – Quiz 10 preguntas	"Responden un quiz de 10 preguntas sobre JWT, OAuth2, resiliencia, cifrado y VPNs"	≥ 80 % de respuestas correctas en el instrumento	Individual, en papel o digital; discusión posterior si hay tiempo	Refuerza comprensión global y permite detectar conceptos débiles	25 min
5	Cierre y repaso	"Revisión de dudas y reflexión sobre los aprendizajes clave"	Participación activa y preguntas relevantes	Explicacion Guia Ape Asociada	Consolida lo aprendido y orienta hacia el uso en futuros proyectos	10 min
-	Transiciones, organización y pausas breves	Logística entre bloques, tiempo para preguntas, cambio de actividad	Actividades inician y terminan sin retrasos	Coordinación del docente + flexibilidad para ajustes menores	Asegura fluidez de la sesión	25 min (distribuidos entre bloques)

Introduccion

En sistemas distribuidos modernos, mantener la identidad del usuario entre servicios sin usar sesiones en el servidor es fundamental. Aquí entra en juego JWT, un estándar abierto (RFC 7519) que permite transmitir información segura y firmada entre partes como un objeto JSON compacto.

1. **¿Qué es JWT?**

JWT (JSON Web Token) es un estándar abierto definido en la RFC 7519 que permite el intercambio seguro de información entre dos partes. Es especialmente útil en sistemas distribuidos donde no se desea mantener el estado de sesión en el servidor (stateless).

Un token JWT consta de **tres partes** codificadas en Base64 y separadas por puntos:

- **Header (Encabezado):**
  Contiene metadatos del token, como el tipo (typ: "JWT") y el algoritmo de firma utilizado, por ejemplo HS256 o RS256.
- **Payload (Cuerpo):**
  Contiene los **claims** o declaraciones, que son datos relevantes del usuario o del sistema, como:

- **Signature (Firma):**

  Es una firma digital generada a partir del header, el payload y una clave secreta (o una clave privada en el caso de algoritmos asimétricos). Esta garantiza que el contenido no ha sido alterado y que fue emitido por una fuente confiable.

Ventajas principales:

- Stateless (no requiere sesiones del lado servidor)
- Rápido para verificar identidad
- Fácil de transmitir por headers HTTP o almacenamiento local

2. **¿Diferencia entre Autenticación y Autorización?**

Autenticación:

Es el proceso mediante el cual un sistema verifica la identidad de un usuario o servicio. Generalmente, esto implica el ingreso de credenciales (usuario y contraseña). Si las credenciales son válidas, el sistema emite un JWT, que el cliente usará en las siguientes solicitudes.

Ejemplo: El sistema confirma que "juan.perez@ejemplo.com" es un usuario válido.

Autorización:

Se refiere a determinar qué acciones o recursos puede acceder un usuario autenticado. Esto depende de su rol, permisos u otras políticas de control de acceso.

Ejemplo: Aunque "juan.perez@ejemplo.com" está autenticado, solo los usuarios con rol "admin" pueden acceder al módulo de configuración del hospital.

Relación entre ambos:

- La autenticación siempre debe ocurrir antes de la autorización.
- JWT puede contener información de ambos procesos, pero el backend debe verificar y validar cada uno por separado.

3. **Implementación práctica en Spring Boot**

a) Model para crear el usuario

Usuario.js

```
src > models > JS Usuario.js > ...
1   const pool = require('../config/database');
2
3   class Usuario {
4     // Método para crear un nuevo usuario
5     static async create({ username, password, rol = 'doctor', medico_id = null }) {
6       const connection = await pool.getConnection();
7       try {
8         // Inserta el usuario en la base de datos
9         const [result] = await connection.execute(
10          'INSERT INTO usuarios (username, password, rol, medico_id) VALUES (?, ?, ?, ?)',
11          [username, password, rol, medico_id]
12        );
13        return result.insertId;
14      } finally {
15        connection.release();
16      }
17    }
18
19    // Método para buscar un usuario por su nombre de usuario
20    static async findOne({ username }) {
21      const connection = await pool.getConnection();
22      try {
23        const [rows] = await connection.execute(
24          'SELECT * FROM usuarios WHERE username = ?',
25          [username]
26        );
27        return rows[0];
28      } finally {
29        connection.release();
30      }
31    }
32  }
33  module.exports = Usuario;
34
```

