SYSTEM ANALYSIS AND DESIGN

4.1 Overview

In this chapter, the system analysis and design approach for the IDSafe Password Manager WebApp Tool with Biometric Authentication and AES-256 Encryption will be thoroughly discussed. Section 4.2 will explain the requirements of the system in terms of its functional and non-functional requirements, and the software and hardware requirements. Section 4.3 will analyse the overall or general architecture of the IDSafe system. Section 4.4 discusses comprehensively regarding the analysis of the system which include the drawings of the context diagram, data flow diagram (DFD), entity relationship diagram (ERD), and the flowchart diagram for the system. Section 4.5 will examine the database design of the system which includes the database relational schema and data dictionary. Section 4.6 reviews the user interfaces design for the system.

4.2 System Requirements

4.2.1 Functional Requirements

Table 4.1: End-User Functional Requirements

Functional Requirement	Description
FR-EU1: User Registration via WebAuthn	The system shall allow an end-user to create an account by registering a WebAuthn credential using a platform authenticator (e.g. device biometric or screen lock).
FR-EU2: User Login via WebAuthn	The system shall allow an end-user to log in using WebAuthn by responding to a server-issued challenge with a valid signature from their registered platform authenticator.
FR-EU3: Session Management	The system shall establish a secure session for the end-user after successful WebAuthn authentication and shall terminate the session upon logout or inactivity timeout.

FR-EU4: Vault Key Generation	The system shall generate a unique 256-bit vault key (DEK) for each end-user during initial setup for encrypting and decrypting password entries.
FR-EU5: Local Vault Key Storage	The system shall store the vault key securely on the end-user's device (e.g. in IndexedDB or equivalent secure storage) so it can be used after WebAuthn login without re-entering the recovery passphrase.
FR-EU6: Recovery Passphrase Setup	The system shall allow an end-user to define a recovery passphrase during initial setup and confirm it to minimise input errors.
FR-EU7: Passphrase-based KEK Derivation	The system shall derive a Key Encryption Key (KEK) from the recovery passphrase using Argon2id with a unique salt and configured parameters.
FR-EU8: Backup Vault Key Wrapping	The system shall encrypt (wrap) the vault key using the KEK and store the resulting wrapped vault key and KDF metadata on the server for recovery purposes.
FR-EU9: Create Vault Entry	The system shall allow an end-user to create a new vault entry containing at least a title, username, password, URL, and optional notes.
FR-EU10: Encrypt Vault Entry	The system shall encrypt each vault entry on the client side using AES-256-GCM with a unique IV before sending it to the server for storage.
FR-EU11: View Vault Entries	The system shall allow an authenticated end-user to retrieve encrypted vault entries from the server, decrypt them on the client using the vault key, and view the plaintext details.
FR-EU12: Update Vault Entry	The system shall allow an end-user to update an existing vault entry and shall re-encrypt the updated data on the client side before storing it on the server.

FR-EU13: Delete Vault Entry	The system shall allow an end-user to delete an existing vault entry. The system may implement soft-delete (trash) or permanent deletion according to design.
FR-EU14: Search and Filter Entries	The system shall allow an end-user to search and/or filter their decrypted vault entries locally (e.g. by title, username, or URL) without sending plaintext search terms or results to the server.
FR-EU15: Password Generator	The system shall provide a password generator that allows the end-user to create strong passwords based on configurable criteria (length, character sets).
FR-EU16: Copy to Clipboard	The system shall allow an end-user to copy a password or username to the clipboard.

FR-EU17: Change Recovery Passphrase	The system shall allow an authenticated end-user to change their recovery passphrase, causing the system to derive a new KEK and re-wrap the vault key.
FR-EU18: Account Recovery Request	The system shall allow an end-user who has lost access to their device/WebAuthn credential to initiate an account recovery process (e.g. via email magic link).
FR-EU19: Recovery Email Verification	The system shall send a time-limited recovery link to the registered email and verify the token when the end-user accesses the link.
FR-EU20: Recovery Using Passphrase	During recovery, the system shall prompt the end-user to enter their recovery passphrase, derive the KEK using Argon2id, decrypt the wrapped vault key, and restore access to their vault on the new device.
FR-EU21: Register New WebAuthn Credential After Recovery	After successful recovery, the system shall allow the end-user to register a new WebAuthn credential on the new device and associate it with the existing account.
FR-EU22: View Account Profile	The system shall allow an authenticated end-user to view basic account information such as email, registration date, and registered devices/credentials.

FR-EU23: Manage Registered Authenticators	The system shall allow an authenticated end-user to view and optionally remove registered WebAuthn credentials (e.g. remove a lost or unused device).
FR-EU24: Logout	The system shall allow an authenticated end-user to log out explicitly, clearing any in-memory vault keys and terminating the session.

Table 4.2 shows the system administrator functional requirements of the IDSafe system.

Table 4.2: System Administrator Functional Requirements

Functional Requirement	Description
FR-SA1: Admin Authentication	The system shall provide a separate, secure login mechanism for system administrators (e.g. username/password with 2FA or admin-specific WebAuthn) to access the admin interface.
FR-SA2: View User Accounts	The system shall allow a system admin to view a list of user accounts with non-sensitive metadata (e.g. user ID, email, registration date, last login time, number of active credentials).
FR-SA3: View WebAuthn Credential Metadata	The system shall allow a system admin to view metadata about registered WebAuthn credentials (e.g. AAGUID, attestation format, registration time, last used time) without revealing any private keys.
FR-SA4: View Audit Logs	The system shall allow a system admin to view audit logs, including events such as login attempts, account recovery requests, vault operations (at metadata level), and admin actions.
FR-SA5: Search and Filter Logs	The system shall allow a system admin to search and filter audit logs by user, date range, event type, IP address, or status (success/failure).

