**SupaSafe Requirements Document**

Version 1.2

March 25, 2025

**Table of Contents**

[**1.** **Introduction** 3](#_Toc193824686)

[**2.** **Functional Requirements** 4](#_Toc193824687)

[**2.1** **User Authentication** 4](#_Toc193824688)

[**2.2** **Password Management** 4](#_Toc193824689)

[**2.3** **User Interface** 5](#_Toc193824690)

[**3.** **Non-Functional Requirements** 6](#_Toc193824691)

[**3.1** **Security** 6](#_Toc193824692)

[**3.2** **Performance** 7](#_Toc193824693)

[**3.3** **Scalability** 7](#_Toc193824694)

[**3.4** **Usability** 7](#_Toc193824695)

[**3.5** **Reliability** 7](#_Toc193824696)

[**3.6** **Maintainability** 8](#_Toc193824697)

[**3.7** **Constraints** 8](#_Toc193824698)

[**4.** **Database Requirements** 9](#_Toc193824699)

[**4.1** **Users Table** 9](#_Toc193824700)

[**4.2** **Passwords Table** 10](#_Toc193824701)

[**5.** **Assumptions** 11](#_Toc193824702)

[**6.** **Deliverables** 11](#_Toc193824703)

# **Introduction**

The SupaSafe application will be a password manager designed to securely store, retrieve, and manage user passwords for various sites and services. SupaSafe will provide a user-friendly solution for individuals seeking to protect their digital credentials while ensuring ease of access through a streamlined interface. The application will leverage a master password to derive a Key Encryption Key (KEK) and a randomly generated Data Encryption Key (DEK) for robust cryptographic security, ensuring that user data remains protected even in the event of a data breach.

SupaSafe will be built with a modern tech stack to ensure reliability and performance. The frontend will be developed using React.js, providing a responsive and intuitive user interface. The backend will be powered by Node.js with Express, handling secure API interactions and cryptographic operations. Supabase will serve as the database, offering a scalable and managed PostgreSQL solution with built-in security features like HTTPS. This combination of technologies will enable SupaSafe to deliver a secure, efficient, and maintainable password management experience.

The purpose of this document is to outline the functional and non-functional requirements for SupaSafe, detailing the features, security measures, performance expectations, and constraints that will guide its development. This ensures that the application will meet user needs while adhering to best practices in security and usability.

# **Functional Requirements**

Functional requirements specify the features and capabilities SupaSafe must provide to users.

## **User Authentication**

1. **Signup**

* SupaSafe shall allow a user to create an account by providing a unique email and a master password.
* The master password shall be hashed using bcrypt and stored with a randomly generated salt in the users table in Supabase.
* A random Data Encryption Key (DEK) shall be generated for the user and encrypted with a Key Encryption Key (KEK) derived from the plaintext master password.
* Upon successful signup, the user shall be redirected to the login page.

1. **Login**

* SupaSafe shall allow a user to log in by entering their email and master password.
* The backend shall verify the master password against the stored bcrypt hash in Supabase.
* If valid, the backend shall derive a KEK using PBKDF2 from the plaintext master password and salt (per NFR2), encrypt the KEK with a server-side secret, and embed it in a JWT.
* The frontend shall receive and store the JWT in localStorage.

1. **Logout**

* SupaSafe shall allow a logged-in user to log out.
* Upon logout, the JWT shall be cleared from localStorage, rendering the KEK inaccessible until the next login.

## **Password Management**

1. **Add Password**

* SupaSafe shall allow a logged-in user to add a password by entering a site\_name (e.g., 'Gmail'), a username, and either manually entering a password (e.g., 'myp@ss123') in the password field or using a 'Generate Password' button.
* The frontend shall display a slider (8-32 characters, default 16). If 'Generate Password' is clicked, it shall populate the password field with a random password of the slider’s length, containing uppercase, lowercase, numbers, and special characters (e.g., !@#$%^&\*()), which the user can then edit.
* The frontend shall send the JWT to the backend, which shall decrypt the KEK from the JWT, decrypt the DEK, encrypt the password (manual or generated, as edited) with the DEK (AES-256-CBC) and a random IV, and store the encrypted password, IV, site\_name, and username in the passwords table in Supabase.

1. **Retrieve Passwords**

* SupaSafe shall allow a logged-in user to view all their saved passwords.
* The frontend shall send the JWT to the backend.
* The backend shall decrypt the KEK from the JWT, decrypt the DEK, fetch all encrypted passwords for the user from the passwords table in Supabase, decrypt them with the DEK, and return the plaintext site\_name, username and password triples to the frontend.
* The frontend shall display the retrieved passwords in a list.