b) Endpoint de Login

Se creó un endpoint /api/auth/login que recibe credenciales en formato JSON (email y contraseña). Si son válidas, se genera un JWT:

```
// Controlador para el login
exports.login = async (req, res) => {
  try {
    const { username, password } = req.body;
    // Busca el usuario en la base de datos
    const usuario = await Usuario.findOne({ username });

    // Verifica si el usuario existe y la contraseña es correcta
    if (!usuario || !(await bcrypt.compare(password, usuario.password))) {
      return res.status(401).json({ error: 'Credenciales inválidas' });
    }

    // Genera el token JWT
    const token = generarToken(usuario);
    res.json({ token });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};
```

c) **Generacion del Token**

Se usó la librería jjwt para construir el JWT con fecha de expiración, subject y firma:

d) **Filtro de Autentificacion**

Se implementó un filtro personalizado que lee el token desde el header Authorization, lo valida y autoriza o bloquea el acceso:

2. Patrones de Resiliencia: Circuit Breaker y Retry

En sistemas distribuidos, los patrones de resiliencia como Circuit Breaker y Retry son esenciales para mantener la estabilidad, disponibilidad y robustez de los servicios ante fallos.

2.1 Circuit Breaker

a) ¿Qué es?

El Circuit Breaker se basa en el principio de "fail-fast" para evitar el desperdicio de recursos en operaciones condenadas al fracaso. Su nombre proviene de los disyuntores eléctricos que protegen circuitos de sobrecargas.

b) ¿Cómo funciona?

1. **Closed:**
   - Todas las llamadas se ejecutan normalmente
   - Mantiene un contador de fallos recientes (sliding window)
   - Métricas que monitorea: tasa de error, latencia, timeouts
   - Umbral típico: 50% de fallos en los últimos N requests o X fallos en Y segundos

2. **Open:**
   - Rechaza inmediatamente todas las llamadas
   - Devuelve una excepción predefinida (CircuitBreakerOpenException)
   - Inicia un temporizador para el período de recuperación
   - Período típico: 30-60 segundos para servicios HTTP, 5-10 minutos para bases de datos

3. **Half-Open:**
   - Permite un número limitado de llamadas de prueba (típicamente 1-5).
   - Si todas las pruebas son exitosas (Cerrado)
   - Si alguna prueba falla (Abierto)
   - Implementa un mecanismo de "single flight" para evitar múltiples pruebas simultáneas

c) Ventajas:

- Previene fallos en cascada.

- Protege los recursos.
- Mejora la resiliencia.
- Facilita la recuperación

d) Desventajas:

- Requiere buena configuración.
- Puede generar falsos positivos.
- Aumenta la complejidad.
- El usuario puede recibir más errores visibles.

2.2 Retry

a) ¿Qué es?

El patrón **Retry** es un patrón de resiliencia que permite que un sistema maneje fallas transitorias al volver a intentar automáticamente una operación fallida. Este patrón es particularmente útil cuando los errores no son permanentes, sino causados por condiciones temporales

b) ¿Cómo funciona?

- Reintenta después de un intervalo fijo o variable.
- Puede usar estrategias como backoff exponencial y jitter.

c) Ventajas:

- Tolerancia a fallos transitorios.
- Experiencia del usuario mejorada.
- Compatible con muchas tecnologías.
- Fácil de implementar.

d) Desventajas

- Puede sobrecargar el sistema.
- Aumenta la latencia.
- Necesita control de intentos y tiempos.
- No es útil ante errores persistentes.

2.3 Ejemplo Aplicado (MedCity)

a) Escenario:

Un paciente agenda una cita médica usando la aplicación. Esto involucra microservicios como autenticación, disponibilidad, citas y notificaciones.

Uso de Retry:

Si el servicio de disponibilidad falla momentáneamente, el servicio de citas intenta 2 o 3 veces antes de declarar el error.

Uso de Circuit Breaker:

Si el servicio sigue fallando, el circuito se abre para evitar sobrecarga. Mientras está abierto, se informa al usuario que el sistema no está disponible momentáneamente.

