Security Policy

Security Improvements Summary

This document outlines the security improvements made to the Purchase Requisition System.

Critical Vulnerabilities Fixed

1. ✅ Plaintext Password Storage

Severity: CRITICAL - CVE-like: CWE-256

Issue:

- Passwords were stored in plaintext in the database
- Authentication compared passwords directly without hashing

Fix:

- Implemented bcrypt password hashing with 10 salt rounds
- Created password hashing utility (`backend/utils/auth.js`)
- Added script to hash existing passwords (`backend/scripts/hashPasswords.js`)
- Updated login endpoint to use `comparePassword()`

Files Modified:

- `backend/server.js` (lines 189-230)
- `backend/utils/auth.js` (new file)
- `backend/scripts/hashPasswords.js` (new file)

---

2. ✅ No Authentication/Authorization

Severity: CRITICAL - CVE-like: CWE-306

Issue:

- All API endpoints were publicly accessible
- No token-based authentication
- No role-based access control

Fix:

- Implemented JWT (JSON Web Token) authentication
- Created authentication middleware (`backend/middleware/auth.js`)
- Created authorization middleware with role checking
- Protected all API endpoints with `authenticate` middleware
- Added role-based restrictions using `authorize` middleware

Protected Endpoints:

- All `/api/requisitions/*` endpoints now require authentication
- Role-specific endpoints restricted (e.g., only HOD can approve at HOD stage)
- Admin role can access all endpoints

Files Modified:

- `backend/middleware/auth.js` (new file)
- `backend/server.js` (all endpoint definitions)
- `frontend/app.js` (API client updated)

---

3. ✅ SQL Injection Vulnerability

Severity: HIGH - CVE-like: CWE-89

Issue:

- String interpolation used in SQL queries (line 510 in original server.js)
- User input directly concatenated into SQL

```
// VULNERABLE CODE (REMOVED)
queries[key] += ` ${key === 'total' ? 'WHERE' : 'AND'} created_by = ${user_id}`;
```

Fix:

- Converted to parameterized queries
- All user inputs now passed as query parameters
- SQL injection no longer possible

```
// SECURE CODE (NEW)
queries.total.sql += " WHERE created_by = ?";
queries.total.params.push(user_id);
```

Files Modified:

- `backend/server.js` (lines 546-605)

---

4. ☑ CORS Misconfiguration

Severity: MEDIUM - CVE-like: CWE-942

Issue:

- CORS configured to allow all origins (`*`)
- Allowed requests from any domain

Fix:

- Implemented whitelist-based CORS
- Only allowed origins from environment variable
- Default safe origins for development

Files Modified:

- `backend/server.js` (lines 19-36)
- `backend/.env` (ALLOWED_ORIGINS variable)

---

5. ☑ No Input Validation

Severity: MEDIUM - CVE-like: CWE-20

Issue:

- No server-side validation of user inputs
- Malformed data could crash the server
- No email/format validation

Fix:

- Implemented express-validator middleware
- Created comprehensive validation rules
- Validation for login, requisition creation, user creation, etc.
- Email format validation
- Password strength requirements

Files Modified:

- `backend/middleware/validation.js` (new file)
- `backend/server.js` (added validation to endpoints)

---

6. ☑ Poor Error Handling

Severity: MEDIUM - CVE-like: CWE-209

Issue:

- Generic error messages exposing internal details
- Database errors returned directly to client
- No centralized error handling

Fix:

- Created centralized error handler middleware
- Custom AppError class
- Stack traces hidden in production
- Consistent error response format

Files Modified:

- `backend/middleware/errorHandler.js` (new file)
- `backend/server.js` (added error handlers)

---

7. ☑ Hardcoded Secrets

Severity: MEDIUM - CVE-like: CWE-798

Issue:

- No environment configuration
- Hardcoded ports and database paths
- No separation of dev/prod configs

Fix:

- Created .env file for configuration
- Implemented dotenv for environment variables
- JWT_SECRET configurable
- ALLOWED_ORIGINS configurable
- Database path configurable

Files Modified:

- `backend/.env` (new file)
- `backend/server.js` (uses process.env)

---

8. ☑ Sensitive Files in Git

Severity: MEDIUM - CVE-like: CWE-540

Issue:

- No .gitignore file
- Database files would be committed
- .env file would be exposed
- node_modules would be committed

Fix:

- Created comprehensive .gitignore
- Excludes .env, *.db, node_modules
- Excludes IDE and OS files

Files Modified:

- `.gitignore` (new file)

Security Features Added

Authentication System

- JWT-based authentication
- Token expiration (24 hours, configurable)
- Bearer token in Authorization header
- LocalStorage for token persistence
- Token cleared on logout

Authorization System

- Role-based access control (RBAC)
- 6 roles: initiator, hod, procurement, finance, md, admin
- Middleware-based permission checking
- Admin has full access

