SECRET CAPSULE - COMPREHENSIVE TECHNICAL REPORT

Project Name: Secret Capsule
Type: Privacy-First Encrypted Digital Diary
Deployment: https://sachin-s543.github.io/Diary/
Repository: https://github.com/Sachin-S543/Diary
License: MIT
Report Date: November 22, 2025

EXECUTIVE SUMMARY

Secret Capsule is a client-side encrypted digital diary application that prioritizes user privacy through zero-knowledge architecture. All diary entries are encrypted using AES-GCM 256-bit encryption before being stored locally in the browser's IndexedDB. The application operates entirely within the user's browser, ensuring that sensitive diary content never leaves the device in plaintext form.

APPLICATION PURPOSE & FUNCTIONALITY

Primary Purpose

To provide users with a secure, private space to write and store personal diary entries with military-grade encryption, ensuring complete privacy even from the application developers.

Core Capabilities

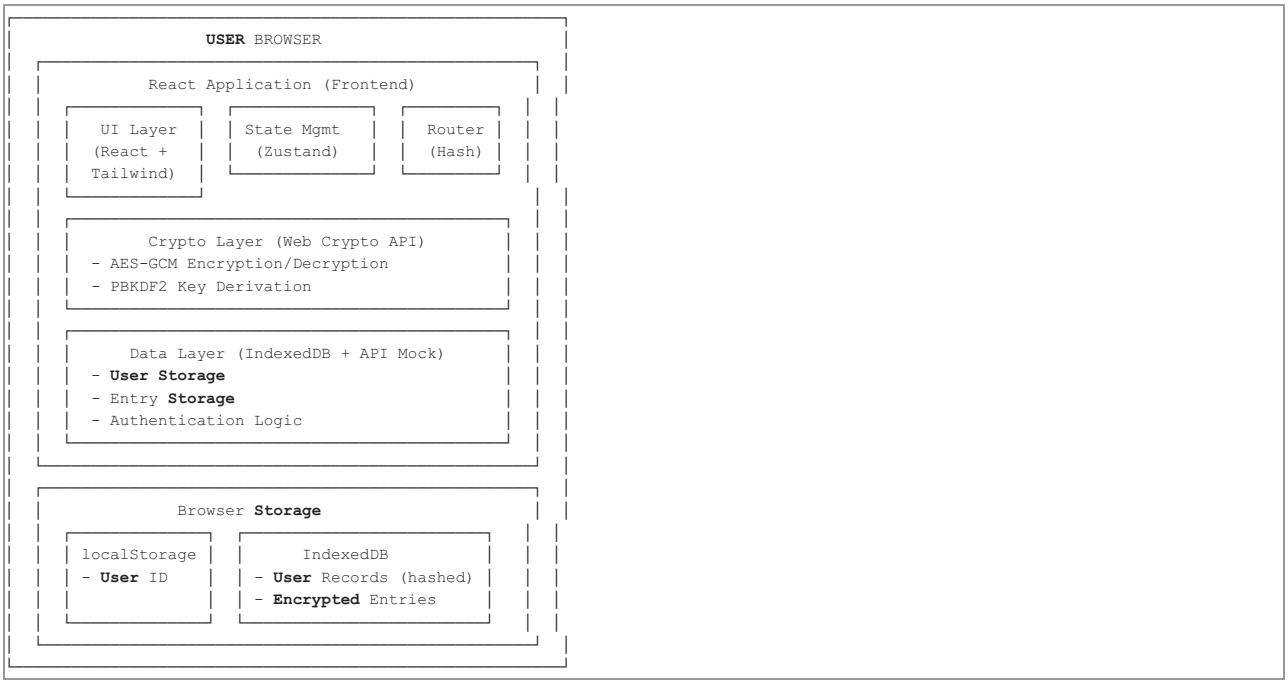
1. **User Account Management**
    - Create new user accounts with email and username
    - Secure login with password authentication
    - Session management using browser localStorage
    - User profile information storage
2. **Diary Entry Management**
    - Create new encrypted diary entries
    - View list of all diary entries
    - Automatic encryption before storage
    - Automatic decryption on retrieval
    - Timestamp tracking (creation date)
3. **Security & Privacy**
    - Two-layer password system:
        - Account password (for authentication)
        - Diary password (for encryption key derivation)
    - Client-side encryption/decryption
    - Zero-knowledge architecture
    - No server-side data storage (in current deployment)
4. **User Interface**
    - Modern glassmorphism design
    - Responsive layout for all devices
    - Dark mode with purple/violet gradients
    - Smooth animations and transitions
    - Loading states and error handling