Característica	**Circuit Breaker**	**Retry**
¿Cuándo se usa?	Cuando un servicio falla frecuentemente	Cuando un servicio falla de forma ocasional
Objetivo principal	Prevenir saturación y fallos en cascada	Recuperarse de fallos transitorios
Reacción ante error	Bloquea llamadas después de varios fallos	Reintenta automáticamente

Cifrado en tránsito: TLS y VPNs

1. **¿Qué es el cifrado en tránsito?**

El cifrado en tránsito protege los datos mientras se están transmitiendo entre servicios o componentes de un sistema:

- Entre el frontend y el backend (cliente-servidor)
- Entre microservicios
- Entre el backend y la base de datos

Este cifrado evita que un atacante pueda interceptar y leer los datos en tránsito, como ocurre en ataques de tipo "man-in-the-middle".

2. TLS (Transport Layer Security)

TLS es un protocolo criptográfico que proporciona comunicación segura a través de una red, típicamente Internet. Es la evolución de SSL (Secure Sockets Layer) y opera entre la capa de transporte y la capa de aplicación del modelo OSI. TLS es el protocolo estándar para el cifrado en tránsito.

2.1.Características principales de TLS:

2.1.1. Cifrado de datos:

- Cifrado simétrico: Utiliza algoritmos como AES-256 para cifrar los datos durante la transmisión
- Cifrado asimétrico: Emplea RSA o ECDSA para el intercambio seguro de claves
- Perfect Forward Secrecy: Genera claves de sesión únicas que no comprometen comunicaciones pasadas

2.1.2. Autenticación:

- **Certificados digitales:** Verifican la identidad del servidor (y opcionalmente del cliente)
- **Cadena de confianza:** Utiliza Autoridades Certificadoras (CA) para validar certificados
- **Mutual TLS (mTLS):** Autenticación bidireccional entre cliente y servidor

2.1.3. Integridad de datos:

- **Hashing criptográfico:** Utiliza SHA-256 o superior para detectar alteraciones
- **Message Authentication Code (MAC):** Garantiza que los datos no han sido modificados

2.2. ¿Cómo protege TLS?

- Cifra los datos antes de enviarlos
- Autentica que el servidor (y opcionalmente el cliente) son quienes dicen ser
- Garantiza la integridad de los datos

2.3. ¿Dónde se usa TLS?

- HTTPS: Es HTTP sobre TLS (común en sitios web seguros)
- En conexiones entre microservicios
- En conexiones a bases de datos (como PostgreSQL con sslmode=require)

2.4. Beneficios en un sistema distribuido:

- Protege contra escuchas (man-in-the-middle)
- Asegura que los servicios se comuniquen con los endpoints correctos
- Cumple con requisitos de compliance como GDPR o HIPAA

3. VPNs (Virtual Private Networks)

Las VPNs crean un túnel cifrado entre dos puntos de red, permitiendo comunicación segura sobre infraestructura no confiable. En arquitecturas distribuidas, conectan servicios ubicados en diferentes redes o centros de datos.

Una VPN crea un túnel cifrado a través de una red pública (como Internet) que:

- Aísla la comunicación entre servicios
- Oculta las direcciones IP internas
- Protege contra ataques de red

3.1.Tipos relevantes para arquitecturas distribuidas:

1. **Site-to-Site VPN:** Conecta redes completas, como un data center con la nube
2. **Client-to-Site VPN:** Para acceso remoto seguro
3. **Mesh VPN:** Para comunicaciones entre múltiples nodos distribuidos

3.2.¿Cómo protege una VPN?

- Todo el tráfico dentro del túnel está cifrado
- Permite que solo dispositivos autorizados dentro de la VPN se comuniquen entre sí

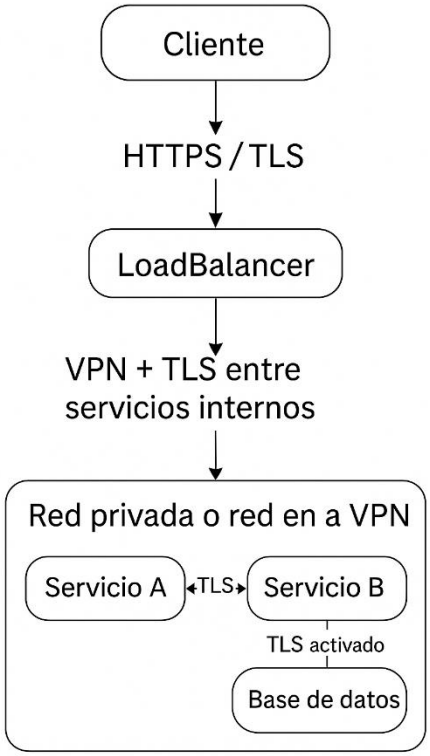
3.3.¿Cuándo usar una VPN?

