FMB Log – Admin Guide

Version 1.0.0

2026-01-12

Contents

1 Admin Guide

Dieser Teil richtet sich an Administratoren und Key-User. Er erklärt die Aufgaben, die für einen stabilen Betrieb notwendig sind: Ersteinrichtung, Nutzer- und Rechteverwaltung, Pflege von Freigabewerten sowie Betrieb/Backup der Datenbank.

1.1 Inhalt

- Ersteinrichtung
- Sicherheitsarchitektur
- Benutzer
- Gruppen & Rechte
- Freigabewerte (FGW)
- Betrieb & Datenbank
- Technische Dokumentation
- Troubleshooting

1.2 Ablaufdiagramm

Zusammenfassung

- Admins legen die Grundstruktur fest: Gruppen, Rechte, Nutzer, Stammdaten.
- Danach können Endanwender im Tagesgeschäft (Import/Gebinde/Tagesabrechnung) arbeiten.
- Änderungen an FGW und NV sind fachlich kritisch und sollten nur nach Freigabeprozess erfolgen.

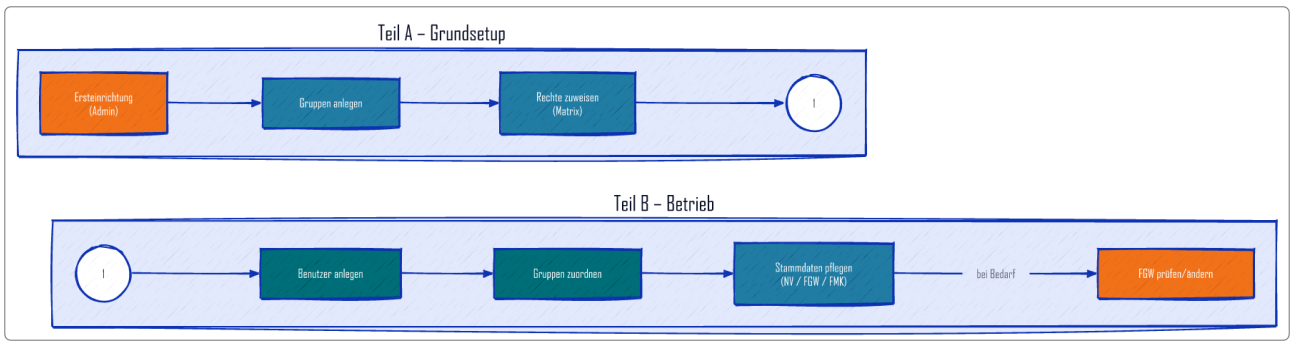

Figure 1: Ablaufdiagramm (Admin)

2 Ersteinrichtung

2.1 Administrator anlegen (einmalig)

Beim ersten Start ist die Anwendung im Setup-Modus. In diesem Schritt wird der **erste Administrator** angelegt. Dieser Account ist notwendig, weil ohne Admin niemand Nutzer, Gruppen und Stammdaten verwalten kann.

Nach dem Anlegen des Administrators wechselt die Anwendung zur Anmeldung. Ab diesem Zeitpunkt können weitere Nutzer und Gruppen angelegt werden.

Technischer Hinweis: Die Anwendung erkennt „Ersteinrichtung erforderlich“, wenn in der Datenbank noch **kein** Benutzer existiert.

Empfehlung

- Legen Sie mindestens einen zweiten Admin an, sobald die Grundkonfiguration steht (Single-Admin-Risiko).

2.2 Datenquelle wählen (wenn bereits eine produktive DB existiert)

Wenn FMB Log auf mehreren Arbeitsplätzen betrieben wird, existiert in der Regel bereits eine produktive Hub-Datenquelle (DB + `vaults/` + `protocols/` im gemeinsamen Ordner).

In diesem Fall muss auf einem neuen Client **keine Ersteinrichtung** durchgeführt werden. Stattdessen:

1. Im Setup-Modus auf **Datenquelle ändern** klicken.
2. Den Pfad zur bestehenden Hub-DB auswählen (z. B. im Netzlaufwerk).
3. Die Anwendung lädt die Datenquelle; wenn bereits Benutzer existieren, wechselt FMB Log automatisch in den normalen **Anmeldemodus**.

Kurzfassung

- **Nur einmal** im Betrieb: Ersteinrichtung + erster Admin.
- **Alle weiteren Clients**: Datenquelle ändern → anmelden.

