Sistema de Autenticación y Gestión de Usuarios

SEGURIDAD EN EL DESARROLLO DE APLICACIONES

Grupo: IDGS10

Víctor Manuel Rangel Mejía

Docente: Freddy Uriel Moreno Guzmán

Universidad Tecnológica de Querétaro

31 de marzo de 2025

Índice general

1. Introducción al Sistema

1.1. Descripción General

El Sistema de Autenticación y Gestión de Usuarios es una aplicación desarrollada en Node.js que proporciona un conjunto completo de funcionalidades para la gestión de identidades, autenticación segura y control de acceso basado en roles. Implementa un sistema robusto que protege contra amenazas comunes de seguridad mientras ofrece una experiencia de usuario fluida.

1.2. Objetivos del Sistema

- Proporcionar un sistema de autenticación seguro y confiable
- Implementar autenticación multi-factor con varios proveedores OAuth
- Establecer controles de acceso basados en roles (RBAC)
- Proteger contra ataques de fuerza bruta y otras amenazas
- Facilitar la gestión de usuarios para administradores
- Implementar flujos seguros de restablecimiento de contraseñas

1.3. Tecnologías Utilizadas

Tecnología	Descripción
Node.js	Entorno de ejecución para JavaScript del lado del servidor
Express	Framework web para Node.js
MongoDB	Base de datos NoSQL utilizada para almacenar la información de usuarios
Passport.js	Middleware de autenticación para Node.js con soporte para múltiples estrategias
OAuth 2.0	Protocolo estándar para la autorización de acceso
Nodemailer	Módulo para el envío de correos electrónicos
Express-session	Middleware para la gestión de sesiones en Express
Crypto	Librería para funciones criptográficas

Cuadro 1.1: Tecnologías principales utilizadas en el sistema

1.4. Arquitectura del Sistema

El sistema sigue una arquitectura de aplicación web de tres capas:

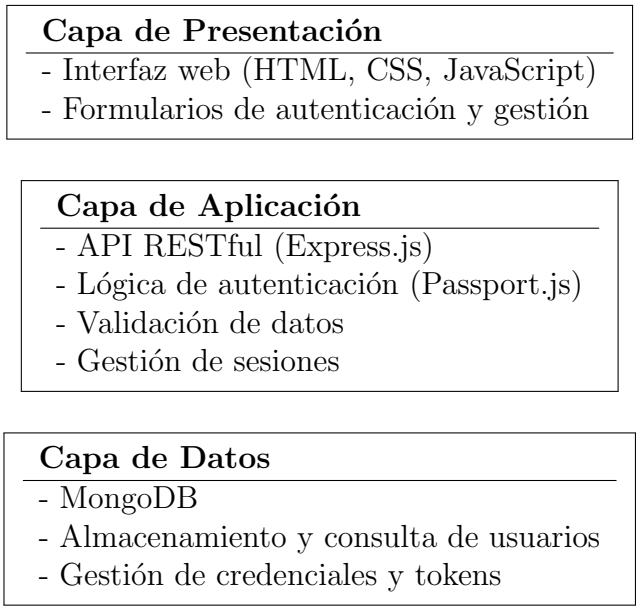

Figura 1.1: Arquitectura de tres capas del sistema

2. Componentes del Sistema

2.1. Módulo de Autenticación

El módulo de autenticación es el componente central del sistema, encargado de verificar la identidad de los usuarios y gestionar sus sesiones. Implementa múltiples estrategias de autenticación:

2.1.1. Autenticación Básica

La autenticación básica utiliza el modelo tradicional de nombre de usuario y contraseña. El sistema implementa las siguientes medidas de seguridad:

- Almacenamiento de contraseñas utilizando hashing SHA-256
- Protección contra ataques de fuerza bruta mediante limitación de intentos fallidos
- Bloqueo de cuentas temporalmente después de 3 intentos fallidos (5 minutos)
- Validación de datos en el lado del servidor

```
app.post('/login', async (req, res) => {
  const { username, password } = req.body;

  if (!username || !password) {
    return res.status(400).send('Usuario y contrase a son requeridos');
  }

  try {
    const user = await db.collection('users').findOne({ username });

    if (!user) {
```

```
12        return res.status(401).send('Usuario no encontrado');
13      }
14
15      // Verificar si el usuario puede intentar iniciar sesi n
16      const canAttempt = await checkLoginAttempts(username);
17      if (!canAttempt) {
18        return res.status(403).send('Demasiados intentos
    fallidos. Espere 5 minutos o reinicie su contrase a.');
19      }
20
21      const hashedPassword = crypto.createHash('sha256').update
    (password).digest('hex');
22
23      if (hashedPassword === user.password) {
24        // Restablecer intentos fallidos
25        await db.collection('users').updateOne(
26          { username },
27          { $set: { failedAttempts: 0, lastFailedAttempt: null
    } }
28        );
29
30        req.session.username = username;
31        req.session.role = user.role;
32        req.session.authenticated = true;
33
34        // Redirigir seg n el rol
35        if (user.role === 'admin') {
36          return res.send({
37            success: true,
38            message: `Bienvenido, ${username} (${user.role})`,
39            redirectTo: '/admin'
40          });
41        } else {
42          return res.send({
43            success: true,
44            message: `Bienvenido, ${username} (${user.role})`,
45            redirectTo: '/user'
46          });
47        }
48      } else {
49        // Incrementar intentos fallidos
50        await db.collection('users').updateOne(
51          { username },
52          {
53            $inc: { failedAttempts: 1 },
54            $set: { lastFailedAttempt: Date.now() }
55          }
56        );
57
```

```
58        return res.status(401).send('Credenciales incorrectas')
      ;
59      }
60    } catch (err) {
61      console.error('Error en el inicio de sesi n:', err);
62      res.status(500).send('Error en el servidor');
63    }
64  });
```

Listing 2.1: Proceso de autenticación básica

2.1.2. Autenticación OAuth

El sistema también soporta autenticación mediante proveedores OAuth 2.0, incluyendo:

- Google
- GitHub
- Microsoft

La implementación utiliza Passport.js para gestionar los flujos de autenticación OAuth. El sistema almacena identificadores únicos para cada proveedor OAuth en el perfil del usuario, permitiendo vincular cuentas existentes.

```
1  // Funci n para procesar el resultado de autenticaci n
      OAuth
2  async function handleOAuthLogin(profile, provider, done) {
3    try {
4      // Extraer email del perfil
5      const email = profile.emails && profile.emails.length > 0
6        ? profile.emails[0].value
7        : null;
8
9      // Construir consulta para buscar usuario existente
10     const query = { $or: [{ [`${provider}Id`]: profile.id }]
       };
11     if (email) {
12       query.$or.push({ email });
13     }
14
15     // Buscar si el usuario ya existe
16     let user = await db.collection('users').findOne(query);
17
18     if (user) {
19       // Si el usuario existe pero no tiene ID del proveedor,
       actualizar
```

```
20        if (!user[`${provider}Id`]) {
21          await db.collection('users').updateOne(
22            { _id: user._id },
23            { $set: { [`${provider}Id`]: profile.id } }
24          );
25        }
26        return done(null, user);
27      }
28
29      // Si el usuario no existe, crearlo
30      const newUser = {
31        username: profile.displayName || profile.username || `${provider}-user-${profile.id}`,
32        email: email,
33        [`${provider}Id`]: profile.id,
34        role: 'operador', // Rol por defecto para usuarios OAuth
35        createdAt: new Date(),
36        failedAttempts: 0,
37        lastFailedAttempt: null
38      };
39
40      const result = await db.collection('users').insertOne(newUser);
41      newUser._id = result.insertedId;
42
43      return done(null, newUser);
44    } catch (err) {
45      return done(err, null);
46    }
47  }
```