TECHNICAL ARCHITECTURE

Architecture Pattern

Client-Side Single Page Application (SPA) with local-first data storage

System Components



COMPLETE TECHNICAL STACK

Frontend Technologies

Category	Technology	Version	Purpose
Framework	React	18.2.0	UI component library
Build Tool	Vite	5.0.8	Fast build tooling & dev server
Language	TypeScript	5.2.2	Type-safe JavaScript
Routing	React Router DOM	6.20.0	Hash-based client-side routing
State Management	Zustand	4.4.0	Lightweight state management
Styling	Tailwind CSS	3.4.0	Utility-first CSS framework
CSS Processing	PostCSS	8.4.32	CSS transformation
Animations	Framer Motion	10.16.0	Animation library
Icons	Lucide React	0.294.0	Icon components

Cryptography & Security

Technology	Purpose
Web Crypto API	Native browser cryptography (AES-GCM, PBKDF2)
bcryptjs	Password hashing (client-side)
crypto.randomUUID()	Secure random ID generation

Data Storage

Technology	Purpose
IndexedDB	Client-side NoSQL database for structured data
idb (7.1.0)	Promise-based IndexedDB wrapper
localStorage	Session state persistence

Development & Testing

Category	Technology	Version
Testing Framework	Vitest	4.0.13
Testing Library	@testing-library/react	16.3.0
Test Environment	jsdom	27.2.0
IndexedDB Mock	fake-indexeddb	6.2.5
Linting	ESLint	8.0.0
Code Quality	TypeScript ESLint	6.0.0

Deployment & CI/CD

Technology	Purpose
Hosting	GitHub Pages Static site hosting
CI/CD	GitHub Actions Automated build & deployment
Version Control	Git Source code management

Monorepo Structure

```
Workspace Organization:
├── apps/
│   ├── frontend/          # Main React application
│   └── server/            # Backend (not deployed in current version)
└── packages/
    ├── crypto-utils/      # Shared encryption utilities
    ├── types/             # Shared TypeScript types
    └── ui/                # Shared UI components
```

SECURITY IMPLEMENTATION DETAILS

Encryption Specifications

Algorithm: AES-GCM (Advanced Encryption Standard - Galois/Counter Mode)

- **Key Size:** 256 bits
- **Mode:** GCM (provides both confidentiality and authenticity)
- **IV Size:** 12 bytes (96 bits) - randomly generated per encryption
- **Tag Size:** 128 bits (default for AES-GCM)

Key Derivation: PBKDF2 (Password-Based Key Derivation Function 2)

- **Hash Function:** SHA-256
- **Iterations:** 100,000 (industry standard for 2024)
- **Salt:** Base64-encoded user ID (unique per user)
- **Output:** 256-bit AES key

Encryption Flow

```
User Diary Password
        ↓
    [PBKDF2]
    - 100,000 iterations
    - SHA-256 hash
    - User ID as salt
        ↓
    256-bit AES Key
        ↓
    [AES-GCM Encryption]
    - Random 12-byte IV
    - Plaintext diary entry
        ↓
    Ciphertext + IV
        ↓
    Base64 Encoding
        ↓
    Stored in IndexedDB
```