2.3 Erstkonfiguration (empfohlen)

Nach der Ersteinrichtung empfiehlt sich eine kurze Grundkonfiguration, damit das Tagesgeschäft sauber getrennt ist (Messung/Prüfung/Administration):

1. Legen Sie mindestens eine **Benutzergruppe** an (z. B. „Messung“, „Auswertung“, „Key-User“).
2. Weisen Sie den Gruppen die benötigten Rechte in der **Rechte-Matrix** zu.
3. Legen Sie Nutzer an und ordnen Sie sie den Gruppen zu.

Warum dieser Schritt wichtig ist: In der Praxis sollen kritische Aktionen (z. B. FGW ändern, Passwörter zurücksetzen) nur wenigen Personen möglich sein, während der Import und die Arbeit mit Gebinden breiter verfügbar sein kann.

Kurzfassung

- Admin anlegen → anmelden → Gruppen/Rechte/Nutzer einrichten
- Rechte steuern Sichtbarkeit und Bearbeitung in der UI

2.4 Integritätsschutz – DB-Key-Zertifikat (verpflichtend)

Der Integritätsschutz verhindert, dass Manipulationen an **Benutzern/Gruppen/Rechten** in der SQLite-Datei unbemerkt bleiben. Technisch werden die relevanten Datensätze signiert.

Damit ein Angreifer den Schutz nicht durch einen einfachen Austausch des DB-Public-Keys aushebeln kann, ist ein **DB-Key-Zertifikat** verpflichtend: Der DB-Public-Key wird durch eine Root-Signatur autorisiert und die Anwendung prüft diese Signatur gegen den fest eingebauten Root-Public-Key.

Dabei gilt:

- Der **Root-Public-Key** ist fest in der Anwendung hinterlegt (Compile-Time).
- Der **Root-Private-Key** bleibt beim Administrator und wird nur zum Erstellen des Zertifikats benötigt (z. B. beim Build/Deployment).

2.4.1 Vorgehen

1. Root-Schlüssel erzeugen (einmalig, nur auf dem Admin-Build-Rechner):
   - `pnpm -s integrity:rootkey`
   - Ergebnis: Public Key in `src/lib/security/rootPublicKey.jwk.json`, Private Key in `.secrets/…` (darf nicht committed werden)
2. In der Anwendung: **Einstellungen → Integritätsschutz → Aktivieren**
   - Dadurch wird `<db>.integrity.pub.json` neben der DB erzeugt.
3. Zertifikat für den DB-Public-Key erzeugen:
   - `pnpm -s integrity:certify --db-public <db>.integrity.pub.json --out <db>.integrity.dbkey.json`
4. Auf dem Netzlaufwerk: Die Dateien gegen Änderungen schützen (ACL/Schreibrechte):
   - `<db>.integrity.dbkey.json` (Zertifikat)
   - `<db>.integrity.pub.json` (Public Key)

Hinweise

- Ohne `<db>.integrity.dbkey.json` gilt der Integritätsschutz als **nicht funktionsfähig** (Fail-Closed) und FMB Log blockiert sicherheitsrelevante Vorgänge, bis das Zertifikat wieder vorhanden ist.
- Bewahren Sie den Root-Private-Key außerhalb des Repos sicher auf.
- Das Zertifikat ist nicht geheim, aber wichtig für Recovery/Deployment: Sie können es zusätzlich separat sichern.

3 Sicherheitsarchitektur

Dieses Kapitel beschreibt die in FMB Log umgesetzte Sicherheitsarchitektur und ordnet sie konkreten Risiko-Szenarien zu. Ziel ist **kein absoluter Schutz vor lokaler Manipulation**, sondern eine praktikable Absicherung für den Mehrnutzerbetrieb mit einer gemeinsam genutzten SQLite-Datenbasis (z. B. Netzlaufwerk).

3.1 Ziel & Threat Model (Kurzfassung)

FMB Log wird typischerweise in einer Umgebung betrieben, in der mehrere Nutzer auf dieselben Daten zugreifen:

- Die Anwendung läuft als Desktop-App unter dem Windows-Benutzerkonto des jeweiligen Nutzers.
- Die Daten (SQLite-DB und Vaults) können in einem gemeinsamen Ordner liegen.
- Ein Nutzer *kann* potentiell direkten Dateizugriff auf die DB haben (z. B. via SQLite-Browser).