- Para proteger la comunicación entre servidores distribuidos en diferentes regiones
- Para que los desarrolladores accedan de forma segura a la infraestructura
- Para conexiones entre servicios en diferentes redes privadas o proveedores de nube

4. ¿Cuándo usar TLS y cuándo VPN?

Escenario	**TLS**	**VPN**
Comunicación cliente-servidor	Obligatorio	Opcional
Comunicación entre microservicios	Recomendado	Recomendado
Acceso seguro de desarrolladores	No	Necesario
Conexión a base de datos remota	Sí (SSL/TLS)	Opcional

5. Ejemplo de arquitectura protegida (TLS + VPN)



6. Creación de túnel

Pasos para exponer el backend en Express a internet usando Ngrok:

- **Requisitos previos**

  Tener Node.js y Express instalados

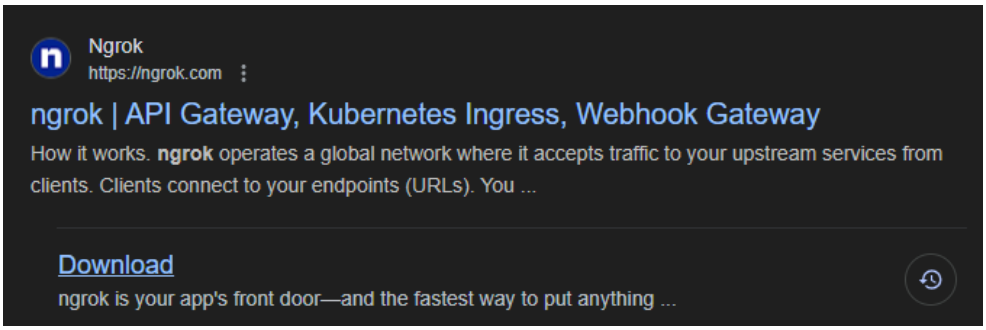
  Tener una aplicación backend en Express funcionando localmente

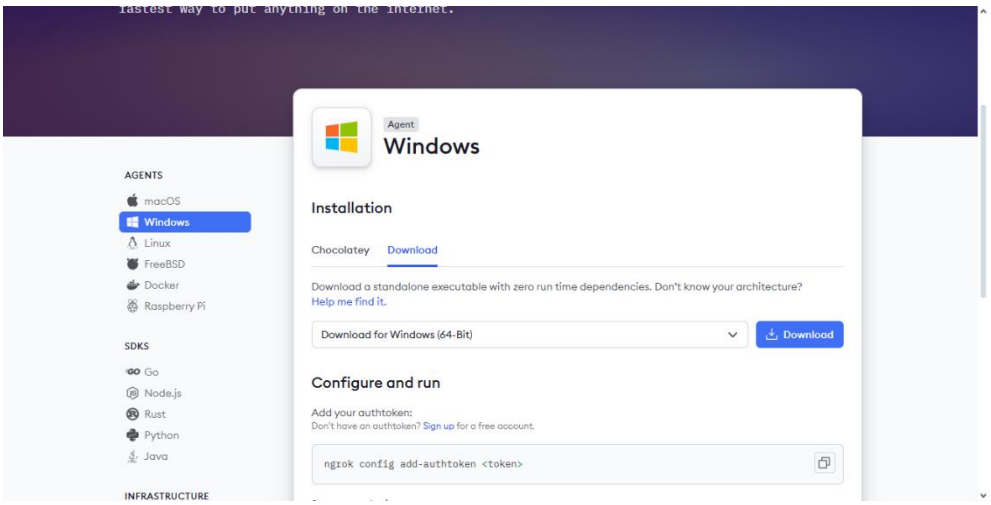
  Tener una cuenta en Ngrok (la versión gratuita es suficiente para pruebas)

