

EJEMPLO INTEGRADO DE UNA API REST SEGURA

Node.js/Express implementación práctica de los elementos de seguridad en aplicaciones.

```
```javascript
```

```
// app.js - API Segura con mejores prácticas
const express = require('express');
const helmet = require('helmet');
const rateLimit = require('express-rate-limit');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const crypto = require('crypto');
const winston = require('winston');
const { body, validationResult, param } = require('express-validator');
```

```
// Configuración de logging segura
```

```
const logger = winston.createLogger({
 level: 'info',
 format: winston.format.combine(
 winston.format.timestamp(),
 winston.format.json() // No loguear datos sensibles
),
 transports: [
 new winston.transports.File({ filename: 'security.log' })
]
});
```

```
// Modelo de usuario (simulado)
```

```
const users = [
 {
 id: 1,
 email: 'admin@empresa.com',
 passwordHash: '$2a$10$N9qo8uLOickgxZMRZoMy.Mrq4Yp3R.ZE6k2QkU.OVn1ZTRpPuJyW',
 // password123
```

**Comentado [LP1]:** Valores solo de ejemplo, ya que NO se ponen dentro del código sino en tablas  
La contraseña no se la guarda en texto plano

```
 role: 'admin',
 twoFactorEnabled: true
 }
];

// Almacenamiento seguro de sesiones (en memoria para el ejemplo)
const activeSessions = new Map();

class SecurityManager {
 // 1. AUTENTICACIÓN Y AUTORIZACIÓN

 static async authenticateUser(email, password) {
 const user = users.find(u => u.email === email);

 if (!user) {
 logger.warn(`Intento de login fallido para email: ${email}`);
 throw new Error('Credenciales inválidas');
 }

 const isValid = await bcrypt.compare(password, user.passwordHash);
 if (!isValid) {
 logger.warn(`Password incorrecto para usuario: ${email}`);
 throw new Error('Credenciales inválidas');
 }

 return user;
 }

 static generateToken(user) {
 const sessionId = crypto.randomBytes(16).toString('hex');

 const token = jwt.sign(
 {
 userId: user.id,
```

```
 role: user.role,
 sessionId: sessionId
 },
 process.env.JWT_SECRET,
 { expiresIn: '1h' }
);

// Almacenar sesión
activeSessions.set(sessionId, {
 userId: user.id,
 createdAt: new Date(),
 lastActivity: new Date()
});

return token;
}

// 2. CONTROL DE ACCESO
static authorize(requiredRole) {
 return (req, res, next) => {
 try {
 if (!req.user) {
 return res.status(401).json({ error: 'No autenticado' });
 }

 const userRole = req.user.role;
 const rolesHierarchy = { user: 1, manager: 2, admin: 3 };

 if (rolesHierarchy[userRole] < rolesHierarchy[requiredRole]) {
 logger.warn(`Intento de acceso no autorizado. Usuario: ${req.user.userId}, Ruta: ${req.path}`);
 return res.status(403).json({ error: 'Acceso denegado' });
 }
 }
 };
}
```

```
}

 next();
} catch (error) {
 next(error);
}
};

}

// 3. GESTIÓN DE SESIONES

static validateSession(sessionId) {
 const session = activeSessions.get(sessionId);
 if (!session) return false;

 // Verificar timeout (15 minutos de inactividad)
 const now = new Date();
 const inactiveTime = now - session.lastActivity;
 if (inactiveTime > 15 * 60 * 1000) {
 activeSessions.delete(sessionId);
 return false;
 }

 // Actualizar última actividad
 session.lastActivity = now;
 return true;
}

static logout(sessionId) {
 activeSessions.delete(sessionId);
 logger.info(`Sesión cerrada: ${sessionId}`);
}
}
```

```
// Middleware de autenticación JWT
const authenticateToken = (req, res, next) => {
 const authHeader = req.headers['authorization'];
 const token = authHeader && authHeader.split(' ')[1];

 if (!token) {
 return res.status(401).json({ error: 'Token requerido' });
 }

 jwt.verify(token, process.env.JWT_SECRET, (err, user) => {
 if (err) {
 logger.warn(`Token inválido: ${err.message}`);
 return res.status(403).json({ error: 'Token inválido' });
 }

 // Validar sesión activa
 if (!SecurityManager.validateSession(user.sessionId)) {
 return res.status(403).json({ error: 'Sesión expirada' });
 }

 req.user = user;
 next();
 });
};

// 4. VALIDACIÓN DE ENTRADAS
const userValidationRules = {
 login: [
 body('email').isEmail().normalizeEmail(),
 body('password').isLength({ min: 8 })
],
};
```

```
createUser: [
 body('email').isEmail().normalizeEmail(),
 body('password').isStrongPassword({
 minLength: 8,
 minLowercase: 1,
 minUppercase: 1,
 minNumbers: 1,
 minSymbols: 1
 }),
 body('role').isIn(['user', 'manager', 'admin'])
],
userId: [
 param('id').isInt({ min: 1 })
]
};

// 5. MANEJO SEGURO DE ERRORES
class AppError extends Error {
 constructor(message, statusCode, isOperational = true) {
 super(message);
 this.statusCode = statusCode;
 this.isOperational = isOperational;

 Error.captureStackTrace(this, this.constructor);
 }
}

const errorHandler = (err, req, res, next) => {
 let error = { ...err };
 error.message = err.message;

 // Log del error
```

```
logger.error({
 message: err.message,
 stack: err.stack,
 path: req.path,
 method: req.method,
 ip: req.ip,
 user: req.user ? req.user.userId : 'anon'
});