Damit ist "jemand kann den Inhalt der DB-Datei ändern" ein realistisches Szenario. Die Sicherheitsmaßnahmen sind darauf ausgelegt, **unbemerkte Rechte- und Benutzer-Manipulationen zu verhindern** und **Passwort-/Account-Übernahmen über reine DB-Manipulation deutlich zu erschweren**.

3.2 Komponenten

3.2.1 1) Passwort-Hashing (Argon2id) + Benutzer-Pepper (Stronghold)

Problem: Wenn ein Passwort-Hash nur vom Passwort abhängt, kann ein Angreifer (mit Schreibzugriff auf die DB) den eigenen Hash in das Konto eines anderen Nutzers kopieren und so dessen Passwort "ersetzen" (Hash-Swap-Angriff).

Umsetzung in FMB Log:

- Das Passwort wird per **Argon2id** gehasht (Backend/Rust, PHC-String).
- Zusätzlich besitzt jeder Nutzer ein **zufälliges Pepper** (32 Byte).
- Das Pepper wird **nicht in der DB** gespeichert, sondern in einer **Stronghold-Vault**:
  - Datei: `vaults/<user_id>.vault` (neben der DB im gemeinsamen Datenordner)
  - Die Vault ist verschlüsselt mit einem Key, der aus `user_id + password` per Argon2id abgeleitet wird.
- Der gespeicherte Hash ist **user-gebunden**: Es wird nicht nur `password`, sondern `user_id:password:pepper` gehasht.

Folge: Ein Hash-Swap aus der DB allein funktioniert nicht, weil das Pepper pro Nutzer verschieden und geheim ist.

3.2.2 2) Passwort-Änderung und Admin-Reset

- **Passwort ändern (Nutzer):** Das bestehende Pepper bleibt gleich, wird aber mit dem neuen Passwort neu verschlüsselt (Vault-Re-Keying). Anschließend wird der Hash neu berechnet.
- **Admin-Reset:** Es wird ein **neues Pepper** und ein **temporäres Passwort** erzeugt; Vault + Hash werden ersetzt und das Flag `must_change_password` gesetzt.

3.2.3 3) Integritätsschutz für Benutzer/Gruppen/Rechte (Ed25519-Signaturen)

Problem: Selbst mit sicherem Passwort-Hashing könnte ein Nutzer mit Dateizugriff `is_admin=1` setzen oder Berechtigungen manipulieren.

Umsetzung in FMB Log:

- Für sicherheitsrelevante Tabellen wird pro Datensatz eine **Ed25519-Signatur** gespeichert:
  - `users.signature`
  - `groups.signature`
  - `user_groups.signature`
  - `group_permissions.signature`
- Bei aktivem Integritätsschutz werden diese Signaturen **bei der Anmeldung und bei Rechtesync/Anzeige geprüft**.
- Ist eine Signatur ungültig, wird das als Manipulation interpretiert und der Zugriff wird blockiert.

3.2.4 4) DB-Public-Key und DB-Key-Zertifikat (Root-Trust)

Damit die Signaturprüfung nicht durch Austausch des Public Keys ausgehebelt werden kann, gibt es zwei Stufen:

1. **DB Public Key** (neben der DB)
   - Datei: `<db>.integrity.pub.json`
   - Wird von der App erzeugt, wenn Integritätsschutz aktiviert wird.
2. **DB-Key-Zertifikat** (verpflichtend)
   - Datei: `<db>.integrity.dbkey.json`
   - Enthält den **DB-Public-Key** und eine **Root-Signatur über diesen DB-Key** (kein Root-Public-Key).
   - Die App enthält den **Root Public Key fest eingebaut** (Compile-Time) und prüft damit die Root-Signatur.

Folge: Ohne Zertifikat startet FMB Log den Integritätsschutz nicht (Fail-Closed), da sonst ein Austausch des DB-Public-Keys durch reine Datei-Manipulation möglich wäre.

Warum braucht man '<db>.integrity.dbkey.json', obwohl der Root-Public-Key in der App steckt?

Der Root-Public-Key allein reicht nicht aus: Er sagt nur, **womit** eine Root-Signatur geprüft wird.
Das Zertifikat liefert zusätzlich **welcher DB-Public-Key** gültig sein soll *und* die dazugehörige Root-Signatur. Ohne Zertifikat gibt es für die App keinen vertrauenswürdigen Bezug zwischen „dieser DB-Public-Key" und „ist durch Root autorisiert".