- **Pasos**

  1. Instalar Ngrok

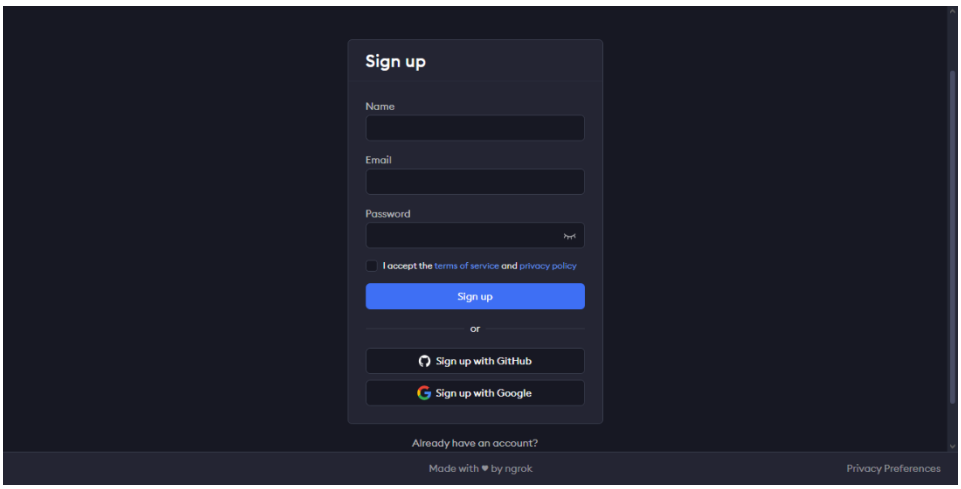
  En el buscador ingresar directamente a la opción de descarga en la primera recomendación