FR-SA6: Manage User Account Status	The system shall allow a system admin to lock, unlock, or deactivate a user account (e.g. in case of suspected abuse or compromise), without accessing the user's encrypted vault contents.
FR-SA7: Reset User Sessions	The system shall allow a system admin to invalidate active sessions for a specific user, forcing re-authentication on next access.
FR-SA8: Configure Security Policies	The system shall allow a system admin to configure security-related parameters such as session timeout duration, maximum login attempts, and rate-limiting thresholds.

Table 4.2: Continued

FR-SA9: Configure KDF Parameters	The system shall allow a system admin (or security operator) to configure Argon2id parameters (time cost, memory cost, parallelism) for new users while preserving compatibility with existing users' stored parameters.
FR-SA10: Configure WebAuthn Attestation Policy	The system shall allow a system admin to configure which attestation formats and AAGUIDs are accepted (e.g. platform authenticators only) and update trusted attestation roots.
FR-SA11: Monitor System Health	The system shall provide an admin view or endpoint displaying key health and performance indicators (e.g. request counts, error rates, database status).
FR-SA12: Backup and Restore Database	The system shall provide mechanisms for a system admin to trigger or configure automated database backups and to perform restore operations in a controlled manner.
FR-SA13: Manage Application Configuration	The system shall allow a system admin to manage general application configuration values (e.g. email server settings, base URLs for magic links) through a secure interface or configuration file.
FR-SA14: View Recovery Events	The system shall allow a system admin to view a log of account recovery events, including time, source IP, and outcome, while not revealing recovery passphrases or vault contents.

FR-SA15: Export Audit Data (Compliance)	The system shall allow a system admin to export selected audit log data in a standard format (e.g. CSV) for analysis or compliance reporting.

4.2.2 Non-Functional Requirements

Non-functional requirements (NFRs) define the system's security attributes, performance, and usability attributes. They describe how the system performs its functions and are critical for ensuring the overall user experience and system effectiveness. Table 4.3 displays the security NFRs of the IDSafe system.

Table 4.3: Security NFRs

Non-Functional Requirement	Description
NFR-S1: End-to-End Encryption	The system shall ensure that all password vault entries are encrypted on the client using AES-256-GCM before transmission, and that no plaintext vault data is stored or processed on the server.
NFR-S2: Strong Authentication	The system shall enforce WebAuthn-based passwordless authentication using platform authenticators as the primary login mechanism and reject untrusted attestation formats according to the configured policy.
NFR-S3: Confidentiality of Keys	The system shall ensure that the Vault Key (DEK), KEK, and recovery passphrase never appear in plaintext on the server or in logs; all key derivation and decryption shall occur only on the client.
NFR-S4: Secure Communication	All communication between client and server shall use HTTPS with TLS 1.2 or higher, and HTTP access (if any) shall be redirected to HTTPS.
NFR-S5: Session Security	The system shall use secure, HttpOnly, SameSite cookies or equivalent mechanisms for session tokens and shall invalidate sessions upon logout or timeout.

NFR-S6: Password-Based KDF Hardening	The system shall use Argon2id with configurable time and memory cost parameters to derive the KEK from the recovery passphrase, providing resistance against offline brute-force attacks.
NFR-S7: Input Validation & XSS Protection	The system shall validate and sanitise all user input and apply Content Security Policy (CSP) and other mitigations to reduce the risk of XSS and injection attacks.
NFR-S8: Rate Limiting & Brute-Force Protection	The system shall apply rate limiting and lockout policies to login and recovery endpoints to mitigate brute-force attempts and abuse.
NFR-S9: Logging without Sensitive Data	The system shall record security-relevant events (logins, failures, recovery attempts) in audit logs without logging sensitive data such as vault contents, recovery passphrases, or cryptographic keys.
NFR-S10: Least Privilege & Access Control	The system shall separate end-user and admin privileges, ensuring that administrators cannot directly view or decrypt user vault data while still being able to perform monitoring and maintenance tasks.

Table 4.4 shows the performance NFRs of the IDSafe system.

Table 4.4: Performance NFRs

Non-Functional Requirement	Description
NFR-P1: Response Time for Core Operations	Under normal load, the system shall respond to core API operations (login, listing entries, viewing a single entry) within 2 seconds 95% of the time.
NFR-P2: WebAuthn Login Latency	The WebAuthn authentication process, excluding user interaction time, shall complete within 1 second on a mid-range device under normal network conditions.
NFR-P3: Vault Decryption Time	Decrypting and rendering 100 stored entries after login shall complete within 2 seconds on a mid-range device.

NFR-P4: Argon2id Execution Time	Argon2id parameter selection shall target an execution time between 200 ms and 500 ms on a mid-range device, balancing security and usability.
NFR-P5: Scalability	The system architecture shall support horizontal scaling of the application server so that additional instances can be added to handle increased user load without major redesign.
NFR-P6: Concurrent Users	The system shall be able to support at least X concurrent active users (to be defined based on FYP scope) without critical degradation of response times.
NFR-P7: Database Performance	Typical database read/write operations for vault metadata and logs shall complete within 200 ms under normal load.

Table 4.5 shows the usability NFRs of the IDSafe system.

Table 4.5: Usability NFRs

Non-Functional Requirement	Description
NFR-U1: Biometric-First Login Experience	The system shall allow end-users to access their vault using a single biometric or device unlock action via WebAuthn, without requiring a password or passphrase for normal logins.
NFR-U2: Simple Recovery Flow	The system shall provide a guided recovery flow that clearly explains each step and allows end-users to recover their account using an email magic link and recovery passphrase without needing technical knowledge.
NFR-U3: Consistent UI Design	The system shall use a consistent visual style (layout, colours, typography, icons) across all pages to reduce user confusion and support intuitive navigation.
NFR-U4: Responsive Layout	The user interface shall adapt to different screen sizes and orientations, ensuring that all core features are usable on both desktop and mobile browsers.
NFR-U5: Accessibility	The system shall follow basic accessibility guidelines (e.g. keyboard navigation, sufficient colour contrast, descriptive labels) to support users with common accessibility needs.