3.2.5 5) Stronghold-Vault für den Signierschlüssel (Admin-Entsperren)

Der private Signierschlüssel der DB (Ed25519) wird **nicht in der DB** abgelegt:

- Datei: `vaults/<db>.integrity.vault`
- Verschlüsselt in **Stronghold**, entsperrt über ein **Signier-Passwort** (session-basiert).

Hinweis

- Das Signier-Passwort wird beim Aktivieren des Integritätsschutzes festgelegt und gilt anschließend für alle Admins (shared secret), damit mehrere Admins signieren können.
- Es muss nicht identisch mit dem Login-Passwort eines Admin-Kontos sein.

3.2.6 6) Tauri Capabilities / Permissions (Reduzierte Angriffsfläche)

Tauri erlaubt fein granulare Berechtigungen für native Aktionen (Dateizugriffe, opener, stronghold, SQL, …). Dadurch wird die Angriffsfläche reduziert, falls es zu:

- fehlerhaften UI-Routen,
- unerwarteten Script-Ausführungen,
- oder einer späteren Erweiterung mit externen Inhalten

kommt. Nicht benötigte native Aktionen werden nicht freigegeben.

3.2.7 7) SQL-Injection-Schutz (Parameter Binding)

SQL-Statements werden über `@tauri-apps/plugin-sql` ausgeführt und verwenden gebundene Parameter (z. B. `$1`). Dadurch werden klassische **SQL-Injection-Angriffe über Eingabefelder** verhindert (sofern keine String-Konkatenation zu SQL erfolgt).

3.3 Risiko-Szenarien: Was wird womit abgedeckt?

Die folgende Übersicht ist bewusst praxisnah: **welches Risiko** ist realistisch, **welche Komponente** wirkt dagegen, und **was bleibt** als Restrisiko.

Risiko-Szenario	Wirkung in der Praxis	Abdeckung durch	Restrisiko / Hinweise
Nutzer setzt sich in SQLite `is_admin=1` oder ändert Gruppen/Rechte	Privilegieneskalation	Integritätsschutz (Signaturen)	DoS bleibt möglich (z. B. Signaturen löschen → Login blockiert). Schutz gegen Schreibzugriff ist grundsätzlich nicht vollständig lösbar ohne Zugriffskontrolle auf Dateiebene.
Hash-Swap: eigener Passwort-Hash wird bei fremdem Nutzer eingetragen	Account-Übernahme	Argon2id + per-User Pepper in Stronghold	Wenn ein Angreifer **zusätzlich** die Pepper-Vault des Zielusers überschreiben darf und Integrität deaktiviert ist, kann er trotzdem "resetten". Integritätsschutz reduziert dieses Risiko deutlich.
Austausch des DB-Public-Keys, um manipulierte Datensätze selbst zu signieren	Integrität aushebeln	DB-Key-Zertifikat + Root Public Key in Binary	Ohne Zertifikat wird Integrität nicht als aktiv betrachtet (Fail-Closed). Zertifikat + ACLs sind Pflicht.
Manipulation/Löschung von Vault-Dateien (`vaults/*.vault`)	Login-Probleme / "Passwort falsch" / DoS	Stronghold-Verschlüsselung	DoS bleibt möglich, wenn Dateien gelöscht werden können. Empfehlung: Vault-ACLs so eng wie möglich.
Admin-Operation ohne Hub-Zugriff (Netzlaufwerk offline)	Inkonsistente Zustände (z. B. Nutzer in DB ohne Vault)	UI-Gating + transaktionale/„cleanup"-Logik	Admin-Funktionen sind in diesem Zustand bewusst deaktiviert. Fach-/Messdaten können weiter über die lokale Replica bearbeitet und später synchronisiert werden.
SQL-Injection über Eingabefelder	Datenverlust / Rechteänderung	Parameter Binding	Schutz gilt nur, solange SQL nicht per String-Konkatenation gebaut wird.
Auslesen von Messdaten aus DB-Datei	Vertraulichkeitsverlust	(keine, DB ist nicht verschlüsselt)	Schutz nur über Datei-/Share-Berechtigungen. Optional: separate Verschlüsselung wäre ein eigenes Projekt mit Trade-offs.
Malware/Administrator auf dem Client	Vollzugriff	(keine)	Nicht im Scope: Ein kompromittiertes System kann immer Daten auslesen/manipulieren.

