PassVault - Technical Report

Emmanuele Piola 2039978 - Francesco Macri 2055851

31 May 2025

Executive Summary

PassVault is a comprehensive password management web application designed to securely store, organize, and manage user passwords. The application provides a full-stack solution with a modern React-based frontend, robust Node.js backend, and PostgreSQL database. The system emphasizes security, usability, and collaboration through features like password health monitoring, folder organization, and secure sharing capabilities.

- **Technology Stack**: Next.js 15, Node.js, PostgreSQL
- **Security Features**: AES encryption, password health assessment, breach detection
- **Collaboration**: Shared folders with multi-user access
- **Deployment**: Containerized with Docker, deployed on Render
- **Authentication**: Email/password + Google OAuth integration

1 Project Overview

1.1 Purpose and Scope

PassVault addresses the critical need for secure password management in today's digital landscape. The application allows users to:

- Store and manage passwords securely with AES encryption
- Organize passwords into customizable folders
- Share password collections through secure folder sharing
- Monitor password health and security ratings
- Generate cryptographically secure passwords
- Access passwords from any device with a modern web browser

1.2 Target Users

- **Individual Users**: Personal password management
- **Small Teams**: Collaborative password sharing
- **Security-Conscious Users**: Advanced security features and monitoring

1.3 Key Features

- **Secure Storage**: AES-256 encryption for all sensitive data
- **Password Health**: Automated security scoring and breach detection
- **Organization**: Hierarchical folder structure for password categorization
- **Sharing**: Secure folder sharing with granular permissions
- **Cross-Platform**: Responsive design for desktop and mobile
- **Authentication**: Multiple login methods including Google OAuth

2 Technical Architecture

2.1 System Architecture Overview

```
+-----------------+    +-----------------+    +-----------------+
|   Frontend      |    |   Backend       |    |   Database      |
|   (Next.js)     |<-->|   (Node.js)     |<-->|   (PostgreSQL)  |
|   Port 3000     |    |   Port 8000     |    |   Port 5432     |
+-----------------+    +-----------------+    +-----------------+
```

2.2 Technology Stack

Frontend (Next.js Application):

- Framework: Next.js 15.3.2 with App Router
- UI Library: React 18.2.0 with TypeScript
- Styling: Tailwind CSS 4.0 with custom components
- Authentication: Google OAuth 2.0 integration
- State Management: React Context API
- Icons: Material Symbols and Lucide React
- Build Tool: Turbopack for development

Backend (Node.js Server):

- Runtime: Node.js with Express.js
- Database Driver: PostgreSQL client (pg)
- Security: AES encryption
- Email: Nodemailer
- Authentication: Google Auth Library
- Validation: Custom password scoring
- API: RESTful with CORS support

Database (PostgreSQL):

- ACID-compliant database
- Tables: Users, Folders, Passwords, Reset Tokens, Folder Sharing
- Constraints: Foreign keys, triggers, and validation
- Security: Encrypted data and isolation

3 Focus

3.1 Backend (Node.js/Express)

The backend is built with Node.js and Express, providing a full RESTful API for secure password management. The main features include:

Authentication and Security:

- Full authentication system with signup/login
- User session management
- Password encryption using secure algorithms (implemented in `encryption.js`)

Password Management:

- Secure password generation with customizable parameters (length, uppercase, lowercase, numbers, symbols)
- Password security validation (`passwordValidation.js`) including:
  - Repetitive pattern detection
  - Breach check via HaveIBeenPwned service
  - Complexity analysis

Data Management:

- PostgreSQL integration for secure storage
- API for folder and item management (CRUD operations)
- Folder sharing system across users
- Email service for password recovery (`emailService.js`)

3.2 Frontend (Next.js/React)

The frontend is developed using Next.js and React, offering a modern and responsive interface. The main features include:

Architecture and State:

- Centralized state management through Context API (`context.tsx`)
- Client-side routing for seamless navigation
- Integrated authentication with the backend

User Features:

- Interactive dashboard for password management
- Folder system to organize passwords
- Interface for secure folder sharing
- Visual display of password security levels
- Search and filter system for stored passwords

UI:

- Modern design with smooth animations (as shown on the homepage)
- Responsive interface
- Instant visual feedback for user actions
- Interactive modals and forms for password handling
- Visual indicators for password security levels

The application is designed with a strong focus on security and user experience, following best practices on both frontend and backend to ensure secure and convenient password management for users.

3.3 From Database to Frontend through the Server

The Node.js server handles communication with the PostgreSQL database using the pg module (node-postgres), leveraging a connection pool configured in db.js. When the frontend requests data (e.g., the list of passwords), the server performs the following operations:

1. Receives the HTTP request with the user ID
2. Verifies the user's authentication
3. Executes the database query using the connection pool
4. Retrieves sensitive data (such as passwords) in encrypted format
5. Decrypts the data using functions defined in `encryption.js`
6. Structures the data into a proper JSON format, enriching it with additional information such as password security levels and sharing details
7. Sends the response back to the frontend with the decrypted and formatted data

This process occurs securely through dedicated REST endpoints (such as `/api/items` and `/api/folders`), maintaining a clear separation between database-level data and presentation logic.

3.4 From Server to Frontend Rendering

The frontend performs the following operations to retrieve and display data:

1. Sends API requests to the backend via `fetch` using functions like `getItems()` and `getFolders()`
2. Stores the received data in the React Context (`context.tsx`) to make it globally accessible
3. Accesses the stored data in pages like `dashboard/page.tsx` and components like `ItemBox.tsx` via the `useSelection()` hook
4. Applies filtering and sorting logic inside `ItemBox.tsx`, such as search term filtering or security level sorting
5. Passes the filtered data to individual `Item` components for rendering
6. Instantly reflects any user-driven changes (e.g., new password creation or updates) in the UI thanks to React's reactivity

This architecture enables real-time synchronization between user actions and the visual interface, ensuring a seamless and responsive user experience.

3.5 Deployment Architecture

Containerization (Docker):

- **Database (PostgreSQL)**: Port 5434 → 5432
- **Backend (Node.js)**: Port 8000
- **Frontend (Next.js)**: Port 3000

The entire application is deployable using Render, which orchestrates containerized services and provides managed hosting for the frontend, backend, and database layers.