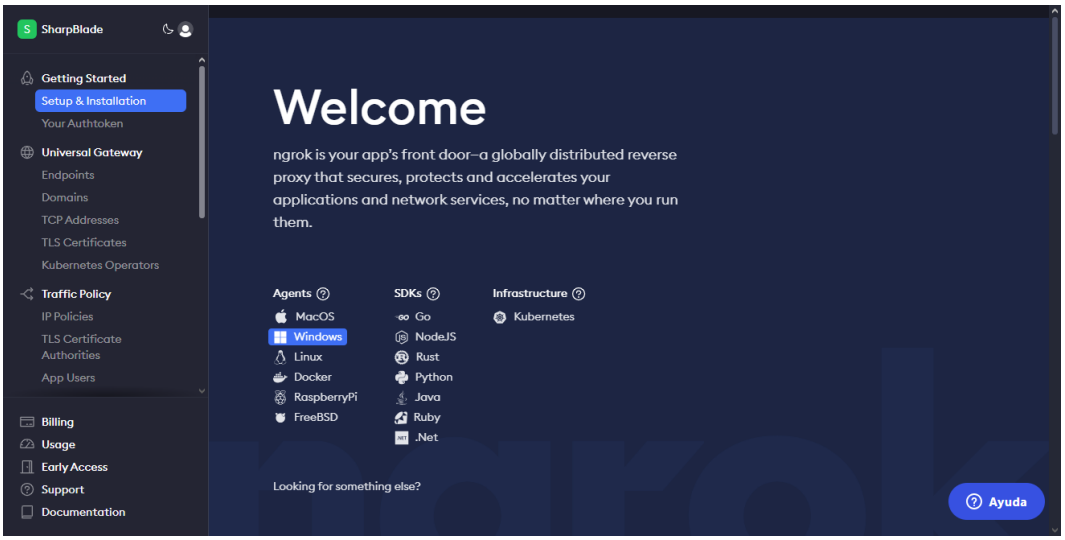


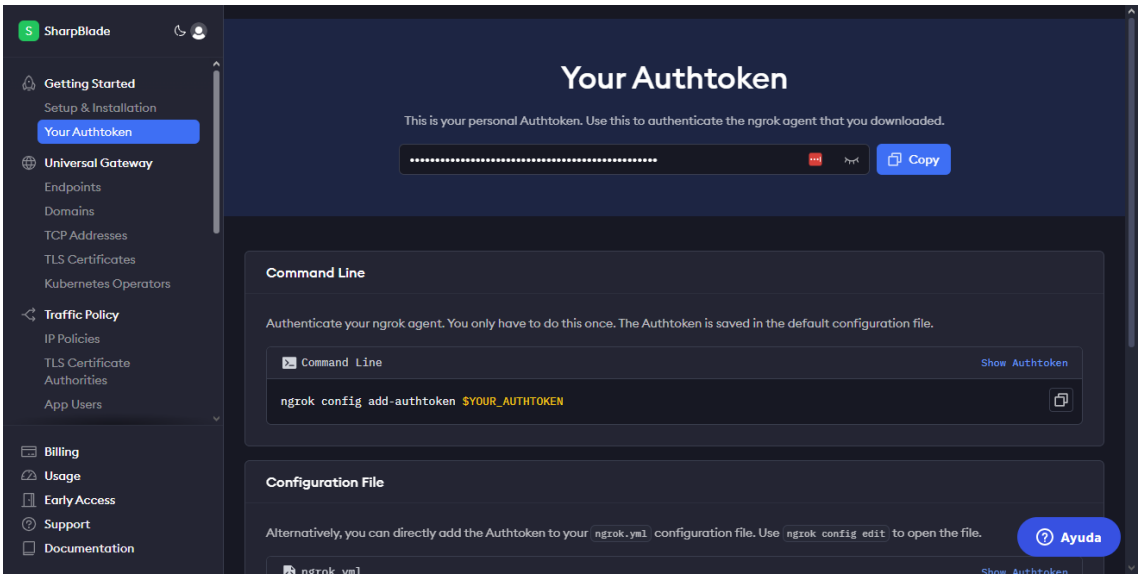
2. **Autenticar mediante una cuenta de Ngrok**

Ingresar a la dirección y crear una cuenta o iniciar con algún proveedor



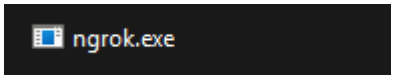
Una vez iniciado sesión nos lleva al dashboard y seleccionamos la opción “Your Authtoken”

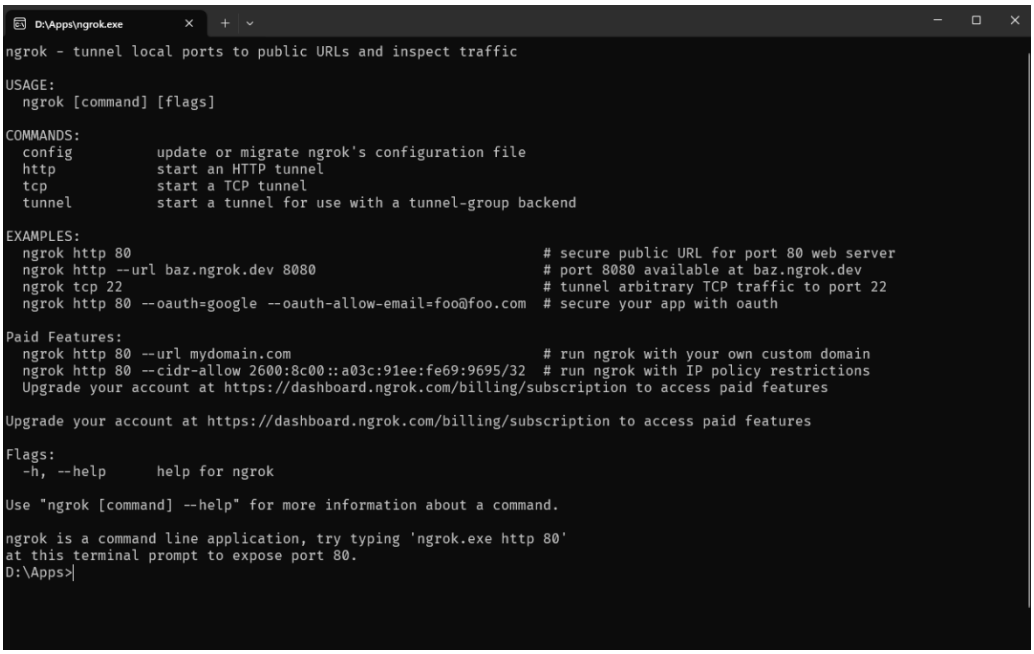




En la misma nos presenta el comando que debemos ingresar en la terminal:

Del archivo .zip que descargamos iniciamos el ejecutable





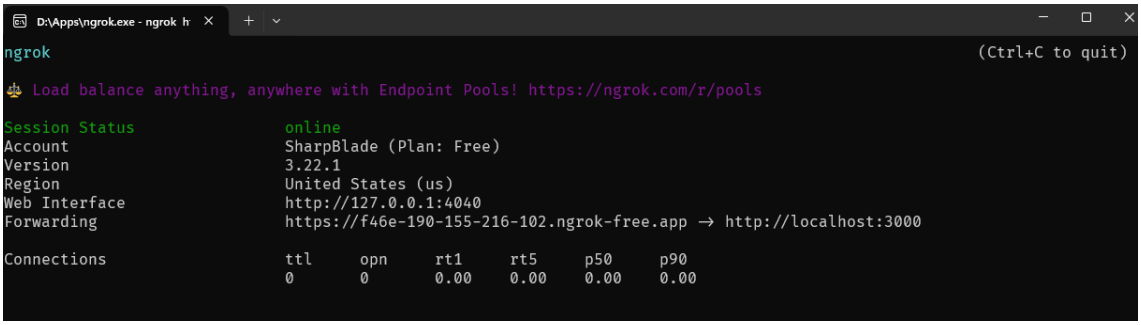
En esta terminal ingresamos el comando y el token para autenticar

```
ngrok config add-authtoken $YOUR_AUTHTOKEN
```

3. Crear el túnel con Ngrok

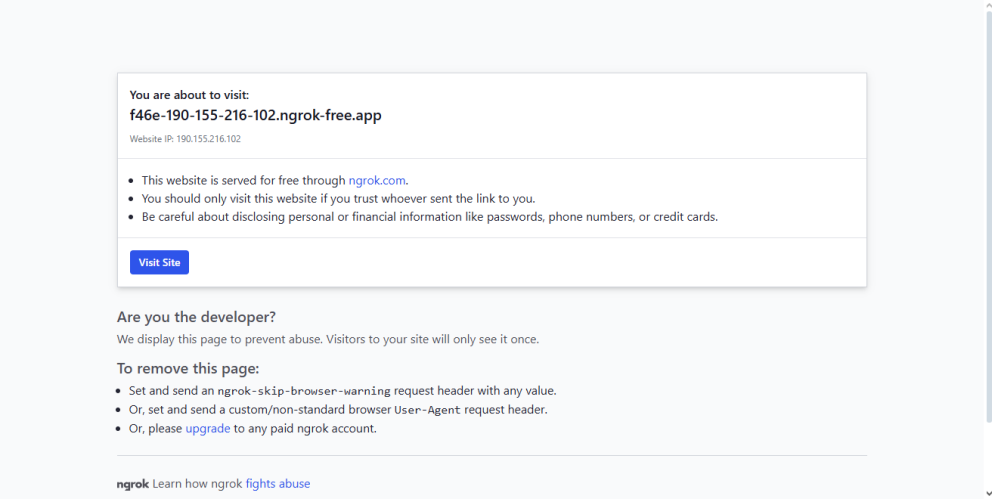
Ejecutar el siguiente comando, si se desea reemplazar el 3000 por otro puerto a su elección:

```
ngrok http 3000
```

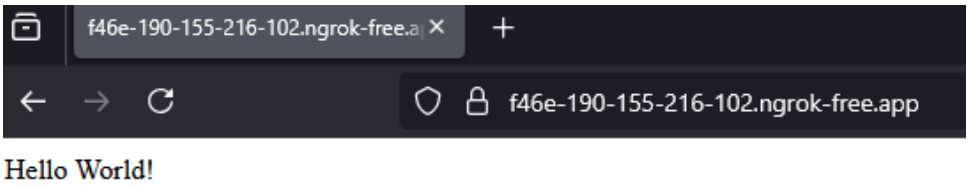


Ngrok nos proporciona una dirección publica con la cual podremos acceder al servicio que levantaremos.