Risiko-Szenario	Wirkung in der Praxis	Abdeckung durch	Restrisiko / Hinweise
Manipulation von Fach-/Messdaten (Gebinde, Messungen, FGW)	Falsche Freigabe / verfälschte Historie	Messdaten-Signaturen (User-Key), Stammdaten-Signaturen (Delegation), Protokoll-Integrität (BLAKE3) und Tagesabrechnungs-Snapshot (BLAKE3 + optional TSA)	DoS bleibt möglich (Dateien löschen/überschreiben). Bei aktivem Integritätsschutz werden nicht verifizierbare Datensätze fail-closed als ungültig behandelt. Tagesabrechnungen können nachträglich ungültig werden, wenn enthaltene Messungen später ungültig gesetzt werden.

3.4 Betriebsempfehlungen (Admin)

1. **DB-Key-Zertifikat verwenden (verpflichtend).** Ohne Zertifikat kann ein Angreifer bei Schreibzugriff auf den Ordner den Public Key austauschen.
2. **ACLs im gemeinsamen Ordner setzen:**
   - Public-Key + Zertifikat: für normale Nutzer **read-only**, für Admins write.
   - Integrity-Vault: möglichst nur Admins (kein Grund, dass normale Nutzer diese Datei lesen).
   - Pepper-Vaults: wenn möglich pro User einschränken (verhindert unnötige DoS-/Manipulationsmöglichkeiten).
3. **Backups/Versionierung der Hub-DB** (Netzlaufwerk): schützt vor versehentlichen Löschungen und ermöglicht Recovery.

3.5 Admin entfernen / Signierrechte entziehen

Wenn ein Nutzer nicht mehr Admin sein soll, ist wichtig zu unterscheiden, **welches Risiko** vorliegt:

3.5.1 1) Nur Admin-Rolle entziehen (kein Key-Leak vermutet)

1. In **Administration → Benutzer** den Haken/Status **Admin** entfernen und speichern.
2. Den Nutzer abmelden lassen (oder App schließen), damit eine bestehende Session nicht weiter als Admin arbeitet.
3. Auf Dateiebene die Berechtigungen prüfen/anpassen (insbesondere `vaults/<db>.integrity.vault` und `<db>.integrity.*`).

   Hinweis: Eine laufende Session kann nicht "remote" beendet werden. Wenn das kritisch ist, Nutzer zusätzlich deaktivieren (`is_active=0`) und später wieder aktivieren.

3.5.2 2) Signier-Passwort könnte bekannt sein (Zugriff entziehen)

Wenn der (ehemalige) Admin das **Signier-Passwort** kennen könnte, reicht das reine Demoten nicht:

- In **Einstellungen → Integritätsschutz → Schlüsselmanagement** das **Signier-Passwort ändern**.
- Danach müssen alle verbleibenden Admins das neue Signier-Passwort kennen, um weiterhin signieren zu können.

3.5.3 3) Private Key könnte kompromittiert sein (stärkste Maßnahme)

Wenn nicht ausgeschlossen werden kann, dass der private Signierschlüssel kopiert wurde, hilft ein Passwortwechsel nicht mehr. Dann:

1. In **Einstellungen → Integritätsschutz → Schlüsselmanagement** den **Signierschlüssel rotieren**.
2. Die Anwendung signiert anschließend die Sicherheitsdaten neu (Benutzer/Gruppen/Rechte).
3. Falls ein **DB-Key-Zertifikat** verwendet wird (`<db>.integrity.dbkey.json`), muss es nach einer Rotation **neu erzeugt** werden:
   - `pnpm -s integrity:certify --db-public <db>.integrity.pub.json --out <db>.integrity.dbkey.json`

Wichtig

Ein vorhandenes DB-Key-Zertifikat ist nach einer Rotation technisch ungültig (weil es den alten DB-Public-Key zertifiziert). Bis zur Neu-Zertifizierung behandelt FMB Log den Integritätsschutz als **nicht funktionsfähig** (Fail-Closed) und blockiert sicherheitsrelevante Vorgänge, bis ein neues Zertifikat vorliegt.