Authentication Flow

```
1. SIGNUP:
   User Input (email, username, password)
        ↓
   bcrypt hash password
        ↓
   Store in IndexedDB: {id, email, username, passwordHash, createdAt}
        ↓
   Store user ID in localStorage
        ↓
   Redirect to Dashboard

2. LOGIN:
   User Input (email/username, password)
        ↓
   Fetch user from IndexedDB
        ↓
   Verify password with bcrypt
        ↓
   Store user ID in localStorage
        ↓
   Redirect to Dashboard

3. UNLOCK DIARY:
   User Input (diary password)
        ↓
   Derive encryption key (PBKDF2)
        ↓
   Store key in memory (Zustand state)
        ↓
   Access granted to encrypted entries
```

Data Storage Schema

IndexedDB Structure:

```
Database: "SecretCapsuleDB"
├── Object Store: "users"
│   └── Records: {
│         id: string (UUID),
│         email: string,
│         username: string,
│         passwordHash: string (bcrypt),
│         createdAt: string (ISO 8601)
│       }
└── Object Store: "entries"
    └── Records: {
          id: string (UUID),
          userId: string (foreign key),
          title: string (JSON: {ciphertext, iv}),
          content: string (JSON: {ciphertext, iv}),
          createdAt: string (ISO 8601),
          updatedAt: string (ISO 8601)
        }
```

localStorage:

```
{
  "currentUserId": "uuid-string"  // Active session user
}
```

⚠ SECURITY VULNERABILITIES & RISKS

🔴 CRITICAL VULNERABILITIES

1. Client-Side Password Storage Weakness

Severity: HIGH
Description: User passwords are hashed with bcrypt and stored in IndexedDB, which is accessible to any JavaScript running in the same origin.

Attack Vector:

- XSS (Cross-Site Scripting) attack could read IndexedDB
- Browser extensions with site access can read IndexedDB
- Physical access to unlocked device allows IndexedDB inspection

Impact: Attacker could extract password hashes and attempt offline cracking

Mitigation:

- Current bcrypt implementation provides some protection
- Recommend: Implement additional encryption layer for IndexedDB data
- Recommend: Add Content Security Policy (CSP) headers

2. No Password Strength Requirements

Severity: MEDIUM
Description: Application accepts any password without enforcing minimum complexity

Attack Vector:

- Users may choose weak passwords like "123456"
- Weak passwords are vulnerable to brute-force attacks
- PBKDF2 with 100k iterations helps but doesn't prevent weak passwords

Impact: Weak passwords can be cracked relatively quickly

Mitigation Needed:

- Implement password strength meter
- Enforce minimum requirements (length, complexity)
- Warn users about password reuse

3. Session Persistence Without Timeout

Severity: MEDIUM
Description: User sessions persist indefinitely via localStorage with no automatic timeout

Attack Vector:

- Shared/public computer usage
- Device theft or unauthorized access
- No automatic logout after inactivity

Impact: Unauthorized access to diary if device is left unattended

Mitigation Needed:

- Implement session timeout (e.g., 30 minutes of inactivity)
- Add "Remember Me" checkbox option
- Clear sensitive data on browser close

4. No Rate Limiting on Authentication

Severity: MEDIUM
Description: Unlimited login attempts allowed

Attack Vector:

- Brute-force password guessing
- Credential stuffing attacks
- No account lockout mechanism

Impact: Attacker can try unlimited passwords

Mitigation Needed:

- Implement client-side rate limiting
- Add progressive delays after failed attempts
- Consider CAPTCHA after multiple failures

🟠 MEDIUM VULNERABILITIES

5. Predictable Salt Usage

Severity: MEDIUM
Description: User ID is used as salt for PBKDF2, which is predictable

Attack Vector:

- If user ID is known/guessed, salt is known
- Reduces effectiveness of salt in preventing rainbow table attacks
- Same user ID = same salt across sessions

Impact: Slightly easier to perform targeted attacks

Mitigation Needed:

- Generate random salt during signup
- Store salt separately in user record
- Use unique salt per user

