

# Assure DeFi<sup>®</sup>

THE VERIFICATION **GOLD STANDARD**



## Security Assessment

## SHARBI FUN APP

Date: 28/07/2025

Audit Status: PASS





Audit Edition: Code audit



ASSURE DEFI<sup>®</sup>  
THE VERIFICATION **GOLD STANDARD**

# Risk Analysis

## Vulnerability summary

Classification	Description
 High	Vulnerabilities that lead to direct compromise of critical assets, large-scale data exposure, unauthorized fund transfers, or full system takeover.
 Medium	Flaws that weaken security posture or privacy but do not immediately enable catastrophic failures.
 Low	Issues that have minimal direct impact, often involving best-practice deviations or potential future risks.
 Informational	Observations, style concerns, or suggestions that do not constitute vulnerabilities but may improve security hygiene.

# Scope

## Target Code And Revision

Project	Assure
Codebase	<a href="https://github.com/dappgenie/sharbi.fun-app">https://github.com/dappgenie/sharbi.fun-app</a> Commit: <a href="#">621c0a2b493ee5194a4d97b5a021202eaf158ad7</a>  Commit: <a href="#">883890ecb776066a8b569bc6d6c9c3753d7fc571</a>
Audit Methodology	Static, Manual



# Detailed Technical Report



## 1. JWT Stored in localStorage Leading to XSS Token Theft [Fixed ✓]

**Location:** src/app/actions/api/api.ts

**Issue:** The application stores the bearer JWT in localStorage, making it accessible to any JavaScript running on the page. If an attacker succeeds in injecting a script (for example via an XSS vulnerability), they can exfiltrate the token and impersonate the user.

**Remediation:** Move token storage to an HttpOnly, Secure, SameSite=strict cookie set by the server.

Update client calls to remove localStorage.getItem('token') and rely on the cookie for authentication.

Harden the backend to issue and refresh tokens via the cookie rather than exposing them to client-side code.

**Fix:** Not in localStorage anymore, as an additional recommendation, you should issue/set HttpOnly cookie server-side and stop reading the token in client code.

## 2. Service Worker Caches Dynamic API Responses and Auth Endpoints [Fixed ✓]

**Location:** next.config.mjs (PWA plugin configuration)

**Issue:** The default next-pwa setup writes the service worker to public/ without any runtime caching rules. As a result, API calls (e.g., /auth/login, /uploads/file, /api/\*) may be cached and served stale or offline, potentially leaking sensitive data or serving unauthorized content.

**Remediation:** In your PWA config, define runtimeCaching to exclude all authentication and API routes. For example:

```
const withPWA = withPWAInit({
  /* ... */
  runtimeCaching: [
    {
      urlPattern: /^\/(auth|api|uploads)\//,
      handler: 'NetworkOnly'
    },
    {
      urlPattern: /\.?(?:png|jpg|jpeg|svg|gif|webp)$/,
      handler: 'CacheFirst'
    }
  ]
})
```

```
]
});
```

**Fix:** runtimeCaching includes: { urlPattern: /^\/(auth|api|uploads)\//, handler: 'NetworkOnly' }

### **3. Build Configuration Ignores Lint and TypeScript Errors [Fixed**

**Location:** next.config.mjs

**Issue:** The settings `eslint.ignoreDuringBuilds=true` and `typescript.ignoreBuildErrors=true` allow production builds to succeed despite linting and type errors. This can mask security-critical mistakes such as unhandled exceptions or improper typings that would otherwise be caught before deployment.

**Remediation:** Remove or disable both `ignoreDuringBuilds` and `ignoreBuildErrors` flags in `next.config.mjs`.

Enforce lint and type checks in CI/CD, failing the build on any errors.

**Fix:** `next.config.mjs` -> `eslint.ignoreDuringBuilds: false` and `typescript.ignoreBuildErrors: false`. PD: I didn't see a CI job that fails the build on lint/type errors (only security workflows). Consider adding one.

#### 4. Missing Content Security Policy (CSP) Configuration [Fixed

**Location:** next.config.mjs

**Issue:** No CSP headers or <meta> tags are configured, leaving the application vulnerable to XSS by allowing inline scripts/styles and loading resources from untrusted origins.

**Remediation:** Add a strict CSP via Next.js' headers() API. For example:

```
async headers() {
  return [
    {
      source: '/*.*',
      headers: [
        {
          key: 'Content-Security-Policy',
          value: [
            "default-src 'self'",
            "script-src 'self'",
            "style-src 'self' 'unsafe-inline'",
            "img-src 'self' data: https://utfs.io",
            "connect-src 'self' " + process.env.NEXT_PUBLIC_API_URL,
          ].join('; ')
        }
      ]
    }
  ];
}
```

**Fix:** next.config.mjs adds a Content-Security-Policy header.

Note: it allows 'unsafe-inline' 'unsafe-eval' for scripts/styles in the snippet I see, which weakens XSS protection. Tighten if possible



MEDIUM

### **1. JWT Signature and Expiration Not Verified on Client [Acknowledge]**

**Location:** src/components/isAuth.tsx

**Issue:** The HOC decodes tokens using jose.decodeJwt() which does not verify signature or expiration then simply checks for existence. An attacker could supply a forged or expired token to bypass client-side gating of protected UIs.