3.6 Wiederherstellung: Public-Key-Datei fehlt

Wenn `<db>.integrity.pub.json` gelöscht wurde:

1. **Falls ein Zertifikat vorhanden ist** (`<db>.integrity.dbkey.json`):
   - Der DB-Public-Key steckt im Zertifikat und kann wiederhergestellt werden:
   - `pnpm -s integrity:restore-pub --cert <db>.integrity.dbkey.json --out <db>.integrity.pub.json`
2. **Falls kein Zertifikat vorhanden ist**:
   - Ohne Zertifikat kann die Anwendung den DB-Key nicht gegen Root verifizieren.
   - FMB Log behandelt den Integritätsschutz dann als **nicht funktionsfähig** (Fail-Closed) und blockiert sicherheitsrelevante Vorgänge (z. B. Login/Benutzer-/Rechte-Sync).
   - Empfehlung: Zertifikat konsequent verwenden und die Dateien per ACL schützen.

Hinweis

Der Root-Private-Key (`.secrets/integrity-root-private.jwk.json`) dient ausschließlich dazu, den DB-Public-Key zu zertifizieren. Er kann den DB-Public-Key nicht "wiederherstellen", wenn dieser (und das Zertifikat) verloren gegangen sind.

3.7 Wiederherstellung / Re-Init (Root-Keys)

In der Praxis gibt es zwei unterschiedliche "Recovery"-Fälle. Entscheidend ist, **was** verloren ging.

3.7.1 A) Zertifikat verloren (`<db>.integrity.dbkey.json`)

Das ist der häufigste Fall (z. B. Datei versehentlich gelöscht). Vorgehen:

1. Stellen Sie sicher, dass `<db>.integrity.pub.json` noch neben der DB existiert.
2. Erzeugen Sie das Zertifikat auf dem Admin-Rechner (Root-Private-Key erforderlich):
   - `pnpm -s integrity:certify --db-public <db>.integrity.pub.json --out <db>.integrity.dbkey.json --root-private <pfad>`
3. Legen Sie `<db>.integrity.dbkey.json` wieder neben der DB ab und setzen Sie die ACLs (normale Nutzer read-only).
4. Starten Sie FMB Log neu.

Hinweis: In diesem Fall müssen die Sicherheitsdaten **nicht** neu signiert werden, weil sich der DB-Signierschlüssel nicht geändert hat. Optional kann in **Einstellungen → Integritätsschutz → Neu signieren** eine Voll-Signierung ausgeführt werden, um fehlende Signaturen zu ergänzen.

3.7.2 B) Integritäts-Vault/Signier-Passwort verloren (`vaults/<db>.integrity.vault`)

Wenn der private DB-Signierschlüssel nicht mehr verfügbar/entsperrbar ist, können vorhandene Signaturen nicht mehr erneuert werden. Das ist ein **Notfall-Szenario**:

1. Anwendung schließen.
2. DB und Vault-Ordner sichern (Backup/Copy).
3. Integritäts-Artefakte entfernen:
   - `vaults/<db>.integrity.vault`
   - `<db>.integrity.pub.json`
   - `<db>.integrity.dbkey.json` (falls vorhanden)
4. FMB Log starten und als Admin anmelden.
5. **Einstellungen → Integritätsschutz → Aktivieren** (neues Signier-Passwort festlegen).
6. Neues Zertifikat erzeugen (Root-Private-Key):
   - `pnpm -s integrity:certify --db-public <db>.integrity.pub.json --out <db>.integrity.dbkey.json --root-private <pfad>`
7. FMB Log neu starten (damit Integrität wieder als "zertifiziert" gilt).
8. **Einstellungen → Integritätsschutz → Entsperren** und anschließend **Neu signieren**, um Benutzer/-Gruppen/Rechte wieder konsistent zu signieren.

Wichtig

Eine Re-Initialisierung stellt nur den Schutz für die Zukunft wieder her. Wenn Integrität zuvor "kaputt" war (fehlendes Zertifikat/Vault), können Manipulationen in der Zwischenzeit nicht mehr kryptografisch nachgewiesen werden. Im Zweifel DB aus Backup wiederherstellen.

3.8 Lokales Schlüsselmaterial (`.secrets/`)

Im Repository wird nur der **Root Public Key** mit ausgeliefert (in der Anwendung fest eingebaut).
Alles andere ist lokales Admin-Material und bleibt bewusst **außerhalb von Git**.