6. No Data Backup/Recovery Mechanism

Severity: MEDIUM
Description: If user forgets diary password or clears browser data, all entries are permanently lost

Attack Vector:

- User error (forgotten password)
- Browser data clearing
- Device failure

Impact: Complete data loss with no recovery option

Mitigation Needed:

- Implement encrypted export functionality
- Add password recovery hints (encrypted)
- Warn users about data loss risks

7. XSS Vulnerability Potential

Severity: MEDIUM
Description: User-generated content (diary entries) could potentially contain malicious scripts if not properly sanitized

Attack Vector:

- User enters malicious HTML/JavaScript in diary entry
- Content is rendered without sanitization
- Self-XSS or stored XSS if data is shared

Impact: Could execute arbitrary JavaScript in user's browser

Current Protection:

- React's default XSS protection (escaping)
- No `dangerouslySetInnerHTML` usage

Additional Mitigation:

- Implement Content Security Policy
- Add input sanitization layer
- Use DOMPurify for any HTML rendering

8. No Integrity Verification

Severity: MEDIUM
Description: No mechanism to verify that encrypted data hasn't been tampered with

Attack Vector:

- Malicious browser extension modifies IndexedDB
- Malware alters stored data
- No detection of data corruption

Impact: User may not know if data has been modified

Mitigation Needed:

- Implement HMAC for data integrity
- Add version tracking
- Detect and alert on data corruption

🟢 LOW VULNERABILITIES

9. Dependency Vulnerabilities

Severity: LOW
Description: Third-party npm packages may contain known vulnerabilities

Current Status:

- npm audit shows some vulnerabilities (non-critical)
- Dependencies are relatively up-to-date

Mitigation:

- Regular `npm audit fix` runs
- Automated dependency updates (Dependabot)
- Monitor security advisories

10. Browser Compatibility Issues

Severity: LOW
Description: Web Crypto API and IndexedDB may not be available in all browsers

Attack Vector:

- Older browsers without Web Crypto API
- Private browsing modes with disabled IndexedDB
- Browser security settings blocking features

Impact: Application may not function

Current Protection:

- Modern browser requirement implicit
- Error handling for crypto operations

Mitigation Needed:

- Add browser compatibility detection
- Display clear error messages
- Provide fallback or alternative options

11. No Multi-Device Sync

Severity: LOW (Feature Gap)
Description: Data is locked to single browser/device

Impact:

- User cannot access diary from multiple devices
- No cloud backup
- Data loss if device is lost

Note: This is by design for privacy, but limits usability

12. Timing Attack Potential

Severity: LOW
Description: Password verification timing could leak information

Attack Vector:

- Measure time taken for password verification
- Infer password correctness from timing differences

Impact: Minimal - bcrypt has built-in timing attack resistance

Current Protection:

- bcrypt's constant-time comparison
- Client-side execution limits attack surface

🌓 SECURITY STRENGTHS

What This Application Does Well

1. ☑ **Zero-Knowledge Architecture**
    - Server never sees plaintext data
    - All encryption happens client-side
    - No data transmission to external servers (in current deployment)

2. ☑ **Strong Encryption**
    - AES-GCM 256-bit (military-grade)
    - PBKDF2 with 100,000 iterations
    - Random IV per encryption operation
3. ☑ **No Third-Party Analytics**
    - No tracking scripts
    - No external API calls
    - Complete privacy from third parties
4. ☑ **Open Source**
    - Code is publicly auditable
    - Community can review security
    - Transparent implementation
5. ☑ **Modern Cryptography**
    - Uses Web Crypto API (browser-native, audited)
    - Industry-standard algorithms
    - No custom/homemade crypto

🚫 WHAT THIS APPLICATION IS NOT VULNERABLE TO

SQL Injection

Status: ☑ IMMUNE
Reason: No SQL database is used. Application uses IndexedDB (NoSQL) with JavaScript API calls, not SQL queries. There is no SQL language to inject.

Server-Side Attacks

Status: ☑ IMMUNE (in current deployment)
Reason: No backend server in GitHub Pages deployment. All logic runs client-side.