Listing 2.2: Configuración de Passport para OAuth

2.2. Módulo de Gestión de Usuarios

Este módulo proporciona funcionalidades para administrar usuarios, incluyendo:

- Registro de nuevos usuarios (solo para administradores)
- Listado de todos los usuarios del sistema
- Visualización de información de perfil

El sistema implementa un control de acceso basado en roles (RBAC) con los siguientes roles definidos:

Rol	Permisos
admin	Acceso completo al sistema, incluyendo registro de usuarios, visualización de todos los usuarios, acceso a panel de administración.
supervisor	Acceso a la interfaz de usuario y funcionalidades básicas de operación. Permisos adicionales por definir.
operador	Acceso básico a la interfaz de usuario y funcionalidades limitadas.

Cuadro 2.1: Roles del sistema y sus permisos

2.3. Módulo de Restablecimiento de Contraseñas

Este módulo maneja el flujo completo de restablecimiento de contraseñas olvidadas, implementando las siguientes medidas de seguridad:

- Generación de códigos de verificación aleatorios
- Límite de tiempo para los códigos (3 minutos)
- Envío seguro de códigos por correo electrónico
- Verificación de correo electrónico antes de permitir el cambio de contraseña

```
1  // Ruta para solicitud de restablecimiento de contrase a
2  app.post('/request-password-reset', async (req, res) => {
3    const { email } = req.body;
4
5    if (!email) {
6      return res.status(400).json({ message: 'Correo
       electr nico es requerido' });
7    }
8
9    try {
10     // Verificar si el usuario existe
11     const user = await db.collection('users').findOne({ email
       });
12
13     // Generar c digo de verificaci n (6 caracteres
       alfanum ricos)
14     const verificationCode = Math.random().toString(36).
       substring(2, 8).toUpperCase();
15
```

```
16      // Almacenar el c digo con tiempo de expiraci n (3
        minutos)
17      await db.collection('users').updateOne(
18        { email },
19        {
20          $set: {
21            passwordResetCode: verificationCode,
22            passwordResetExpires: Date.now() + 3 * 60 * 1000
23          }
24        }
25      );
26
27      // Enviar correo con el c digo de verificaci n
28      // ...
29
30      res.status(200).json({
31        message: "Se ha enviado un c digo de verificaci n a
        tu correo electr nico."
32      });
33    } catch (error) {
34      console.error("Error en la solicitud de restablecimiento
        de contrase a:", error);
35      res.status(500).json({
36        message: "Error al procesar la solicitud. Por favor
        intenta m s tarde."
37      });
38    }
39  });
```

Listing 2.3: Generación y verificación de códigos de restablecimiento

2.4. Sistema de Logging

El sistema implementa un registro detallado de actividades para fines de auditoría y seguridad. Los registros incluyen:

- Fecha y hora de las solicitudes
- Método HTTP utilizado
- URL accedida
- Código de estado de la respuesta
- Tiempo de respuesta
- Usuario que realizó la acción

- Hash de la contraseña proporcionada (para facilitar la auditoría)

```
// Middleware de registro personalizado
const customLogger = (req, res, next) => {
  const startTime = Date.now();
  const originalEnd = res.end;

  res.end = function (chunk, encoding) {
    originalEnd.call(this, chunk, encoding);
    const responseTime = Date.now() - startTime;
    const now = new Date();
    const days = ['Dom', 'Lun', 'Mar', 'Mie', 'Jue', 'Vie', '
   Sab'];
    const day = days[now.getDay()];
    const date = `${now.getDate()}/${(now.getMonth() + 1).
   toString().padStart(2, '0')}/${now.getFullYear()}`;
    const method = req.method;
    const url = req.originalUrl || req.url;
    const status = res.statusCode;
    const username = req.session?.username || req.body?.
   username || 'an nimo';

    // Registro de la actividad
    const logEntry = `${day} ${date} ${method} ${status} ${
   responseTime}ms ${username} ${url}`;
    accessLogStream.write(logEntry + '\n');
  };

  next();
};
```

Listing 2.4: Middleware de registro personalizado

3. Análisis de Seguridad

3.1. Evaluación SonarQube

Se realizó un análisis de seguridad utilizando SonarQube, una plataforma de inspección continua de la calidad del código que detecta problemas en el código fuente y evalúa la calidad general del mismo.

