Vulnerabilidades comunes en aplicaciones web (OWASP Top 10) y estrategias de mitigación aplicadas en el backend del proyecto.

Autor: Joel Tapia estudiante de quinto ciclo de la carrera de Computación

I. Introducción

En el panorama digital actual, las aplicaciones de finanzas personales se han vuelto herramientas indispensables. Sin embargo, muchas fracasan al presentar barreras de entrada, como interfaces complejas o procesos de registro de datos tediosos. El proyecto GastanGO nace precisamente para resolver este problema, con un objetivo claro: desarrollar un sistema multiplataforma que facilite la gestión financiera de manera rápida, intuitiva y moderna.

El pilar de GastanGO es su experiencia móvil, diseñada para que un usuario pueda registrar un gasto o ingreso en menos de cinco segundos. Esta agilidad se logra optimizando la interacción al mínimo indispensable, permitiendo al usuario capturar la información en el momento, con la posibilidad de añadir detalles posteriormente.

Sin embargo, esta velocidad y facilidad de uso deben estar cimentadas sobre una base de seguridad robusta. La naturaleza del proyecto implica el manejo de la información más sensible de un usuario: sus datos financieros personales. La confianza es, por lo tanto, el activo más importante de la aplicación. Una brecha de seguridad no solo comprometería datos críticos, sino que destruiría la viabilidad completa del proyecto.

Este informe presenta un análisis analítico y crítico de la postura de seguridad del backend que soporta GastanGO. Utilizando como marco de referencia el estándar de la industria OWASP Top 10 (versión 2021), se identificarán las principales vulnerabilidades que pueden afectar a nuestra arquitectura. El objetivo es analizar proactivamente los riesgos presentes en el código y la configuración, documentar las medidas de mitigación implementadas y reforzar la capacidad del sistema para ser éticamente responsable y seguro.

II. Identificación de al menos 5 vulnerabilidades del OWASP Top 10 (2021) relevantes para su backend

Las 5 vulnerabilidades más relevantes del OWASP Top 10 (2021) identificadas para el backend de GastanGO son:

1. A01:2021 - Pérdida de Control de Acceso (Broken Access Control)
2. A02:2021 - Fallos Criptográficos (Cryptographic Failures)
3. A03:2021 - Inyección (Injection)
4. A05:2021 - Configuración de Seguridad Incorrecta (Security Misconfiguration)
5. A07:2021 - Fallos de Identificación y Autenticación

III. Descripción del riesgo y del posible impacto de cada vulnerabilidad en su sistema

A01:2021 - Pérdida de Control de Acceso:
Riesgo: Un usuario autenticado (Usuario A) podría acceder, modificar o eliminar los recursos de otro usuario (Usuario B). Esto sucedería si los endpoints de la API (ej. GET /api/transactions/123) solo validan la autenticación (un JWT válido) pero no la autorización (si la transacción 123 pertenece al usuario A).
Impacto: Es el riesgo más crítico para GastanGO. Resultaría en una fuga masiva de datos financieros privados, la corrupción de la información y la pérdida total de confianza en la plataforma.
A02:2021 - Fallos Criptográficos:
Riesgo: El riesgo se presenta en dos frentes:
En Reposo: Almacenar las contraseñas de los usuarios en texto plano o con un hash débil (como MD5) en la base de datos PostgreSQL.
En Tránsito: No utilizar HTTPS para la comunicación entre la app y la API.
Impacto: Una brecha en la base de datos (por Inyección SQL, por ejemplo) expondría las credenciales de todos los usuarios. La falta de HTTPS permitiría a un atacante en la misma red (Man-in-the-Middle) robar los JWT y los datos financieros.
A03:2021 - Inyección:
Riesgo: Principalmente, Inyección SQL. Ocurre si la entrada del usuario (ej. una nota de texto, un ID, o un rango de fechas) se concatena directamente en una consulta SQL.
Impacto: Un atacante podría extraer datos de otros usuarios, eludir la autenticación, o incluso borrar toda la base de datos PostgreSQL (ej. DROP TABLE users;). Esto afectaría directamente a los componentes Transaction Controller y Reporting Service.
A05:2021 - Configuración de Seguridad Incorrecta:
Riesgo: El framework Express, por defecto, puede exponer información sensible o tener configuraciones permisivas. Los riesgos incluyen:
Cabeceras HTTP que revelan la tecnología (ej. X-Powered-By: Express).
Mensajes de error detallados (stack traces) en producción, revelando la estructura del código.
Políticas de CORS (Cross-Origin Resource Sharing) demasiado abiertas (ej. Access-Control-Allow-Origin: *).
Impacto: Facilita a los atacantes el reconocimiento del sistema para planificar ataques más sofisticados. Errores detallados pueden filtrar claves o rutas de archivos, y un CORS mal configurado permitiría a sitios web maliciosos interactuar con la API.
A07:2021 - Fallos de Identificación y Autenticación: Riesgo: Relacionado con la gestión del ciclo de vida de la autenticación. El riesgo principal es que los JWTs no expiren o tengan una vida útil muy larga (ej. 30 días).