NFR-U6: Error Feedback	The system shall provide clear, human-readable error messages (e.g. for failed login, invalid passphrase, or decryption failure) without revealing sensitive technical details.
NFR-U7: Learnability	New users shall be able to register, add a password entry, and log in again successfully with minimal guidance, supported by inline hints or tooltips where necessary.
NFR-U8: Non-Intrusive Security Prompts	Security-related prompts (e.g. session timeout, recovery reminders) shall be presented in a way that does not overly disrupt user tasks while still encouraging secure behaviour.

4.2.3 Software and Hardware Requirements

Tables 4.6 and 4.7 shows the software and hardware requirements respectively for the IDSafe system.

Table 4.6: Software Requirements

Category	Requirement	Description
Operating system	Windows 10/11 or Ubuntu 22.04 LTS for development; Ubuntu 22.04 LTS (or equivalent Linux server) for deployment	Defines the OS platforms used to develop, test, and deploy the system.
Client-side storage	IndexedDB (for encrypted vault entries and encrypted vault key backup metadata)	Refers to storage mechanisms available in the user's browser.
Server-side storage	PostgreSQL 15 relational database	Specifies the database technology used on the server to store user account metadata, WebAuthn credential records, wrapped vault keys, and audit logs (all non-plaintext secrets).
Web server	Node.js 20 application server	Represents the software responsible for handling

		HTTP/HTTPS requests, routing, and API endpoints.
Frontend language	TypeScript (compiled to JavaScript ES2020)	Indicates the programming language used for client-side logic.
Backend language	TypeScript on Node.js 20	Indicates the language used for implementing backend services and APIs.
IDE/Editor	Visual Studio Code (with TypeScript, ESLint, Prettier extensions)	Refers to the main development tools used to write and manage source code.
Frontend framework and/or library	React 18 with Next.js 14	Specifies the framework/library used to build the user interface.
Backend framework and/or library	Express on Node.js with @simplewebauthn/server, Prisma ORM	Defines the server-side frameworks and libraries used to structure the backend, handle routing, verify WebAuthn ceremonies, and interact with the PostgreSQL database.
Browsers supported	Latest versions of Google Chrome, Mozilla Firefox, Microsoft Edge, and Apple Safari on desktop and mobile	Describes the web browsers in which the system is expected to run correctly.

Table 4.7: Hardware Requirements

Category	Requirement	Description
Processor	Development machine: 64-bit quad-core CPU • Server: 1 vCPU or higher	Defines the minimum processing capability required to develop, run, and test the application locally and on the deployment environment.

Table 4.7: Continued

RAM	Development machine: Minimum 8 GB (16 GB recommended) • Server: 1–2 GB	Specifies the memory needed to support the IDE, Node.js, database, and browser tools concurrently. The server requires less RAM but must be sufficient to run the Node.js backend, WebAuthn verification library, and PostgreSQL smoothly.
Storage	Development machine: At least 256 GB total, with ≥ 10–20 GB free for project files, databases, and dependencies • Server: At least 20 GB SSD	Refers to disk capacity for source code, dependencies (Node modules), local databases, logs, and build artifacts.
Display	Development machine: Minimum 13″ display, Full HD (1920×1080) resolution or higher	Defines the minimum display requirements to comfortably design, code, and debug the web application.
Network	Development machine: Stable broadband connection (minimum 10 Mbps) • Server: Reliable internet with public-facing HTTPS endpoint	Describes connectivity requirements for accessing online resources, pushing/pulling from version control, testing WebAuthn flows, and allowing users/examiners to access the deployed system over the internet.
Cloud VM (for deployment)	Example VM spec: 1 vCPU, 1–2 GB RAM, 20 GB SSD, public IP (e.g. AWS EC2 t3.micro / Lightsail / similar)	Represents the minimal cloud or hosting resources required to deploy the Node.js backend, PostgreSQL database, and web assets for demonstration and evaluation of the FYP.

4.3 System Architecture

The IDSafe system architecture is organised into three primary layers: the End User Device, the Backend Application Layer, and the Data Storage Layer, connected through secure HTTPS communication. On the end-user side, the system runs within a web browser built using React/Next.js and leverages the Web Crypto API to perform client-side cryptographic operations such as AES-256-GCM encryption and Argon2id key derivation, ensuring that sensitive vault data is encrypted before transmission. The browser also interacts directly with the platform authenticator, which performs WebAuthn-based biometric authentication within secure hardware. The Backend Application Layer, implemented using Node.js, Express, and TypeScript, exposes REST/JSON APIs for authentication, account management, recovery workflows, and audit logging. It also validates WebAuthn registration and login requests using attestation and signature verification logic. Finally, the Data Storage Layer consists of a PostgreSQL database that stores user accounts, WebAuthn credential metadata, wrapped vault keys, encrypted vault entries, and audit logs, as well as an email/SMTP service responsible for delivering magic-link emails for account recovery. Together, these layers establish a secure, zero-knowledge, and modular architecture in which encryption happens entirely on the client while the backend enforces authentication, manages workflows, and persistently stores only ciphertext and metadata.

4.4 System Analysis

In the system analysis section, we will analyse the IDSafe system in detail and in a systematic and structured manner. Various diagrams including the context diagram, data flow diagrams (DFD), entity relationship diagram (ERD), and the system users' flowcharts will be used to achieve this.

4.4.1 Context Diagram

A context diagram is similar to a data flow diagram (DFD) level 0 and is used interchangeably.