Quality Gate:	**PASSED**
Last analysis:	4 minutes ago

Figura 3.1: Estado del Quality Gate en SonarQube

3.1.1. Métricas de Seguridad

Métrica	Valor	Rating
Security Issues	0 open issues	ratingAA
Reliability Issues	5 open issues	ratingCC
Maintainability Issues	13 open issues	ratingAA
Security Hotspots	21	ratingEE

Cuadro 3.1: Métricas de seguridad de SonarQube

3.1.2. Métricas Adicionales

Métrica	Valor
Accepted Issues	0
Coverage	0.0 % (on 457 lines)
Duplications	0.8 % (on 5.4k lines)

Cuadro 3.2: Métricas adicionales de SonarQube

3.2. Análisis de Puntos Críticos de Seguridad

El análisis de SonarQube identificó 21 puntos críticos de seguridad (Security Hotspots) que requieren revisión. Estos puntos críticos representan áreas del código que podrían ser vulnerables a ataques si no se implementan adecuadamente.

3.2.1. Principales Categorías de Puntos Críticos

1. **Validación de Entrada**: Puntos donde se aceptan datos del usuario que podrían no ser validados adecuadamente.
2. **Gestión de Secretos**: Áreas donde se manejan secretos (contraseñas, claves API) que podrían exponerse.
3. **Configuración de Seguridad**: Configuraciones que podrían no ser óptimas para entornos de producción.
4. **Manejo de Sesiones**: Puntos relacionados con la gestión de sesiones de usuario.

3.3. Vulnerabilidades Identificadas

A continuación se detallan las principales vulnerabilidades identificadas y las medidas implementadas para mitigarlas:

Vulnerabilidad	Descripción	Mitigación
Ataques de fuerza bruta	Intentos repetidos de adivinar contraseñas	Limitación de intentos fallidos, bloqueo temporal después de 3 intentos
Exposición de datos sensibles	Información confidencial visible en logs o respuestas	Ocultación de datos sensibles en registros, uso de HTTPS
Ataques de inyección	Manipulación de consultas a la base de datos	Validación de entradas, uso de consultas parametrizadas
Secuestro de sesión	Toma de control de sesiones activas	Cookies HttpOnly, regeneración de ID de sesión, expiración de sesiones
Cross-Site Scripting (XSS)	Inyección de scripts maliciosos en páginas web	Escape de datos de entrada, encabezados de seguridad

Cuadro 3.3: Vulnerabilidades identificadas y mitigaciones implementadas

3.4. Mejoras Recomendadas

En base al análisis de SonarQube y las mejores prácticas de seguridad, se recomiendan las siguientes mejoras:

1. **Implementar Pruebas Automatizadas**: Aumentar la cobertura de pruebas (actualmente 0.0 %) mediante pruebas unitarias e integración.
2. **Reforzar la Gestión de Secretos**: Utilizar un sistema de gestión de secretos más robusto, como AWS Secrets Manager o HashiCorp Vault.
3. **Implementar Autenticación de Dos Factores (2FA)**: Añadir una capa adicional de seguridad mediante 2FA para todos los usuarios.
4. **Actualizar el Algoritmo de Hash**: Migrar de SHA-256 a algoritmos más seguros como bcrypt o Argon2.
5. **Establecer una Política de Contraseñas Robusta**: Implementar requisitos de complejidad y caducidad para contraseñas.