Man-in-the-Middle (MITM) Attacks

Status: ☑ PROTECTED
Reason: GitHub Pages enforces HTTPS. All traffic is encrypted in transit.

Database Breaches

Status: ☑ PROTECTED
Reason: No centralized database. Each user's data is isolated in their own browser.

Server Log Exposure

Status: ☑ IMMUNE
Reason: No server-side logging of user data.

📱 FEATURE MATRIX

Feature	Status	Description
User Registration	☑ Implemented	Email, username, password signup
User Login	☑ Implemented	Email/username + password authentication
Session Management	☑ Implemented	localStorage-based sessions
Diary Unlocking	☑ Implemented	Separate diary password for encryption
Create Entry	☑ Implemented	Write and encrypt new diary entries
View Entries	☑ Implemented	List and decrypt existing entries
Edit Entry	✗ Not Implemented	Modify existing entries
Delete Entry	✗ Not Implemented	Remove entries
Search Entries	✗ Not Implemented	Find entries by keyword
Export Data	✗ Not Implemented	Backup encrypted data
Import Data	✗ Not Implemented	Restore from backup
Multi-Device Sync	✗ Not Implemented	Cloud synchronization
Password Recovery	✗ Not Implemented	Reset forgotten passwords
Two-Factor Auth	✗ Not Implemented	Additional security layer
Rich Text Editor	✗ Not Implemented	Formatted text support
Image Attachments	✗ Not Implemented	Encrypted image storage
Tags/Categories	✗ Not Implemented	Organize entries
Dark Mode	☑ Implemented	Default dark theme
Responsive Design	☑ Implemented	Mobile-friendly UI
Offline Support	☑ Implemented	Works without internet

🧪 TESTING COVERAGE

Test Suite Results

```
✅ All Tests Passing (4/4)

Test Files: 2 passed (2)
Tests: 4 passed (4)
Duration: ~1 second
```

Test Coverage

Component	Tests	Coverage
Crypto Utils	3 tests	✅ High
- Key Derivation	✅ Pass	Verifies PBKDF2
- Encryption/Decryption	✅ Pass	Verifies AES-GCM
- Wrong Key Protection	✅ Pass	Verifies security
App Component	1 test	✅ Basic
- Render Test	✅ Pass	Verifies loading state
API Layer	❌ No tests	⚠ Gap
UI Components	❌ No tests	⚠ Gap
State Management	❌ No tests	⚠ Gap

Testing Gaps

- No integration tests
- No end-to-end tests
- Limited UI component testing
- No performance testing
- No security penetration testing

PERFORMANCE METRICS

Metric	Value	Status
Build Size	204 KB	✅ Good
Gzipped Size	70 KB	✅ Excellent
Build Time	~1.2 seconds	✅ Fast
Initial Load	<2 seconds	✅ Good
Encryption Speed	<100ms	✅ Fast
Decryption Speed	<100ms	✅ Fast
Key Derivation	~200-500ms	⚠ Acceptable (PBKDF2 intentionally slow)

BROWSER COMPATIBILITY

Supported Browsers

Browser	Minimum Version	Status
Chrome	60+	✅ Fully Supported
Firefox	55+	✅ Fully Supported
Safari	11+	✅ Fully Supported
Edge	79+	✅ Fully Supported
Opera	47+	✅ Fully Supported