- `.secrets/integrity-root-private.jwk.json`: Root-Private-Key zum Signieren von DB-Key-Zertifikaten (Build-/Admin-Tooling).
- `.secrets/integrity-db-private.jwk.json`: wird von FMB Log **nicht** verwendet (der DB-Private-Key liegt in `vaults/<db>.integrity.vault`). Diese Datei kann gelöscht werden, falls sie noch existiert.

Zusammenfassung

- **Passwörter:** Argon2id + Stronghold-Pepper pro Nutzer verhindert Hash-Swap und erschwert DB-basierte Account-Übernahmen.
- **Rechte/Administratoren:** Signaturen (Ed25519) verhindern unbemerkte Manipulationen; mit Zertifikat ist auch Public-Key-Swap abgefangen.
- **Dateiebene bleibt entscheidend:** Ohne passende ACLs sind DoS-Szenarien (Dateien löschen/überschreiben) in einem gemeinsamen Ordner grundsätzlich nicht vollständig verhinderbar.

3.9 Architekturdiagramm

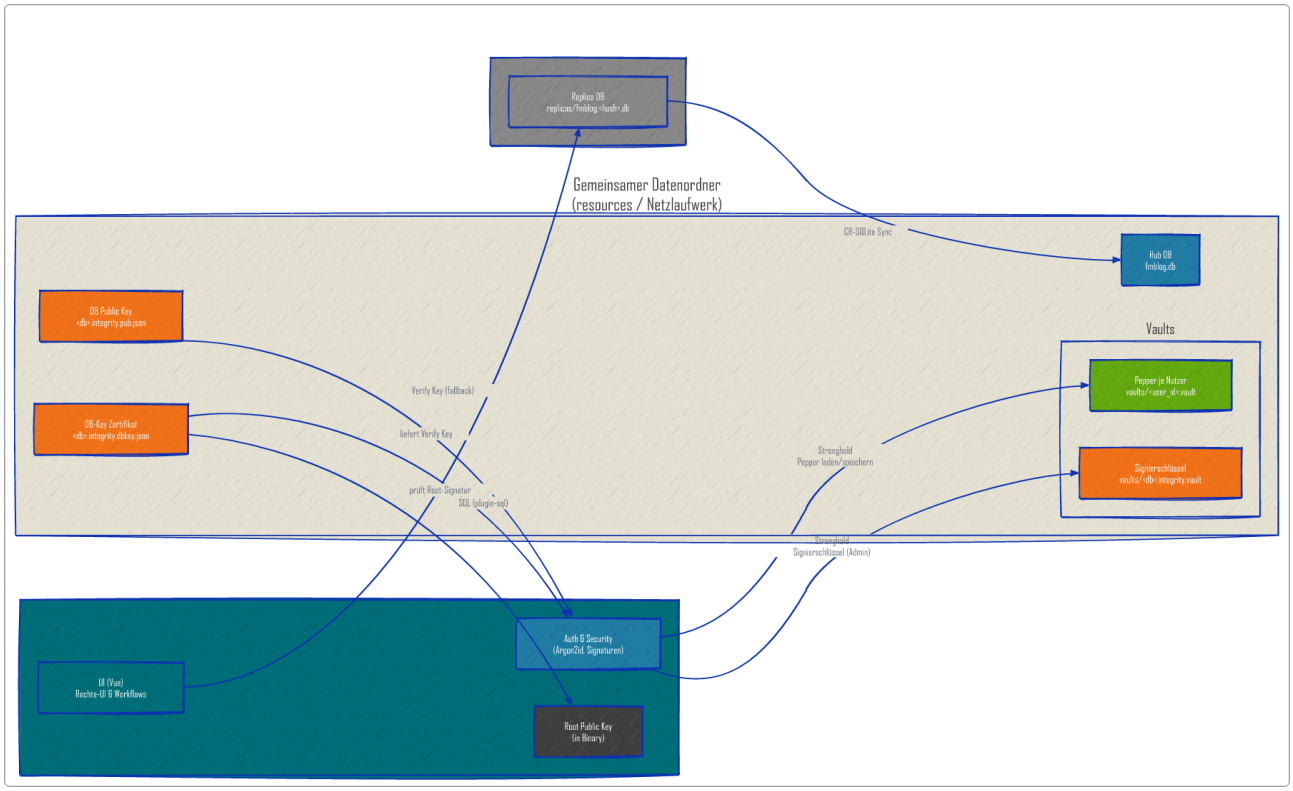

Figure 2: Sicherheitsarchitektur

3.10 Integrität für Messdaten & Tagesabrechnung (Stand)

Neben Benutzer/Gruppen/Rechten werden in FMB Log inzwischen auch **fachliche Daten** (Stammdaten, Mess-Revisionen, Protokolle) sowie die **Tagesabrechnung** selbst so abgesichert, dass Manipulationen an der SQLite-Datei **auffallen** (tamper-evident).