1. **Change Master Password**

* SupaSafe shall allow a logged-in user to change their master password by entering their current master password and a new master password.
* The backend shall verify the current master password against the stored hash in Supabase.
* If valid, it shall derive the old KEK, decrypt the DEK, derive a new KEK from the new master password and a new salt, re-encrypt the DEK with the new KEK, update the users table in Supabase with the new hash, salt, encrypted DEK, and IV, and issue a new JWT with the encrypted new KEK.

## **User Interface**

1. **Signup Page**

* SupaSafe shall provide a form for users to enter their email and master password to create an account.

1. **Login Page**

* SupaSafe shall provide a form for users to enter their email and master password to log in.

1. **Password Management Interface**

* SupaSafe shall provide a form to input site\_name, username, and password for adding passwords, including a 'Generate Password' button and a slider to select password length (8-32 characters, default 16) always visible. When 'Generate Password' is clicked, the password field shall be populated with a generated password based on the slider’s value, which the user can edit, and a list to display saved passwords.

1. **Change Password Page**

* SupaSafe shall provide a form for users to enter their current and new master passwords.

# **Non-Functional Requirements**

Non-functional requirements define the quality attributes and constraints of SupaSafe.

## **Security**

1. **Password Hashing**

* The master password shall be hashed using bcrypt with a work factor of at least 12 to ensure resistance to brute-force attacks.

1. **Key Derivation**

* The KEK shall be derived from the master password and a salt using PBKDF2 with 100,000 iterations, a 32-byte output, and SHA-256 to ensure a strong key.

1. **Encryption**

* The DEK shall be a randomly generated 32-byte key encrypted with the KEK using AES-256-CBC with a unique 16-byte IV per encryption, stored as VARCHAR(44) (base64-encoded 32 bytes) with the IV as VARCHAR(24) (base64-encoded 16 bytes) in the users table under EncryptedDEK and DEKIV.
* Stored passwords shall be encrypted with the DEK using AES-256-CBC with a unique 16-byte IV per password, stored as VARCHAR(44) (base64-encoded 32 bytes) under EncryptedPass with the IV as VARCHAR(24) (base64-encoded 16 bytes) in the passwords table (assuming a max plaintext password length of 32 characters, padded to the nearest 16-byte block).
* The UserID in the passwords table shall be a GUID, referencing the ID GUID in the users table.

1. **Transport Security**

* All communication between frontend, backend, and Supabase shall use HTTPS to protect data in transit.

1. **Key Protection**

* The KEK shall be encrypted with a server-side secret (KEY\_ENCRYPTION\_SECRET) before embedding in the JWT.
* The plaintext master password shall not be stored or logged at any point.

1. **Data Loss on Master Password Forget**

* If a user forgets their master password, their stored passwords shall be unrecoverable without administrative intervention.

1. **Password Generation**

* The "Generate Password" feature shall use cryptographically secure random number generation (e.g., crypto.getRandomValues in JavaScript) to ensure unpredictable passwords.

## **Performance**

1. **Response Time**

* API responses (e.g., login, add password, retrieve passwords) shall complete within 2 seconds under normal load (single user).

1. **Key Derivation Latency**

* KEK derivation via PBKDF2 shall not exceed 500ms to balance security and user experience.

1. **Encryption/Decryption Speed**

* AES-256 encryption/decryption of passwords shall complete within 100ms per operation.

## **Scalability**

1. **User Capacity**

* SupaSafe shall support up to 100 concurrent users within Supabase’s free tier (500 MB storage).

1. **Password Storage**

* The system shall handle up to 100 stored passwords per user without significant performance degradation.

## **Usability**

1. **Interface Simplicity**

* The UI shall be intuitive, requiring no more than 3 clicks to perform any core action (e.g., add password, view passwords).

1. **Feedback**

* SupaSafe shall provide clear success/error messages for all actions (e.g., "Password saved", "Invalid credentials").

## **Reliability**

1. **Uptime**

* The deployed application shall achieve 95% uptime, assuming Supabase and hosting provider stability.

1. **Data Integrity**

* Encrypted passwords and DEKs shall remain consistent across operations—no corruption due to encryption/decryption errors.

## **Maintainability**

1. **Code Documentation**

* All critical functions (e.g., key derivation, encryption) shall include inline comments, and a README shall detail setup/deployment with Supabase.

1. **Modularity**

* The codebase shall separate frontend, backend, and security logic for easy updates.

## **Constraints**

1. **Tech Stack**

* SupaSafe shall use React.js (frontend), Node.js + Express (backend), and Supabase (database).