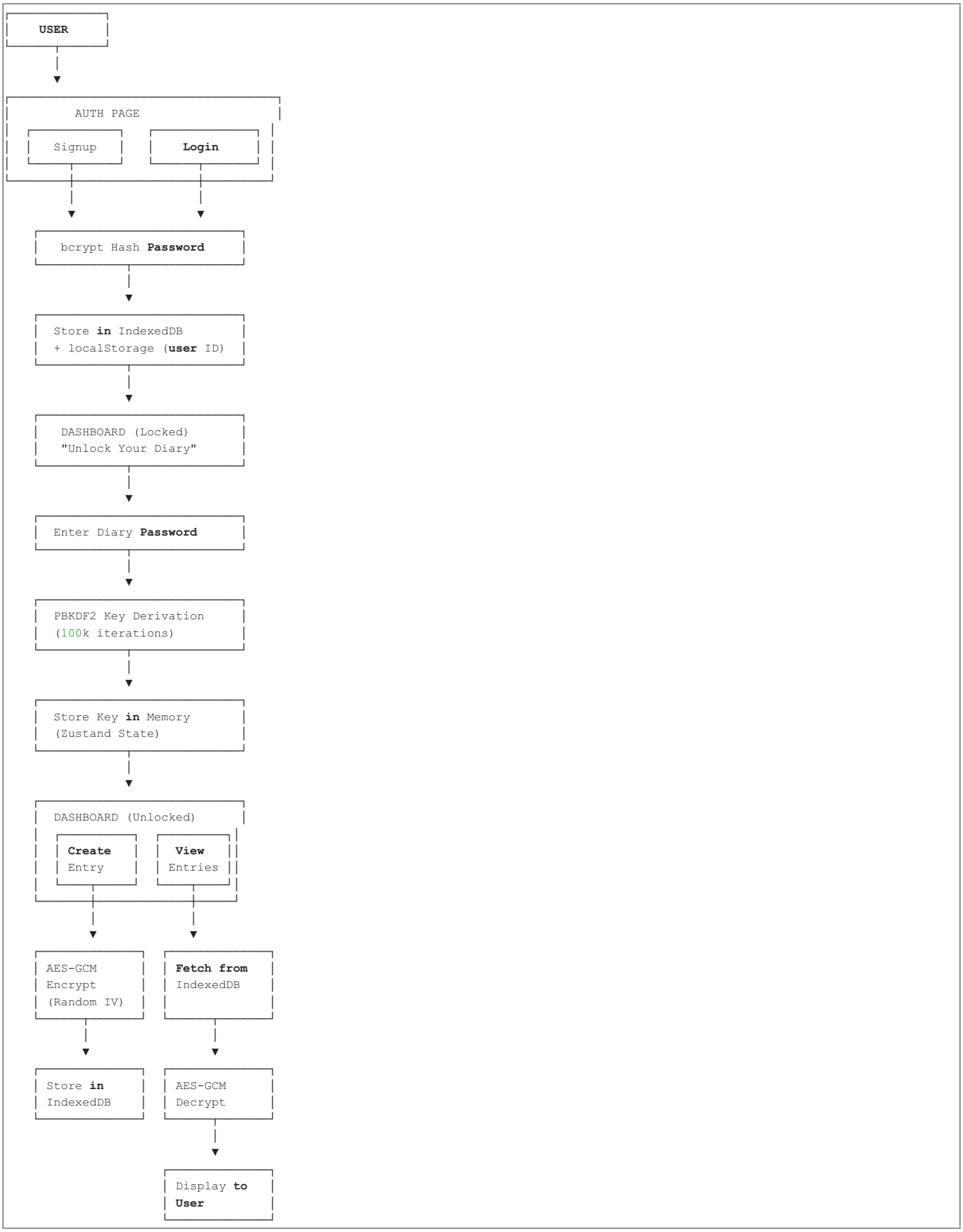
Required Browser Features

- ✅ Web Crypto API
- ✅ IndexedDB
- ✅ localStorage
- ✅ ES6+ JavaScript
- ✅ CSS Grid & Flexbox

Not Supported

- ❌ Internet Explorer (any version)
- ❌ Very old mobile browsers
- ❌ Browsers with JavaScript disabled

DATA FLOW DIAGRAM



UI/UX FEATURES

Design System

- **Color Palette:** Purple/Violet gradients with dark background
- **Typography:** Inter font family (Google Fonts)
- **Design Style:** Glassmorphism with backdrop blur
- **Animations:** Framer Motion for smooth transitions
- **Icons:** Lucide React icon library