Password Security

- Bcrypt hashing (10 rounds, configurable)
- Minimum password length: 8 characters
- Password strength requirements (uppercase, lowercase, number)
- Secure password comparison

Input Validation

- Username: 3-50 characters, alphanumeric + special chars
- Email: Valid email format, normalized
- Phone: Valid phone number format
- Quantity: Positive integers
- Amounts: Positive numbers

Security Configuration

Environment Variables

```
# Required
JWT_SECRET=your-secure-random-secret-key
ALLOWED_ORIGINS=http://localhost:3002

# Optional (with defaults)
NODE_ENV=development
PORT=3001
JWT_EXPIRES_IN=24h
BCRYPT_ROUNDS=10
```

Generating Secure JWT Secret

```
node -e "console.log(require('crypto').randomBytes(32).toString('hex'))"
```

Testing Security

Password Hashing Test

```
cd backend
node scripts/hashPasswords.js
```

Authentication Test

```
# Login
curl -X POST http://localhost:3001/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"username":"john.banda","password":"password123"}'

# Use token from response
curl http://localhost:3001/api/requisitions \
  -H "Authorization: Bearer YOUR_TOKEN_HERE"
```

SQL Injection Test (Should fail safely)

```
curl "http://localhost:3001/api/stats?user_id=1%20OR%201=1" \
  -H "Authorization: Bearer YOUR_TOKEN"
```

Remaining Security Recommendations

High Priority

1. **Rate Limiting**
    - Add express-rate-limit
    - Limit login attempts
    - Prevent brute force attacks
2. **HTTPS**
    - Use HTTPS in production
    - Force HTTPS redirect
    - Secure cookies
3. **Database Migration**
    - Move from SQLite to PostgreSQL/MySQL
    - Enable SSL for database connections
    - Use connection pooling
4. **Logging & Monitoring**
    - Implement Winston/Pino logging
    - Log authentication attempts
    - Monitor suspicious activity
    - Set up alerts
5. **Session Management**
    - Implement refresh tokens
    - Add token revocation
    - Track active sessions

Medium Priority

6. **CSRF Protection**
    - Add CSRF tokens for state-changing operations
    - Use csurf middleware
7. **XSS Prevention**
    - Add helmet.js middleware
    - Content Security Policy headers
    - Sanitize user inputs
8. **File Upload Security** (if implemented)
    - Validate file types
    - Limit file sizes
    - Scan for malware
    - Store outside web root
9. **API Versioning**
    - Version API endpoints (/api/v1/)
    - Maintain backward compatibility
    - Deprecation notices
10. **Dependency Scanning**
    - Run `npm audit` regularly
    - Update dependencies
    - Use Snyk or similar tools

Low Priority

11. **2FA/MFA**
    - Two-factor authentication
    - TOTP support
    - Backup codes
12. **Account Security**

    - Password reset functionality
    - Account lockout after failed attempts
    - Email verification
    - Security questions

13. **Audit Improvements**

    - More detailed audit logs
    - IP address tracking
    - User agent logging
    - Export audit reports

Security Checklist for Deployment

- [ ] Change all default passwords
- [ ] Generate strong JWT_SECRET
- [ ] Set NODE_ENV=production
- [ ] Update ALLOWED_ORIGINS to production domain
- [ ] Enable HTTPS
- [ ] Use production database (PostgreSQL/MySQL)
- [ ] Set up database backups
- [ ] Implement rate limiting
- [ ] Add logging and monitoring
- [ ] Run security audit (`npm audit`)
- [ ] Update all dependencies
- [ ] Review and test error handling
- [ ] Set up SSL/TLS certificates
- [ ] Configure firewall rules
- [ ] Set up intrusion detection
- [ ] Document security procedures

Reporting Security Issues

If you discover a security vulnerability, please:

1. **DO NOT** open a public issue
2. Email: security@yourcompany.com
3. Include:
    - Description of the vulnerability
    - Steps to reproduce
    - Potential impact
    - Suggested fix (if any)

We will acknowledge receipt within 48 hours and provide a timeline for fix.

Security Update History

Date	Version	Changes
2025-10-22	2.0.0	Major security overhaul - password hashing, JWT auth, SQL injection fixes, input validation, CORS config, error handling
Initial	1.0.0	Initial insecure version

Compliance

This system is designed to meet the following security standards:

- ✅ OWASP Top 10 (2021) - Most issues addressed
- ⚠️ PCI DSS - Partial (not storing payment data)
- ⚠️ GDPR - Partial (data protection basics in place)
- ⚠️ SOC 2 - Partial (audit logging implemented)

Note: Full compliance requires additional controls and documentation.

Security Contact: security@yourcompany.com
Last Security Audit: 2025-10-22
Next Audit Due: 2025-11-22