**Remediation:** Use jose.jwtVerify() with the backend's public key to ensure signature validity and that exp has not passed.

Keep critical route protection server-side; rely on API responses rather than client-only checks.

### **2. Container Runs as Root User [Fixed ✓]**

**Location:** Dockerfile

**Issue:** The final image uses the default root user, increasing the blast radius if the container is compromised.

**Remediation:** Add a non-root user and switch before CMD. For example:

```
RUN addgroup -S appgroup && adduser -S appuser -G appgroup
USER appuser
```

**Fix:** Fix: Dockerfile -> creates non-root user and switches with USER appuser.

### **3. Broad SemVer Ranges Without Vulnerability Scanning [Fixed ✓]**

**Location:** package.json & bun.lockba

**Issue:** Many dependencies use caret (^) ranges, which can pull in breaking or vulnerable updates. There's no automated CVE scanning configured.

**Remediation:** Pin critical dependencies to exact versions.

Integrate a tool like Dependabot, Snyk, or GitHub Code Scanning to catch new vulnerabilities.

**Fix:** package.json → app deps are pinned to exact versions (no ^/~ in dependencies/devDependencies).

Security automation present:

- .github/workflows/security.yml (audit & dependency review)
- .github/dependabot.yml (security-only updates)
- SECURITY-STATUS.md present.

#### **4. Malformed NEXT\_PUBLIC\_API\_URL Could Lead to Unintended Targets [Fixed ✓]**

**Location:** .env.example

**Issue:** The example uses a value like `https://sharbi-api.buildverse.app#http://localhost:3001`, which injects a fragment (`#..`) that browsers strip before the request. Misconfiguration could cause API calls to silently fail or point to unintended hosts.

**Remediation:** Define one valid URL per environment, like `NEXT_PUBLIC_API_URL=https://api.myapp.com`.

Use separate dotfiles (or CI secrets) for production vs. local development.

**Fix:** .env.example: `NEXT_PUBLIC_API_URL=https://sharbi-api.buildverse.app` (no fragment).



#### **1. No Client-Side File Size Enforcement on Upload [Fixed ✓]**

**Location:** `src/components/atoms/input-image.tsx`

**Issue:** The file input restricts MIME types but does not enforce the advertised “Less than 3 MB” guideline; users can select larger files.

**Remediation:** In `handleImageChange`, add a check if `(file.size > 3 * 1024 * 1024)` throw new `Error('File too large');`.

**Fix:** `src/components/atoms/input-image.tsx` now checks `file.size > 3 * 1024 * 1024` and errors.

#### **2. Console Logging of Errors in Production [Acknowledge]**

**Location:** Multiple client components (for example `file-uploader.ts`, `login.ts`)

**Issue:** Use of `console.error` exposes stack traces or internal messages to users’ developer consoles, which can leak implementation details.

**Remediation:** Replace with a production-aware logger that strips or redacts sensitive details. Disable console logs in production builds via a Babel plugin or runtime flag.



No informational issues were found.



# Technical Findings Summary

## Findings

Vulnerability Level	Total	Pending	Not Apply	Acknowledged	Partially Fixed	Fixed
<div><div></div>High</div>	4					4
<div><div></div>Medium</div>	4			1		3
<div><div></div>Low</div>	2			1		1
<div><div></div>Informational</div>	0					

# Assessment Results

## Score Results

Review	Score
Global Score	85/100
Assure KYC	Not completed
Audit Score	85/100

The Following Score System Has been Added to this page to help understand the value of the audit, the maximum score is 100, however to attain that value the project must pass and provide all the data needed for the assessment. Our Passing Score has been changed to 84 Points for a higher standard, if a project does not attain 85% is an automatic failure. Read our notes and final assessment below. The Global Score is a combination of the evaluations obtained between having or not having KYC and the type of contract audited together with its manual audit.

## Audit PASS

The SHARBI FUN APP presently fails to mitigate several critical vulnerabilities. Until these high-risk issues are fully addressed, the SHARBI FUN APP cannot be considered secure for production use.

After the fixes, we inform you that the project has met the necessary security standards.

# Disclaimer

Assure Defi has conducted an independent security assessment to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the reviewed code for the scope of this assessment. This report does not constitute agreement, acceptance, or advocating for the Project, and users relying on this report should not consider this as having any merit for financial AIToken in any shape, form, or nature. The contracts audited do not account for any economic developments that the Project in question may pursue, and the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude, and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are entirely free of exploits, bugs, vulnerabilities or deprecation of technologies. All information provided in this report does not constitute financial or investment in AIToken, nor should it be used to signal that any person reading this report should invest their funds without sufficient individual due diligence, regardless of the findings presented. Information is provided 'as is, and Assure Defi is under no covenant to audit completeness, accuracy, or solidity of the contracts. In no event will Assure Defi or its partners, employees, agents, or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions or actions with regards to the information provided in this audit report. The assessment of AITokens provided by Assure Defi are subject to dependencies and are under continuing development. You agree that your access or use, including but not limited to any AITokens, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies with high levels of technical risk and uncertainty. The assessment reports could include false positives, negatives, and unpredictable results. The AIToken may access, and depend upon, multiple layers of third parties.