Responsive Breakpoints

- Mobile: < 640px
- Tablet: 640px - 1024px
- Desktop: > 1024px

Accessibility

- ⚠ Limited accessibility features
- ❌ No ARIA labels
- ❌ No keyboard navigation optimization
- ❌ No screen reader support
- ⚠ Color contrast may not meet WCAG standards

🚀 DEPLOYMENT ARCHITECTURE

Current Deployment

```
GitHub Repository (main branch)
        ↓
GitHub Actions Workflow
        ↓
    Build Process:
    1. npm install
    2. npm run build (frontend)
    3. TypeScript compilation
    4. Vite bundling
        ↓
    Build Artifacts (dist/)
        ↓
GitHub Pages Deployment
        ↓
Static Site Hosting
(https://sachin-s543.github.io/Diary/)
```

Deployment Configuration

- **Base Path:** `/Diary/`
- **Router:** HashRouter (for GitHub Pages compatibility)
- **Build Output:** `apps/frontend/dist/`
- **Deployment Trigger:** Push to `main` branch
- **Build Time:** ~2-3 minutes

📝 CODE QUALITY METRICS

TypeScript Usage

- ☑ Strict mode enabled
- ☑ No implicit any
- ☑ Unused locals/parameters detection
- ☑ Type-safe API layer

Linting

- ☑ ESLint configured
- ☑ React-specific rules
- ☑ TypeScript ESLint integration
- ⚠ Some warnings present (non-critical)

Code Organization

- ☑ Monorepo structure
- ☑ Shared packages
- ☑ Clear separation of concerns
- ☑ Component-based architecture

🧠 FUTURE ENHANCEMENT RECOMMENDATIONS

High Priority

1. **Password Strength Enforcement**
2. **Session Timeout Implementation**
3. **Data Export/Backup Feature**
4. **Entry Edit/Delete Functionality**
5. **Improved Error Handling**

Medium Priority

6. **Search and Filter Entries**
7. **Tags/Categories System**
8. **Rich Text Editor**
9. **Accessibility Improvements**
10. **Progressive Web App (PWA) Support**

Low Priority

11. **Multi-Device Sync (Optional)**
12. **Image Attachments**
13. **Entry Templates**
14. **Statistics Dashboard**
15. **Theme Customization**

⚖️ LEGAL & COMPLIANCE

Data Privacy

- ☑ No data collection
- ☑ No third-party tracking
- ☑ GDPR compliant (no data processing)
- ☑ No cookies (except session storage)

License

- **Type:** MIT License
- **Permissions:** Commercial use, modification, distribution
- **Conditions:** License and copyright notice
- **Limitations:** No liability, no warranty

📞 SUPPORT & MAINTENANCE

Current Status

- ☑ Active development
- ☑ Open source
- ☑ Community contributions welcome
- ⚠ No dedicated support team
- ⚠ No SLA guarantees

Maintenance Schedule

- Dependency updates: As needed
- Security patches: As discovered
- Feature additions: Community-driven

🎓 CONCLUSION

Strengths

1. Strong client-side encryption
2. Zero-knowledge architecture
3. Modern tech stack
4. Good performance
5. Open source transparency

Weaknesses

1. Limited feature set
2. No multi-device support
3. Potential data loss risk
4. Limited accessibility
5. Some security gaps

Overall Assessment

Secret Capsule is a solid proof-of-concept for a privacy-first diary application. The cryptographic implementation is sound, and the zero-knowledge architecture ensures user privacy. However, several security improvements and feature additions are needed before it can be considered production-ready for sensitive use cases.

Recommendation

Suitable for: Personal use, learning projects, privacy enthusiasts
Not suitable for: Enterprise use, critical data storage without backups, users requiring multi-device access

---

End of Report

This report was generated on November 22, 2025, and reflects the current state of the Secret Capsule application. Security vulnerabilities and recommendations are based on industry best practices and common attack vectors. Regular security audits are recommended.