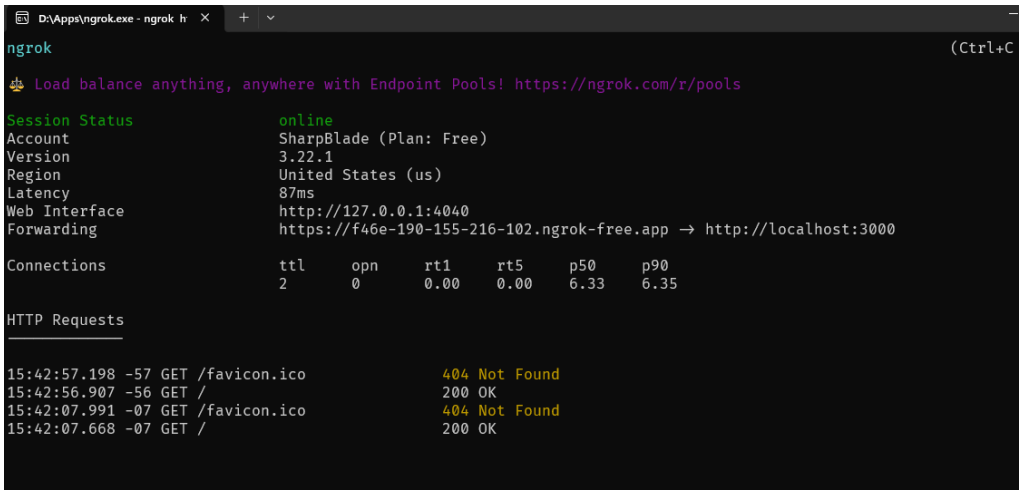
Ahora iniciamos el backend de la aplicación e ingresaremos a la dirección publica



Aceptamos dirigirnos hacía el enlace



Mediante esa url el servicio es accesible desde cualquier lugar de internet y enviara el tráfico hacía el servidor local



1.7 Resultados obtenidos

Resultados de JWT

- El sistema devuelve un token válido tras autenticarse.
- Peticiones con el token acceden a los recursos protegidos.
- Peticiones sin token o con token inválido reciben error 403 Forbidden.

El estudiante es capaz de aplicar técnicas prácticas de seguridad como JWT, patrones de resiliencia y cifrado en tránsito en el contexto de un sistema distribuido real. Muestra entendimiento funcional de cómo proteger servicios en entornos críticos como el hospitalario.

1.8 Habilidades blandas empleadas en la práctica

☐ Liderazgo

☐ Trabajo en equipo

☒ Comunicación asertiva

☒ La empatía

☐ Pensamiento crítico

☐ Flexibilidad

☒ La resolución de conflictos

☒ Adaptabilidad

☒ Responsabilidad

1.9 Conclusiones

El empleo de JWT posibilita una autenticación sin mantener estado en el servidor, lo que resulta ideal para entornos distribuidos que requieren alta escalabilidad y eficiencia. Al incorporar información relevante directamente en el payload, como el rol o el identificador del usuario, es posible realizar procesos de autorización básicos sin recurrir constantemente a la base de datos, reduciendo la carga y mejorando el rendimiento. Esta estrategia promueve buenas prácticas en seguridad, optimiza la comunicación entre microservicios y facilita la implementación de APIs modernas y seguras.

1.10 Recomendaciones

- Estudiar previamente los fundamentos de JWT, TLS y patrones de resiliencia (como Retry y Circuit Breaker), con el fin de comprender su aplicación en arquitecturas distribuidas.
- Consultar la documentación oficial y fuentes confiables para una correcta implementación y configuración de cada tecnología involucrada en el sistema.
- Realizar pruebas exhaustivas en un entorno de desarrollo local, asegurando el correcto funcionamiento de los mecanismos de seguridad antes del despliegue en producción.
- Buscar orientación con el docente o tutor académico ante cualquier dificultad relacionada con la integración o funcionamiento de las capas de seguridad del sistema.

1.11 Referencias bibliográficas

[1] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," *RFC 7519*, Internet Engineering Task Force (IETF), May 2015. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7519

[2] C. Richardson, *Microservice Patterns: With examples in Java*, Shelter Island, NY: Manning Publications, 2018.

[3] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 8th ed., Boston, MA: Pearson, 2020.