

SupaSafe Requirements Document

Version 1.0

March 24, 2025

Table of Contents

1.	Introduction	3
2.	Functional Requirements.....	4
2.1	User Authentication	4
2.2	Password Management	4
2.3	User Interface	5
3.	Non-Functional Requirements	6
3.1	Security	6
3.2	Performance	6
3.3	Scalability	7
3.4	Usability	7
3.5	Reliability	7
3.6	Maintainability.....	7
3.7	Constraints	7
4.	Assumptions	8
5.	Deliverables	8

1. 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.

2. Functional Requirements

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

2.1 User Authentication

FR1 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.

FR2 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 from the plaintext master password and salt, encrypt the KEK with a server-side secret, and embed it in a JSON Web Token (JWT).
- The frontend shall receive and store the JWT in localStorage.

FR3 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.

2.2 Password Management

FR4 Add Password

- SupaSafe shall allow a logged-in user to add a password by entering a site_name (e.g., "Gmail") and a password (e.g., "myp@ss123").
- The frontend shall send the JWT with the request to the backend.
- The backend shall decrypt the KEK from the JWT, decrypt the user's DEK using the KEK, encrypt the password with the DEK (AES-256) and a random IV, and store the encrypted password, IV, and site name in the passwords table in Supabase.

FR5 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 and password pairs to the frontend.
- The frontend shall display the retrieved passwords in a list.

FR6 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.

2.3 User Interface**FR7 Signup Page**

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

FR8 Login Page

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

FR9 Password Management Interface

- SupaSafe shall provide a form to input site_name and password for adding passwords and a list to display saved passwords.

FR10 Change Password Page

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

3. Non-Functional Requirements

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

3.1 Security

NFR1 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.

NFR2 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.

NFR3 Encryption

- The DEK shall be a randomly generated 32-byte key encrypted with the KEK using AES-256-CBC with a unique IV per encryption.
- Stored passwords shall be encrypted with the DEK using AES-256-CBC with a unique IV per password.

NFR4 Transport Security

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

NFR5 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.

NFR6 Data Loss on Master Password Forget

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

3.2 Performance

NFR7 Response Time

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

NFR8 Key Derivation Latency

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

NFR9 Encryption/Decryption Speed

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

3.3 Scalability

NFR10 User Capacity

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

NFR11 Password Storage

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

3.4 Usability

NFR12 Interface Simplicity

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

NFR13 Feedback

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

3.5 Reliability

NFR14 Uptime

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

NFR15 Data Integrity

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

3.6 Maintainability

NFR16 Code Documentation

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

NFR17 Modularity

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

3.7 Constraints

NFR18 Tech Stack

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

4. 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.

5. Deliverables

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