// Errores de JWT
if (err.name === 'JsonWebTokenError') {
 const error = new AppError('Token inválido', 401);
 return res.status(error.statusCode).json({ error: error.message });
}

if (err.name === 'TokenExpiredError') {
 const error = new AppError('Token expirado', 401);
 return res.status(error.statusCode).json({ error: error.message });
}

// Errores de validación
if (err.name === 'ValidationError') {
 const error = new AppError('Datos de entrada inválidos', 400);
 return res.status(error.statusCode).json({ error: error.message });
}

// Error por defecto
const statusCode = error.statusCode || 500;
const message = statusCode === 500 ? 'Error interno del servidor' : error.message;

res.status(statusCode).json({
 error: message
```

```
});

};

// Configuración de la aplicación
const app = express();

// 6. MIDDLEWARES DE SEGURIDAD
app.use(helmet({
 contentSecurityPolicy: {
 directives: {
 defaultSrc: ["'self'"],
 scriptSrc: ["'self'"],
 styleSrc: ["'self'", "'unsafe-inline'"],
 imgSrc: ["'self'", "data:", "https:"]
 }
 },
 hsts: {
 maxAge: 31536000,
 includeSubDomains: true,
 preload: true
 }
}));

// Rate limiting
const limiter = rateLimit({
 windowMs: 15 * 60 * 1000, // 15 minutos
 max: 100, // máximo 100 peticiones por ventana
 message: {
 error: 'Demasiadas peticiones desde esta IP'
 }
});

app.use(limiter);
```

```
app.use(express.json({ limit: '10kb' })); // Limitar tamaño de payload

// 7. RUTAS SEGURAS
app.post('/api/auth/login',
 userValidationRules.login,
 async (req, res, next) => {
 try {
 // Validar entrada
 const errors = validationResult(req);
 if (!errors.isEmpty()) {
 return res.status(400).json({ errors: errors.array() });
 }
 }

 const { email, password } = req.body;

 // Autenticar usuario
 const user = await SecurityManager.authenticateUser(email, password);

 // Generar token
 const token = SecurityManager.generateToken(user);

 // Log de login exitoso
 logger.info(`Login exitoso para usuario: ${email}`);

 res.json({
 token,
 user: {
 id: user.id,
 email: user.email,
 role: user.role
 }
 })
 }
)
```

```
});

} catch (error) {
 next(error);
}

};

);

app.post('/api/auth/logout', authenticateToken, (req, res) => {
 SecurityManager.logout(req.user.sessionId);
 res.json({ message: 'Sesión cerrada exitosamente' });
});

// 8. PROTECCIÓN DE DATOS - Ejemplo con datos sensibles
app.get('/api/users/:id',
 authenticateToken,
 SecurityManager.authorize('admin'),
 userValidationRules.userId,
 (req, res, next) => {
 try {
 const errors = validationResult(req);
 if (!errors.isEmpty()) {
 return res.status(400).json({ errors: errors.array() });
 }
 }

 const userId = parseInt(req.params.id);
 const user = users.find(u => u.id === userId);

 if (!user) {
 return res.status(404).json({ error: 'Usuario no encontrado' });
 }
}
```

```
// Sanitizar salida - no enviar datos sensibles
const sanitizedUser = {
 id: user.id,
 email: user.email,
 role: user.role
 // No incluimos passwordHash, sessions, etc.
};

res.json(sanitizedUser);

} catch (error) {
 next(error);
}

};

// 9. MANEJO SEGURO DE OPERACIONES CRÍTICAS
app.post('/api/transfer',
 authenticateToken,
 SecurityManager.authorize('user'),
 [
 body('amount').isFloat({ min: 0.01, max: 10000 }),
 body('destinationAccount').isLength({ min: 10, max: 34 })
],
 async (req, res, next) => {
 try {
 const errors = validationResult(req);
 if (!errors.isEmpty()) {
 return res.status(400).json({ errors: errors.array() });
 }
 }

 const { amount, destinationAccount } = req.body;
```

```
// Log de operación sensible
logger.info(`Transferencia iniciada. Usuario: ${req.user.userId}, Monto: ${amount}`);

// Simular procesamiento seguro
// En un caso real, aquí irían transacciones de base de datos, etc.

res.json({
 message: 'Transferencia procesada exitosamente',
 transactionId: crypto.randomBytes(8).toString('hex')
});

} catch (error) {
 next(error);
}

};

// Middleware de error debe ir al final
app.use(errorHandler);

// 10. MODELADO DE AMENAZAS - Ejemplo de protección adicional
// (Esto se basa en el análisis de amenazas realizado previamente)

// Protección contra fuerza bruta específica
const loginLimiter = rateLimit({
 windowMs: 15 * 60 * 1000,
 max: 5, // Máximo 5 intentos de login
 message: {
 error: 'Demasiados intentos de login, intente más tarde'
 }
});
```

```
app.use('/api/auth/login', loginLimiter);

// Headers de seguridad adicionales
app.use((req, res, next) => {
 res.removeHeader('X-Powered-By'); // Ocultar tecnología
 res.header('X-Content-Type-Options', 'nosniff');
 res.header('X-Frame-Options', 'DENY');
 next();
});

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
 console.log(`Servidor seguro ejecutándose en puerto ${PORT}`);
 logger.info(`Servidor iniciado en puerto ${PORT}`);
});

module.exports = app;
...

🔒 **Archivo de Configuración de Seguridad (.env)**
```env
# Configuración de Seguridad
JWT_SECRET=tu_clave_super_secreta_muy_larga_y_compleja_aqui_32_caracteres_min
NODE_ENV=production
PORT=3000

# Base de datos
DB_HOST=localhost
DB_USER=usuario_seguro
DB_PASSWORD=password_complejo_123

# CORS
```

ALLOWED_ORIGINS=https://tudominio.com

...