IDSafe Context Diagram Mermaid code:

```
flowchart LR
    subgraph SYSTEM[Password Manager Web Application]
    end

    EU[End User]
    SA[System Administrator]
    ESP[Email Service Provider]
    WA[WebAuthn Platform Authenticator]

    EU --> |Registration, Login, Vault Requests,\nRecovery Passphrase, Account Management| SYSTEM
    SYSTEM --> |Vault Views, Error Messages,\nPassword Generator Output| EU

    SA --> |Admin Login, Config Changes,\nView Logs & Reports| SYSTEM
    SYSTEM --> |Admin Dashboard, Logs,\nStatus & Reports| SA

    SYSTEM --> |Recovery Emails,\nVerification Emails| ESP
    ESP --> |Magic Link HTTP Requests,\nDelivery Responses| SYSTEM

    SYSTEM --> |WebAuthn Challenges,\nAttestation Policy| WA
    WA --> |Registration Response,\nAssertion Response| SYSTEM
```

4.4.2 Data Flow Diagram Level 1

IDSafe Level 1 DFD Mermaid code:

```
flowchart LR
    %% External Entities
    E1[End User]
    E2[System Administrator]
    E3[Email Service Provider]
    E4[WebAuthn Platform Authenticator]
```

```
%% Processes (inside IDSafe)
subgraph IDSafe_System[IDSafe System]
   P1[(P1: Authenticate User)]
   P2[(P2: Manage Vault)]
   P3[(P3: Manage Recovery)]
   P4[(P4: Admin Management)]
   P5[(P5: Audit & Monitoring)]

   D1[(D1: User Accounts)]
   D2[(D2: WebAuthn Credentials)]
   D3[(D3: Encrypted Vault)]
   D4[(D4: Recovery Data)]
   D5[(D5: Audit Logs)]
   D6[(D6: System Configuration)]
end

%% End User flows
E1 --> |Registration & Login Requests| P1
P1 --> |Auth Responses & Errors| E1

E1 --> |Vault CRUD Requests| P2
P2 --> |Decrypted Vault Views| E1

E1 --> |Recovery Request & Passphrase| P3
P3 --> |Recovery Status & New Device Setup| E1

%% Admin flows
E2 --> |Admin Login & Config Changes| P4
P4 --> |Dashboards, Logs, Status| E2

%% Email service flows
P3 --> |Recovery / Magic Link Emails| E3
E3 --> |Magic Link Callback (HTTP Request)| P3
```

%% WebAuthn flows
P1 --> |Registration & Auth Challenges| E4
E4 --> |Attestation & Assertion Responses| P1

%% Data stores
P1 --> |Create/Update User| D1
P1 --> |Store/Update Credential| D2
P1 <-- |Read User/Credential| D1
P1 <-- |Read Credential| D2

P2 --> |Store/Update Ciphertext| D3
P2 <-- |Read Ciphertext| D3

P3 <-- |Read User Account| D1
P3 --> |Store/Update Wrapped Vault Key| D4
P3 <-- |Read Wrapped Vault Key| D4
P3 <-- |Read Encrypted Vault| D3

P4 <-- |Read User & Credential Summaries| D1
P4 <-- |Read Audit Logs| D5
P4 <-- |Read/Write Config| D6

%% Audit & Monitoring
P1 --> |Auth Events| P5
P2 --> |Vault Events| P5
P3 --> |Recovery Events| P5
P4 --> |Admin Events| P5
P5 --> |Write Logs| D5

4.4.2.1 Processes in IDSafe System

Table 4.8 describes the processes in the IDSafe system.

Table 4.8: Processes in IDSafe System (Level 1 DFD)

ID	Process Name	Description
P1	Authenticate User	Handles end-user login and registration using WebAuthn, manages sessions.
P2	Manage Vault	Handles CRUD operations on password vault entries (encrypt/decrypt on client, store ciphertext server-side).
P3	Manage Recovery	Handles account recovery flow using email magic link + recovery passphrase and Argon2id to unwrap the vault key.
P4	Admin Management	Handles admin login plus viewing and managing system-wide data (user summaries, logs, configuration).
P5	Audit & Monitoring	Logs security-relevant events and exposes system health/metrics to admins.

4.4.2.2 Data Stores in IDSafe System

Table 4.9 shows the data stores inside the IDSafe system.

Table 4.9: Data Stores in IDSafe System (Level 1 DFD)

ID	Data Store	What it Holds
D1	User Accounts	User ID, email, status, timestamps; no plaintext secrets.
D2	WebAuthn Credentials	Credential ID, public key, sign counter, attestation metadata, allowed AAGUIDs.
D3	Encrypted Vault	Encrypted vault entries (ciphertext, IV, tag, metadata) per user.

D4	Recovery Data	wrappedVaultKey, Argon2id salt, KDF parameters associated with each user.
D5	Audit Logs	Security and admin events (logins, failures, recovery attempts, admin actions).
D6	System Configuration	Security policies, WebAuthn attestation policy, KDF parameters defaults, email settings.

4.4.2.3 External Entities of IDSafe System

Table 4.10 shows the entities of IDSafe system.

Table 4.10: External Entities of IDSafe System (Level 1 DFD)

ID	Entity	Description
E1	End User	Normal user of IDSafe.

Table 4.10: Continued

E2	System Administrator	Manages configuration, monitoring.
E3	Email Service Provider	Sends verification / recovery emails.
E4	WebAuthn Platform Authenticator	Device authenticator (biometric / screen lock).

4.4.2.4 Data Flow Diagram Description

The primary external entities include the End User (E1), System Administrator (E2), Email Service Provider (E3), and the WebAuthn Platform Authenticator (E4). Each entity exchanges specific inputs and outputs with internal processes, forming the foundation of IDSafe's secure and user-centred workflow.