# **Database Requirements**

## **Users Table**

The users table in Supabase shall store user authentication and encryption data as follows:

1. CREATE TABLE Users (

2. ID UUID PRIMARY KEY DEFAULT gen\_random\_uuid(), -- GUID primary key (FR1)

3. Email VARCHAR(255) UNIQUE NOT NULL,

4. HashedPass VARCHAR(60) NOT NULL, -- Bcrypt output for master password hashing (NFR1)

5. Salt VARCHAR(24) NOT NULL, -- Base64-encoded 16-byte salt for PBKDF2 KEK derivation (NFR2)

6. LastUpdate TIMESTAMP DEFAULT NOW(),

7. CreationDate TIMESTAMP DEFAULT NOW(),

8. EncryptedDEK VARCHAR(44) NOT NULL, -- Base64-encoded 32-byte encrypted DEK (NFR3)

9. DEKIV VARCHAR(24) NOT NULL, -- Base64-encoded 16-byte IV for DEK encryption (NFR3)

10. KEYCreationDate TIMESTAMP DEFAULT NOW()

11. );

|  |  |  |
| --- | --- | --- |
| **Field** | **Definition** | **Constraints/Comments** |
| ID | UUID | PRIMARY KEY DEFAULT gen\_random\_uuid() -- GUID primary key (FR1) |
| Email | VARCHAR(255) | UNIQUE NOT NULL |
| HashedPass | VARCHAR(60) | NOT NULL – Bcrypt output for master password hashing (NFR1) |
| Salt | VARCHAR(24) | NOT NULL – Base64-encoded 16-byte salt for PBKDF2 KEK derivation (NFR2) |
| LastUpdate | TIMESTAMP | DEFAULT NOW() |
| CreationDate | TIMESTAMP | DEFAULT NOW() |
| EncryptedDEK | VARCHAR(44) | NOT NULL – Base64-encoded 32-byte encrypted DEK (NFR3) |
| DEKIV | VARCHAR(24) | NOT NULL – Base64-encoded 16-byte IV for DEK encryption (NFR3) |
| KEYCreationDate | TIMESTAMP | DEFAULT NOW() |

## **Passwords Table**

The passwords table in Supabase shall store encrypted user passwords as follows:

1. CREATE TABLE Passwords (

2. ID UUID PRIMARY KEY DEFAULT gen\_random\_uuid(), -- GUID primary key

3. UserID UUID NOT NULL, -- Foreign key referencing Users.ID (FR4)

4. IV VARCHAR(24) NOT NULL, -- Base64-encoded 16-byte IV for AES-256-CBC (NFR3)

5. Username VARCHAR(100) NOT NULL,

6. EncryptedPass VARCHAR(44) NOT NULL, -- Base64-encoded 32-byte encrypted password (NFR3)

7. SiteName VARCHAR(100) NOT NULL,

8. LastUpdate TIMESTAMP DEFAULT NOW(),

9. CreationDate TIMESTAMP DEFAULT NOW(),

10. WebsiteURL TEXT,

11. FOREIGN KEY (UserID) REFERENCES Users(ID),

12. CONSTRAINT unique\_user\_username UNIQUE (UserID, Username) -- Ensures username is unique per user

13. );

|  |  |  |
| --- | --- | --- |
| **Field** | **Definition** | **Constraints/Comments** |
| ID | UUID | PRIMARY KEY DEFAULT gen\_random\_uuid() -- GUID primary key |
| UserID | UUID | NOT NULL – Foreign key referencing Users.ID (FR4) |
| IV | VARCHAR(24) | NOT NULL – Base64-encoded 16-byte IV for AES-256-CBC (NFR3) |
| Username | VARCHAR(100) | NOT NULL |
| EncryptedPass | VARCHAR(44) | NOT NULL – Base64-encoded 32-byte encrypted password (NFR3) |
| SiteName | VARCHAR(100) | NOT NULL |
| LastUpdate | TIMESTAMP | DEFAULT NOW() |
| CreationDate | TIMESTAMP | DEFAULT NOW() |
| WebsiteURL | TEXT |  |
| FOREIGN KEY |  | (UserID) REFERENCED Users(ID) |
| CONSTRAINT |  | unique\_user\_username UNIQUE (UserID, Username) -- Ensures username is unique per user |

# **Assumptions**

* Users have access to modern web browsers (e.g., Chrome, Firefox).
* Supabase’s free tier (500 MB storage, 2 GB bandwidth/month) will suffice for hosting and storage needs.

# **Deliverables**

* Fully functional SupaSafe application deployed to Verc
* el/Heroku with Supabase integration.
* Source code hosted on GitHub with a README for setup instructions, including Supabase configuration.