Impacto: Si un JWT es robado (por XSS en la web o un dispositivo móvil comprometido), el atacante tendría acceso indefinido a la cuenta del usuario, incluso si este cambia su contraseña, hasta que el token expire.

IV. Medidas de mitigación implementadas o planificadas, explicando cómo se aplicaron en el código o la configuración

A01:2021 - Pérdida de Control de Acceso:
Medida: Se implementa una validación de propiedad en todos los componentes del backend que acceden a recursos.
Aplicación: Después de validar el JWT (autenticación), se extrae el userId del payload del token. Este userId se usa

obligatoriamente en la cláusula WHERE de todas las consultas SQL.
Por ejemplo:
SELECT * FROM transactions WHERE id = $1 AND user_id = $2. Esto asegura que un usuario solo pueda ver o modificar sus propios registros.

A02:2021 - Fallos Criptográficos:
Medida: Hashing de contraseñas y cifrado en tránsito.
Aplicación:
En Reposo: Se utiliza la biblioteca bcrypt en el Authentication Controller para generar un salt y hashear la contraseña antes de guardarla en PostgreSQL.
En Tránsito: El servidor Node.js/Express se configura para operar exclusivamente sobre HTTPS (TLS), cifrando toda la comunicación de la API.

A03:2021 - Inyección:
Medida: Consultas parametrizadas (Prepared Statements).
Aplicación: Se utiliza una librería cliente de PostgreSQL (como node-postgres) que soporta consultas parametrizadas. En lugar de concatenar la entrada del usuario en el string de la consulta, se pasa como un array de valores. Esto asegura que la base de datos trate la entrada siempre como datos y nunca como código SQL ejecutable.

A05:2021 - Configuración de Seguridad Incorrecta:
Medida: Endurecimiento (Hardening) del servidor Express.
Aplicación:
Se añade el middleware helmet (app.use(helmet());), que configura varias cabeceras HTTP de seguridad automáticamente.
Se deshabilita la cabecera que expone la tecnología con app.disable('x-powered-by');.
Se configura el middleware cors para permitir peticiones solo desde los dominios conocidos de la App Web.
Se implementa un manejador de errores global que no filtra stack traces en producción.

A07:2021 - Fallos de Identificación y Autenticación:
Medida: Expiración corta de tokens JWT.
Aplicación: Al generar los access tokens (JWT) en el Authentication Controller, se les asigna una expiración corta (ej. 15 minutos o 1 hora).
Planificado: Implementar refresh tokens (con expiración larga, almacenados de forma segura) que permitan renovar los access tokens sin requerir que el usuario inicie sesión constantemente.

V. Evidencias de verificación: capturas de pruebas, fragmentos de código o resultados de herramientas de análisis

Evidencia para A01 (Control de Acceso):

```
const userId = req.user.id;
const { transactionId } = req.params;

db.query(
 'SELECT * FROM transactions WHERE id = $1 AND user_id = $2',
 [transactionId, userId],
 (err, results) => {
   // Si results.rows.length === 0, no se encontró O no le pertenece
   if (results.rows.length === 0) {
     return res.status(404).send('Transacción no encontrada.');
   }
   res.json(results.rows[0]);
 }
);
```

Evidencia para A07 (Fallos de Autenticación):
Fragmento de código de la generación del JWT:

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

jwt.sign(
 { id: user.id, email: user.email },
 process.env.JWT_SECRET, // Clave secreta
 { expiresIn: '15m' }   // Expiración corta
);
```

VI. Reflexión personal

Durante esta unidad entendí que la seguridad no es algo que se agrega al final, sino una base que debe estar presente desde el primer diseño del sistema. Me quedó claro que buscar rapidez, como en el caso de GastanGO, puede jugar en contra si no se validan bien los datos en el backend. Cada endpoint de la API es una posible puerta de entrada, y asegurarla depende completamente de cómo la construyamos.

Aunque usamos JWT y bcrypt, aprendí que eso solo cubre parte de los riesgos (como A02 y A07). Los problemas más graves, como el control de acceso (A01) o las inyecciones (A03), se reducen realmente si el código y la comunicación con la base de datos están bien diseñados.

Si tuviera que mejorar el diseño a futuro, me enfocaría en dos puntos claves:

Rate Limiting: limitar los intentos fallidos (por ejemplo, 5 logins fallidos por minuto) para evitar ataques de fuerza bruta contra el controlador de autenticación.

Monitoreo inteligente: implementar un sistema de registro que detecte patrones sospechosos, como un usuario accediendo a recursos que no le pertenecen o errores repetitivos en la base de datos.

VII. Preguntas Orientadoras

¿Qué vulnerabilidad del OWASP Top 10 representa el mayor riesgo para tu proyecto y por qué?
La vulnerabilidad de mayor riesgo es A01:2021 - Pérdida de Control de Acceso.
El valor central de GastanGO es la gestión de finanzas personales. La confianza del usuario se basa en la premisa de que sus datos son confidenciales. Un fallo de A03 (Inyección) es catastrófico, pero un fallo de A01 es más sutil y rompe el modelo de negocio. Si un usuario puede, modificando un ID en la URL, acceder a los reportes o transacciones de otro usuario, la aplicación es funcionalmente inútil y una grave violación de la privacidad. Este fallo ocurriría directamente en la lógica de negocio de los componentes Transaction Controller y Reporting Service.

¿Cómo validas que las medidas implementadas realmente mitigan la vulnerabilidad detectada?

La validación se realiza mediante pruebas de integración automatizadas y pruebas de penetración manuales:
Pruebas de Integración (A01): Creamos dos usuarios de prueba (Usuario A y Usuario B) con sus respectivos JWTs. El Usuario A crea una transacción (ej. ID t-123).
Prueba 1: Usuario A solicita GET /api/transactions/t-123. Resultado esperado: 200 OK.
Prueba 2: Usuario B (con su propio JWT) solicita GET /api/transactions/t-123. Resultado esperado: 404 Not Found (o 403 Forbidden). Si devuelve 200 OK, la mitigación falló.

Pruebas Manuales (A03): Utilizamos herramientas como Postman o cURL para enviar payloads de Inyección SQL en los campos de entrada (ej. al añadir detalles a un gasto).
Payload: { "details": "Gasto en...' OR 1=1; --" }

Resultado esperado: La aplicación debe guardar la cadena de texto literalmente en la base de datos o devolver un error 400 (Bad Request) por validación, pero nunca debe ejecutar la consulta ni devolver un error 500 de la base de datos.

Análisis de Cabeceras (A05): Usamos las herramientas de desarrollador del navegador o curl -v para inspeccionar las cabeceras de respuesta del Backend API. Verificamos que la cabecera X-Powered-By no esté presente y que las cabeceras de helmet (como X-Content-Type-Options: nosniff) sí lo estén.

¿Qué relación encuentras entre las vulnerabilidades OWASP y los componentes del modelo C4 (backend, base de datos, API Gateway)?

El modelo C4 [fuentes 100-207] es excelente para mapear las responsabilidades de seguridad:

Contenedor Backend API (Nivel 2): Es la principal superficie de ataque y la primera línea de defensa. Es responsable de mitigar casi todo el OWASP Top 10. Implementa las mitigaciones para A05 (Configuración de Seguridad, cabeceras) y A07 (Autenticación, validación de JWT).

Componentes (Nivel 3): El modelo C4 nos permite ser específicos:

El Authentication Controller es el dueño de mitigar A02 (usando bcrypt) y A07 (gestionando JWTs).
El Transaction Controller y el Reporting Service son los responsables directos de mitigar A01 (Control de Acceso, filtrando por user_id) y A03 (Inyección, usando consultas parametrizadas).
Contenedor Base de Datos (Nivel 2): Es el "activo" que protegemos. Es la víctima de A03 (Inyección) y la fuente de datos que se exfiltra en A01 (Pérdida de Control de Acceso). Su propia configuración (A05) también es vital (ej. permisos de usuario de BD limitados).
Relaciones (Flechas): Las flechas HTTPS/JSON que conectan los frontends con la API son el punto exacto donde A02 (Fallos Criptográficos en tránsito) se mitiga con HTTPS. La flecha JDBC de la API a la Base de Datos es el canal que protegemos contra A03 (Inyección)

VIII. Bibliografía

1] OWASP Foundation, OWASP Top 10 2021. [Online]. Disponible: https://owasp.org/www-project-top-ten/2021/es/

[2] M. Jones, J. Bradley, y N. Sakimura, "JSON Web Token (JWT)," Internet Engineering Task Force (IETF), RFC 7519, mayo 2015. [Online]. Disponible: https://tools.ietf.org/html/rfc7519

[3] Express Team, "Express - Framework de aplicaciones web Node.js," expressjs.com. [Online]. Disponible: https://expressjs.com/es/ (Consultado: 19 de octubre de 2025).

[4] N. Provos y D. Mazières, "A future-adaptable password scheme" (Un esquema de contraseñas adaptable al futuro), en Proc. 1999 USENIX Annual Technical Conference, 1999, pp. 81-92.

[5] The PostgreSQL Global Development Group, Documentación de PostgreSQL 16. [Online]. Disponible: https://www.postgresql.org/docs/16/index.html (Consultado: 19 de octubre de 2025).