Authenticate User, which is Process P1, manages the WebAuthn-based registration and login flows for the users. P1 does this by interacting with both the End User (E1) and the WebAuthn Platform Authenticator (E4), while also maintaining associated user and credential records in the User Accounts (D1) and WebAuthn

Credentials (D2) data stores. Process P2, which is **Manage Vault**, will be handling the secure lifecycle of the user's encrypted vault data. These includes storing user credentials' ciphertext in the Encrypted Vault data store (D3) and managing the client-side ciphertext decryption for display on user devices. Process P3, **Manage Recovery**, coordinates the user-triggered account recovery mechanism by issuing magic link emails through the Email Service Provider (E3) and retrieving the wrapped (encrypted) vault key and its metadata from the Recovery Data store (D4).

The administrative functions of the system are encapsulated in Process P4, **Admin Management**, which allows the System Administrator (E2) to view system status, audit logs, and user summaries, while also giving the admins the ability to manage system-wide configurations stored in the System Configuration data store (D6). Finally, **Audit and Monitoring**, which is Process P5, coordinates the recording of security-sensitive events generated by all of the other processes in the system and then stores them in the Audit Logs data store (D5). The storing of these data is that in order to support system compliance, system bug troubleshooting, and anomaly detection. Overall, the Level 1 DFD presents a structured and comprehensive overview of how IDSafe processes information securely and efficiently across its core functional domains.

4.4.3 Entity Relationship Diagram

Each user (USER) may own one or multiple WebAuthn credentials (WEBAUTHN_CREDENTIAL) and multiple encrypted vault entries (VAULT_ENTRY). Account recovery information, including the wrapped vault key and Argon2id parameters, are stored per user in RECOVERY_DATA, while each time-limited magic link's data is stored in RECOVERY_TOKEN. Security-relevant events are captured in AUDIT_LOG, and system-wide settings are stored in SYSTEM_CONFIG. This supports the zero-knowledge design by ensuring that only ciphertext and metadata have persisted on the server.

IDSafe ERD Mermaid code:

```
erDiagram
  USER {
    int user_id PK
```

```
    string email
    string role
    string status
    datetime created_at
    datetime updated_at
    datetime last_login_at
}

WEBAUTHN_CREDENTIAL {
    int credential_id PK
    int user_id FK
    string external_credential_id
    string public_key
    string aaguid
    string attestation_format
    int sign_count
    boolean is_active
    datetime created_at
    datetime last_used_at
}

VAULT_ENTRY {
    int entry_id PK
    int user_id FK
    string ciphertext_blob
    string iv
    string auth_tag
    string metadata_json
    datetime created_at
    datetime updated_at
    datetime deleted_at
    boolean is_deleted
}
```

```
RECOVERY_DATA {
    int recovery_id PK
    int user_id FK
    string wrapped_vault_key
    string kdf_salt
    string kdf_algorithm
    int kdf_time_cost
    int kdf_memory_cost
    int kdf_parallelism
    datetime created_at
    datetime updated_at
}

RECOVERY_TOKEN {
    int token_id PK
    int user_id FK
    string token_hash
    string token_type
    datetime expires_at
    datetime used_at
    datetime created_at
}

AUDIT_LOG {
    int log_id PK
    int user_id FK
    int actor_id FK
    string event_type
    string ip_address
    string user_agent
    string details_json
    datetime created_at
}
```

```
SYSTEM_CONFIG {
    string config_key PK
    string config_value
    string description
    datetime updated_at
    int updated_by_user_id FK
}

DEVICE_KEY {
    int device_id PK
    int user_id FK
    string device_public_key
    string device_label
    string wrapped_dek
    datetime created_at
    datetime last_used_at
}

%% Relationships
USER ||--o{ WEBAUTHN_CREDENTIAL : "has"
USER ||--o{ VAULT_ENTRY : "owns"
USER ||--|| RECOVERY_DATA : "has"
USER ||--o{ RECOVERY_TOKEN : "is issued"
USER ||--o{ AUDIT_LOG : "subject_of"
USER ||--o{ AUDIT_LOG : "actor_of"
USER ||--o{ SYSTEM_CONFIG : "updates"
USER ||--o{ DEVICE_KEY : "has_devices"
```

4.4.4 System User Flowchart

IDSafe End-User Journey Flowchart Mermaid code:

```
flowchart TD
    %% Terminators
    start([Start])
    end([End])

    %% Main entry
    start --> P_open[Process: User opens IDSafe web app]

    %% New vs existing user
    P_open --> D_account{Decision: Does the user have an account?}

    %% --- New user registration branch ---
    D_account -->|No| IO_register[/Input: Enter email and recovery passphrase/]
    IO_register --> P_webauthn_reg[Process: Register WebAuthn credential on device]
    P_webauthn_reg --> P_create_vault[Process: Create vault key and empty vault]
    P_create_vault --> P_dashboard[Process: Show vault dashboard]

    %% --- Existing user branch: can they use device WebAuthn? ---
    D_account -->|Yes| D_can_auth{Decision: Can user use device WebAuthn?}
    D_can_auth -->|Yes| P_login[Process: WebAuthn login with biometric]
    P_login --> P_dashboard

    %% --- Recovery branch (device lost / WebAuthn unavailable) ---
    D_can_auth -->|No| IO_recover_email[/Input: Enter email to request recovery link/]
    IO_recover_email --> IO_recover_link[/Output: System sends recovery magic link/]
    IO_recover_link --> P_open_link[Process: User opens recovery magic link]
    P_open_link --> IO_recover_pass[/Input: Enter recovery passphrase/]
    IO_recover_pass --> P_recover_key[Process: Derive KEK and decrypt vault key]
```

```
    P_recover_key --> D_recover_ok{Decision: Decryption successful?}

    D_recover_ok -->|No| P_recover_fail[Process: Show error and allow retry]
    P_recover_fail --> IO_recover_pass

    D_recover_ok -->|Yes| P_reg_new_webauthn[Process: Register new WebAuthn credential on this device]
    P_reg_new_webauthn --> P_dashboard

    %% --- Dashboard actions ---
    P_dashboard --> D_action{Decision: User action?}

    D_action -->|Manage vault| P_manage_vault[Process: Add, view, edit or delete vault entries]
    P_manage_vault --> P_dashboard

    D_action -->|Manage account| P_manage_account[Process: Manage recovery passphrase or authenticators]
    P_manage_account --> P_dashboard

    D_action -->|Logout| P_logout[Process: Clear keys and end session]
    P_logout --> end
```