4. API del Sistema

4.1. Endpoints de Autenticación

Endpoint	**Método**	**Descripción**
/login	POST	Autenticación con nombre de usuario y contraseña
/logout	GET	Cierre de sesión de usuario
/auth/google	GET	Inicio del flujo de autenticación con Google
/auth/github	GET	Inicio del flujo de autenticación con GitHub
/auth/microsoft	GET	Inicio del flujo de autenticación con Microsoft
/auth/google/callback	GET	Callback para completar autenticación de Google
/auth/github/callback	GET	Callback para completar autenticación de GitHub
/auth/microsoft/callback	GET	Callback para completar autenticación de Microsoft
/auth/status	GET	Verifica el estado actual de autenticación

Cuadro 4.1: Endpoints de autenticación

4.2. Endpoints de Gestión de Usuarios

Endpoint	Método	Descripción	Roles
/register	POST	Registra un nuevo usuario	admin
/users	GET	Obtiene lista de todos los usuarios	admin
/api/user-info	GET	Obtiene información del usuario actual	todos

Cuadro 4.2: Endpoints de gestión de usuarios

4.3. Endpoints de Restablecimiento de Contraseña

Endpoint	Método	Descripción
/reset-password	GET	Accede a la página de restablecimiento de contraseña
/reset-password/:verificationCode	GET	Accede directamente al formulario con un código específico
/request-password-reset	POST	Solicita un código de verificación para el email
/check-verification-code	POST	Verifica si un código de verificación es válido
/verify-reset-password	POST	Cambia la contraseña después de verificar el código

Cuadro 4.3: Endpoints de restablecimiento de contraseña

4.4. Endpoints de Páginas

Endpoint	Método	Descripción	Roles
/	GET	Página principal (login)	público
/admin	GET	Panel de administración	admin
/user	GET	Página de usuario	autenticado

Cuadro 4.4: Endpoints de páginas del sistema

5. Instalación y Configuración

5.1. Requisitos Previos

- Node.js v14.0.0 o superior
- MongoDB v4.4 o superior
- npm v6.0.0 o superior

5.2. Variables de Entorno

El sistema utiliza las siguientes variables de entorno que deben configurarse en un archivo `.env`:

Variable	Descripción
PORT	Puerto en el que se ejecutará el servidor (por defecto: 3001)
MONGO_HOST	Host de MongoDB
MONGO_USER	Usuario de MongoDB
MONGO_PASSWORD	Contraseña de MongoDB
NODE_SECRET	Secreto para firmar las sesiones
NODE_ENV	Entorno (development, production)
NODE_EMAIL	Correo electrónico para enviar notificaciones
NODE_PASSWORD	Contraseña del correo electrónico
GOOGLE_CLIENT_ID	ID de cliente para OAuth de Google
GOOGLE_CLIENT_SECRET	Secreto de cliente para OAuth de Google
GITHUB_CLIENT_ID	ID de cliente para OAuth de GitHub
GITHUB_CLIENT_SECRET	Secreto de cliente para OAuth de GitHub
MICROSOFT_CLIENT_ID	ID de cliente para OAuth de Microsoft
MICROSOFT_CLIENT_SECRET	Secreto de cliente para OAuth de Microsoft
NGROK_URL	URL pública para desarrollo (opcional)

Cuadro 5.1: Variables de entorno

5.3. Pasos de Instalación

1. Clonar el repositorio

```
git clone https://github.com/tu-usuario/sistema-usuarios.git
cd sistema-usuarios

```

2. Instalar dependencias

```
npm install

```

3. Configurar el archivo .env con las variables necesarias

```
cp .env.example .env
# Editar el archivo .env con los valores correctos

```

4. Iniciar el servidor

```
node server.js

```

5.4. Configuración de OAuth

Para habilitar la autenticación con proveedores OAuth, siga estos pasos:

1. **Google OAuth**:
   - Acceda a la Consola de Desarrolladores de Google
   - Cree un nuevo proyecto y configure las APIs de OAuth
   - Configure el URI de redirección: `http://localhost:3001/auth/google/callback`
   - Obtenga el ID de cliente y el secreto de cliente
2. **GitHub OAuth**:
   - Acceda a GitHub Developer Settings
   - Cree una nueva aplicación OAuth
   - Configure el URI de redirección: `http://localhost:3001/auth/github/callback`
   - Obtenga el ID de cliente y el secreto de cliente
3. **Microsoft OAuth**:
   - Acceda al Portal de Azure
   - Registre una nueva aplicación en Azure Active Directory
   - Redirección: `http://localhost:3001/auth/microsoft/callback`
   - Obtenga el ID de cliente y el secreto de cliente

6. Conclusiones

6.1. Resumen del Sistema

El Sistema de Autenticación y Gestión de Usuarios implementa una solución robusta y segura para la autenticación, autorización y gestión de usuarios. Proporciona múltiples métodos de autenticación, gestión de sesiones seguras, control de acceso basado en roles y funcionalidades de restablecimiento de contraseñas.

6.2. Evaluación de Seguridad

La evaluación de seguridad realizada con SonarQube ha demostrado que el sistema cumple con los estándares básicos de seguridad, obteniendo una calificación .A.en problemas de seguridad. Sin embargo, existen áreas de mejora, especialmente en relación a los 21 puntos críticos de seguridad identificados y la baja cobertura de pruebas.

6.3. Próximos Pasos

Se recomienda implementar las siguientes mejoras para fortalecer aún más la seguridad del sistema:

- Abordar los 21 puntos críticos de seguridad identificados
- Implementar pruebas automatizadas para aumentar la cobertura
- Actualizar el sistema de hash de contraseñas a bcrypt o Argon2
- Implementar autenticación de dos factores (2FA)
- Mejorar la documentación técnica y de usuario
- Realizar pruebas de penetración adicionales

A. Glosario de Términos

Término	**Definición**
RBAC	Control de Acceso Basado en Roles. Sistema que restringe el acceso a recursos según el rol del usuario.
OAuth	Protocolo abierto de autorización que permite el acceso delegado a recursos de un servidor.
Hash	Función criptográfica que convierte datos en una cadena de caracteres de longitud fija.
2FA	Autenticación de Dos Factores. Método que requiere dos formas diferentes de autenticación.
API	Interfaz de Programación de Aplicaciones. Conjunto de reglas que permiten que diferentes programas se comuniquen entre sí.
XSS	Cross-Site Scripting. Tipo de ataque de inyección de código en sitios web.
CSRF	Cross-Site Request Forgery. Ataque que fuerza a un usuario a ejecutar acciones no deseadas en una aplicación web.
SonarQube	Plataforma para la inspección continua de la calidad del código.
MongoDB	Base de datos NoSQL orientada a documentos.
JWT	JSON Web Token. Estándar para la creación de tokens de acceso.

Cuadro A.1: Glosario de términos técnicos

Bibliografía

[1] OWASP Foundation, .ºWASP Top Ten,"2021. [Online]. Available: https://owasp.org/www-project-top-ten/

[2] Node.js Foundation, "Node.js Documentation,"2024. [Online]. Available: https://nodejs.org/en/docs/

[3] Express.js, .ExpressJS Documentation,"2024. [Online]. Available: https://expressjs.com/

[4] MongoDB, Inc., "MongoDB Documentation,"2024. [Online]. Available: https://docs.mongodb.com/

[5] Jared Hanson, "Passport.js Documentation,"2024. [Online]. Available: http://www.passportjs.org/

[6] OAuth, .ºAuth 2.0 Framework,"2024. [Online]. Available: https://oauth.net/2/

[7] SonarSource S.A, "SonarQube Documentation,"2024. [Online]. Available: https://docs.sonarqube.org/

[8] National Institute of Standards and Technology, "Digital Identity Guidelines,"Special Publication 800-63B, June 2017.

[9] Andris Reinman, "Nodemailer Documentation,"2024. [Online]. Available: https://nodemailer.com/