IDSafe System Administrator Journey Flowchart Mermaid code:

```
flowchart TD
    %% Terminators
    start([Start])
    end([End])

    %% Entry
    start --> P_open[Process: Admin opens IDSafe admin portal]

    %% Admin login
    P_open --> IO_login[/Input: Enter admin credentials (and 2FA/WebAuthn)/]
```

```
IO_login --> D_auth{Decision: Authentication successful?}

D_auth -->|No| P_auth_fail[Process: Show error and retry]
P_auth_fail --> IO_login

D_auth -->|Yes| P_dashboard[Process: Display admin dashboard]

%% Main admin actions
P_dashboard --> D_action{Decision: Admin action?}

%% View users
D_action -->|View user accounts| P_view_users[Process: List user accounts and status]
P_view_users --> D_action

%% Manage user status
D_action -->|Manage user status| IO_select_user[/Input: Select user account/]
IO_select_user --> P_change_status[Process: Lock/unlock/deactivate account]
P_change_status --> D_action

%% View audit logs
D_action -->|View audit logs| P_view_logs[Process: Display security & activity logs]
P_view_logs --> IO_filter_logs[/Input: Apply filters (user, date, event type)/]
IO_filter_logs --> P_logs_filtered[Process: Show filtered log results]
P_logs_filtered --> D_action

%% Configure security policies
D_action -->|Configure security policies| P_config_security[Process: Configure session timeout, rate limits, KDF/WebAuthn policies]
P_config_security --> D_action

%% System health & monitoring
```

```
D_action -->|View system health| P_view_health[Process: View server status, performance metrics]
P_view_health --> D_action

%% Backup & restore
D_action -->|Manage backup & restore| P_backup[Process: Trigger or schedule database backup]
P_backup --> D_action

%% Application settings
D_action -->|Manage app configuration| P_app_config[Process: Configure email server, base URLs, general settings]
P_app_config --> D_action

%% Logout
D_action -->|Logout| P_logout[Process: End admin session and clear auth tokens]
P_logout --> end
```

4.5 Database Design

4.5.1 Relational Schema

Table 4.11 shows the relational schema for each table in the database of IDSafe system.

Table 4.11: IDSafe Database Relational Schema

Database Table	Relational Schema
USER	USER(user_id PK, email UNIQUE NOT NULL, role NOT NULL, status NOT NULL, created_at NOT NULL, updated_at NOT NULL, last_login_at NULL)

WEBAUTHN_ CREDENTIAL	WEBAUTHN_CREDENTIAL(credential_id PK, user_id FK → USER(user_id), external_credential_id UNIQUE NOT NULL, public_key NOT NULL, aaguid NOT NULL, attestation_format NOT NULL, sign_count NOT NULL, is_active NOT NULL, created_at NOT NULL, last_used_at NULL)

VAULT_ENTRY	VAULT_ENTRY(entry_id PK, user_id FK → USER(user_id), ciphertext_blob NOT NULL, iv NOT NULL, auth_tag NOT NULL, metadata_json NULL, created_at NOT NULL, updated_at NOT NULL, deleted_at NULL, is_deleted NOT NULL)
RECOVERY_DATA	RECOVERY_DATA(recovery_id PK, user_id FK → USER(user_id) UNIQUE, wrapped_vault_key NOT NULL, kdf_salt NOT NULL, kdf_algorithm NOT NULL, kdf_time_cost NOT NULL, kdf_memory_cost NOT NULL, kdf_parallelism NOT NULL, created_at NOT NULL, updated_at NOT NULL)

RECOVERY_TOKEN	RECOVERY_TOKEN(token_id PK, user_id FK → USER(user_id), token_hash UNIQUE NOT NULL, token_type NOT NULL, expires_at NOT NULL, used_at NULL, created_at NOT NULL)
AUDIT_LOG	AUDIT_LOG(log_id PK, user_id FK → USER(user_id) NULL, actor_id FK → USER(user_id) NULL, event_type NOT NULL, ip_address NULL, user_agent NULL, details_json NULL, created_at NOT NULL)
SYSTEM_CONFIG	SYSTEM_CONFIG(config_key PK, config_value NOT NULL, description NULL, updated_at NOT NULL, updated_by_user_id FK → USER(user_id) NULL)
DEVICE_KEY	DEVICE_KEY(device_id PK, user_id FK → USER(user_id), device_public_key, device_label, wrapped_dek, created_at, last_used_at)

4.5.2 Data Dictionary

Table 4.12 to 4.18 show the data dictionary for each of the database tables in the IDSafe system.

Table 4.12: user Table Data Dictionary

Field Name	Data Type	Constraints	Description
user_id	BIGSERIAL	PK, NOT NULL	Unique identifier for each user account.
email	VARCHAR(255)	UNIQUE, NOT NULL	User's primary email address.
role	VARCHAR(20)	NOT NULL, values: END_USER, ADMIN	User's role in the system.
status	VARCHAR(20)	NOT NULL, values: ACTIVE, LOCKED, DEACTIVATED	Current status of the user account.
created_at	TIMESTAMPTZ	NOT NULL, default NOW()	Timestamp when account was created.
updated_at	TIMESTAMPTZ	NOT NULL, default NOW()	Timestamp of last account update.
last_login_at	TIMESTAMPTZ	NULL	Timestamp of last successful login.

Table 4.13: webauthn_credential Table Data Dictionary

Field Name	Data Type	Constraints	Description
credential_id	BIGSERIAL	PK, NOT NULL	Unique identifier for each credential record.

user_id	BIGINT	FK → users(user_id), NOT NULL, ON DELETE CASCADE	Owner user of this credential.
external_credential_id	TEXT	UNIQUE, NOT NULL	WebAuthn credential ID (encoded).
public_key	TEXT	NOT NULL	Stored WebAuthn public key material.
aaguid	TEXT	NOT NULL	Authenticator Attestation GUID identifier.
attestation_format	TEXT	NOT NULL	WebAuthn attestation format string.
sign_count	BIGINT	NOT NULL, default 0	WebAuthn signature counter for replay detection.
is_active	BOOLEAN	NOT NULL, default TRUE	Indicates if credential is currently active.
created_at	TIMESTAMPTZ	NOT NULL, default NOW()	Timestamp when credential was registered.
last_used_at	TIMESTAMPTZ	NULL	Timestamp when credential was last used.

Table 4.14: vault_entry Table Data Dictionary

Field Name	Data Type	Constraints	Description
entry_id	BIGSERIAL	PK, NOT NULL	Unique identifier for each vault entry.
user_id	BIGINT	FK → users(user_id), NOT NULL, ON DELETE CASCADE	Owner user of this vault entry.
ciphertext_blob	TEXT	NOT NULL	Encrypted vault entry data blob.
iv	TEXT	NOT NULL	AES-GCM initialisation vector for encryption.
auth_tag	TEXT	NOT NULL	AES-GCM authentication tag for integrity.
metadata_json	JSONB	NULL	Encrypted or auxiliary metadata for entry.
created_at	TIMESTAMPTZ	NOT NULL, default NOW()	Timestamp when entry was created.
updated_at	TIMESTAMPTZ	NOT NULL, default NOW()	Timestamp when entry was last updated.
deleted_at	TIMESTAMPTZ	NULL	Timestamp when entry was soft-deleted.
is_deleted	BOOLEAN	NOT NULL, default FALSE	Indicates whether entry is marked deleted.

Table 4.15: recovery_data Table Data Dictionary

Field Name	Data Type	Constraints	Description
recovery_id	BIGSERIAL	PK, NOT NULL	Unique identifier for recovery data record.
user_id	BIGINT	FK → users(user_id), UNIQUE, NOT NULL, ON DELETE CASCADE	User associated with this recovery data.
wrapped_vault_key	TEXT	NOT NULL	Vault key encrypted with KEK.
kdf_salt	TEXT	NOT NULL	Salt used for Argon2id derivation.
kdf_algorithm	VARCHAR(50)	NOT NULL	Name of KDF algorithm (e.g. argon2id).
kdf_time_cost	INTEGER	NOT NULL	Argon2id time cost parameter.
kdf_memory_cost	INTEGER	NOT NULL	Argon2id memory cost parameter.
kdf_parallelism	INTEGER	NOT NULL	Argon2id parallelism parameter.
created_at	TIMESTAMPTZ	NOT NULL, default NOW()	Timestamp when recovery data was created.
updated_at	TIMESTAMPTZ	NOT NULL, default NOW()	Timestamp when recovery data was updated.

Table 4.16: recovery_token Table Data Dictionary

Field Name	Data Type	Constraints	Description
token_id	BIGSERIAL	PK, NOT NULL	Unique identifier for each recovery token.
user_id	BIGINT	FK → users(user_id), NOT NULL, ON DELETE CASCADE	User to whom this token belongs.
token_hash	TEXT	UNIQUE, NOT NULL	Hashed value of recovery or magic token.
token_type	VARCHAR(50)	NOT NULL	Type of token (recovery, verify email).
expires_at	TIMESTAMPTZ	NOT NULL	Expiry timestamp for the token.
used_at	TIMESTAMPTZ	NULL	Timestamp when token was consumed.
created_at	TIMESTAMPTZ	NOT NULL, default NOW()	Timestamp when token was created.

Table 4.17: audit_log Table Data Dictionary

Field Name	Data Type	Constraints	Description
log_id	BIGSERIAL	PK, NOT NULL	Unique identifier for each audit log entry.
user_id	BIGINT	FK → users(user_id), NULL, ON DELETE SET NULL	User who is subject of the event.
actor_id	BIGINT	FK → users(user_id), NULL, ON DELETE SET NULL	User performing the action, if applicable.

event_type	VARCHAR(100)	NOT NULL	Type of event recorded in audit log.
ip_address	INET	NULL	IP address from which request originated.
user_agent	TEXT	NULL	HTTP user-agent string of client.
details_json	JSONB	NULL	Additional structured event details.
created_at	TIMESTAMPTZ	NOT NULL, default NOW()	Timestamp when audit event was recorded.

Table 4.18: system_config Table Data Dictionary

Field Name	Data Type	Constraints	Description
config_key	VARCHAR(100)	PK, NOT NULL	Unique key name for configuration setting.
config_value	TEXT	NOT NULL	Value of configuration parameter.
description	TEXT	NULL	Short description of configuration purpose.
updated_at	TIMESTAMPTZ	NOT NULL, default NOW()	Timestamp when configuration was last updated.
updated_by_user_id	BIGINT	FK → users(user_id), NULL, ON DELETE SET NULL	Admin user who last changed this setting.

Table 4.19: device_key Table Data Dictionary

Field Name	Data Type	Constraints	Description
device_id	BIGSERIAL	PK, NOT NULL	Unique identifier for each registered device bound to the user account.
user_id	BIGINT	FK → users(user_id), NOT NULL, ON DELETE CASCADE	The user who owns this device-binding record. If the user account is deleted, device records are removed automatically.
device_public_key	TEXT	NOT NULL	Public key generated on the user's device (JWK or encoded), used to wrap/unwrap the Vault Key (DEK). The corresponding private key remains on the device.
device_label	TEXT	NULL	Optional human-readable label for the device (e.g., "Laptop", "iPhone 14 Pro"). Helps users identify bound devices.
wrapped_dek	TEXT	NOT NULL	Vault Key (DEK) encrypted (wrapped) using this device's public key. Allows only this device to decrypt the DEK after WebAuthn authentication.
created_at	TIMESTAMPTZ	NOT NULL, default NOW()	Timestamp indicating when the device-binding was created.
last_used_at	TIMESTAMPTZ	NULL	Timestamp indicating when this device last accessed the vault successfully. Useful for session auditing and device management.

Concept: Passphrase only for recovery

- **Normal access to the vault**
  - User authenticates with **WebAuthn** (biometric / device unlock).
  - The **vault key (DEK)** is stored locally on the device in some secure storage (e.g., IndexedDB + OS/device protection).
  - After WebAuthn login, the app reads the local vault key and decrypts the vault.
  - **No passphrase is asked.**
- **Account recovery**
  - If the user loses the device (and the local vault key), the **only way** to get the vault key back is via a **backup copy** that was encrypted with a **recovery passphrase**.
  - The recovery passphrase is run through **Argon2id** to derive a strong key (KEK), which decrypts the backup of the vault key.
  - This is the *only time* the passphrase and Argon2id are used.

So **Argon2id and the passphrase are not part of daily login**; they are just the "break glass in emergency" mechanism to recover the vault key on a new device.

Key Management Design

Key Management Design

The system is designed so that users access their password vault **only** by authenticating with **WebAuthn** (using biometrics or screen unlock on their device). A separate **account recovery passphrase** exists solely for disaster recovery when a device or authenticator is lost. This passphrase is never required for normal logins.

Key Types

The following keys and secrets are used:

1. **WebAuthn Key Pair (per device)**
   - Generated by a trusted platform authenticator (e.g., Secure Enclave, TPM, Android Keystore).
   - Private key remains inside the authenticator; only the public key and metadata are stored on the server.
   - Used to prove the user's identity to the server (authentication), not to encrypt vault data.
2. **Vault Key / Data Encryption Key (DEK)**
   - A random 256-bit symmetric key generated once per user.
   - Used with AES-256-GCM to encrypt and decrypt all vault entries.
   - Stored locally on the user's device in secure storage; never stored in plaintext on the server.
3. **Recovery Passphrase (user-chosen)**
   - Only used in **recovery scenarios**, not during normal logins.

    - Known by the user and entered when they need to recover their vault on a new device.
4. **Key Encryption Key (KEK)**
    - Derived from the recovery passphrase using **Argon2id** with a random salt.
    - A 256-bit symmetric key.
    - Used solely to encrypt and decrypt a **backup copy** of the Vault Key.

Role of Argon2id

Argon2id serves as a **password-based key derivation function** to protect the backup of the Vault Key against offline brute-force attacks.

- Input: recovery passphrase + random salt
- Output: 256-bit KEK
- Parameters (example): time cost = 3, memory = 64 MiB, parallelism = 1

By making each guess slow and memory-expensive, Argon2id raises the cost for an attacker who might steal the database and try to guess the recovery passphrase offline. Argon2id is **never used during normal biometric login**; it is only invoked when the user actively performs account recovery.

Key Life-Cycle

a) Initial Setup

1. The user registers using **WebAuthn**, so the server stores the WebAuthn public key and credential ID.
2. The client generates a random 256-bit **Vault Key (DEK)**.
3. The Vault Key is stored locally on the device in secure storage, accessible after WebAuthn/biometric unlock.
4. The user is asked to define a **recovery passphrase**.
5. The client generates a random salt and runs Argon2id to derive the **KEK**:
    - `KEK = Argon2id(passphrase, salt, params)`
6. The client uses the KEK with AES-256-GCM to encrypt the Vault Key, producing `wrappedVaultKey`.
7. The server stores only `wrappedVaultKey`, the salt and Argon2id parameters, plus the WebAuthn metadata.
    - The recovery passphrase, KEK and plaintext Vault Key never leave the device.

---

b) Normal Login (Day-to-Day Use)

1. The user visits the application and performs **WebAuthn authentication** (e.g., biometric on their device).
2. After successful WebAuthn verification, the client loads the locally stored **Vault Key** from secure storage.

3. Using the Vault Key, the client decrypts the AES-GCM encrypted vault entries.
4. The user can view and manage passwords.

Importantly, **no recovery passphrase is requested** in this flow; the experience is as simple as "open the site → biometric → vault opens".

---

c) Account Recovery on a New Device

If the user loses their original device or WebAuthn credential:

1. The user accesses a **recovery page** (via an email magic link) and proves ownership of their account (by email verification).
2. The client fetches the stored `wrappedVaultKey`, together with the salt and Argon2id parameters.
3. The user enters their **recovery passphrase**.
4. The client runs `KEK = Argon2id(passphrase, salt, params)` to derive the KEK.
5. The client decrypts `wrappedVaultKey` with the KEK to obtain the plaintext **Vault Key**.
6. The Vault Key is stored in secure storage on the **new** device, and the user registers a **new WebAuthn credential** on this device.
7. From this point on, the user again logs in normally with WebAuthn/biometrics only.

Security Properties

- **Biometric-only everyday UX**
  - Normal vault access requires only WebAuthn; the recovery passphrase stays "cold" and is only used in emergencies.
- **Strong offline protection of the backup Vault Key**
  - An attacker who steals the database obtains only the encrypted backup (`wrappedVaultKey`) and the Argon2id parameters.
  - They must guess the recovery passphrase and perform expensive Argon2id computations for each guess to recover the Vault Key.
- **Zero-knowledge server design**
  - The server never learns the recovery passphrase, KEK or Vault Key in plaintext.
  - All encryption and decryption of vault contents happen on the client.