3.10.1 Bedrohungsszenarien (Beispiele)

- Messwerte manipulieren (z. B. `gamma_sum_og`, `iso_unit`, Messdatum), um Freigabefähigkeit zu erzwingen.
- FMK-/NV-/FGW-Stammdaten manipulieren (SW/KF/Freigabewerte), um Grenzwerte zu verschieben.
- Protokolldateien austauschen, sodass die Anzeige nicht mehr zu den gespeicherten Messwerten passt.
- Flags/Status verändern (z. B. „bereits abgerechnet“), um Nachvollziehbarkeit zu stören.

3.10.2 Zielbild

Für alle Daten, die in die Tagesabrechnung einfließen, gilt:

- **Jede Änderung ist kryptografisch nachweisbar** (Signatur/HMAC) und kann nicht durch reine DB-Manipulation „unsichtbar“ gemacht werden.
- **Offline-Import bleibt möglich** (lokale DB), Manipulationen fallen aber beim nächsten Sync/bei der Anzeige auf.
- Die Tagesabrechnung kann optional als **signierter Snapshot** erzeugt werden (Nachweis, welche Datenbasis verwendet wurde).

3.10.3 Umsetzung (phasenweise)

3.10.3.1 Phase 1: Protokoll-Integrität (Dateien) hart machen Die Protokolle liegen bereits als komprimierte Dateien im Hub-Ordner (lokal gecacht). Um Manipulationen eindeutig zu erkennen:

- In der DB wird pro Protokoll eine **BLAKE3** über den komprimierten Bytes gespeichert (`measurement_protocols.blak` 32-Byte-BLOB; Anzeige als Hex).
- Beim Laden/Anzeigen wird der Hash erneut berechnet und mit der DB verglichen.
- Der Protokoll-Hash wird zusätzlich in den signierten Datensatz der Mess-Revision aufgenommen (Phase 2), damit ein Austausch des Hashs in SQLite nicht genügt.

3.10.3.2 Phase 2: Messdaten (Revisionen) signieren – ohne Admin-Onlinepflicht Messungen werden typischerweise von „normalen“ Nutzern importiert. Dafür ist ein **per-User-Signaturschlüssel** sinnvoll:

- Bei der Kontoanlage wird zusätzlich zum Pepper ein **Ed25519-Keypair** erzeugt.
  - Private Key: im bestehenden `vaults/<user_id>.vault` (Stronghold), verschlüsselt durch `user_id + password`.
  - Public Key: in der DB, **zertifiziert** durch den DB-Integritätsschlüssel (damit Public-Key-Swap erkennbar ist).
- Beim Import/Ändern einer Mess-Revision wird eine kanonische Darstellung der fachlichen Felder gehasht (BLAKE3) und mit dem User-Key signiert.
- Bei der Anzeige und bei Tagesabrechnungs-Berechnungen wird diese Signatur geprüft. Ungültige/fehlende Signatur führt zu „ungültig“ (kein Freigabe-Nachweis).

Damit bleibt Offline-Import möglich (User-Key ist lokal verfügbar, weil im User-Vault).

3.10.3.3 Phase 3: Stammdaten (FGW/NV/FMK/SW/KF) signieren (Delegation) Stammdaten werden in FMB Log als **User-Signaturen mit Admin-Delegation** umgesetzt:

- Tabellen wie `fgw_values`, `nuclide_vectors`, `nuclide_vector_nuclides`, `fmks`, `fmk_year_settings` usw. tragen pro Datensatz eine **User-Signatur** + `capability_id`.
- Ein Admin erteilt dafür **Delegationen** (Capability-Zertifikate, DB-signiert). Dafür muss der DB-Signierschlüssel entsperrt sein.
- Key-User können danach Stammdaten pflegen, **ohne** Admin-Signier-Passwort zu kennen.
- Berechnungen (Dashboard/Preview/PDF) nutzen im Integritätsmodus nur **verifizierte** Stammdaten (Fail-Closed für Freigabe-Entscheidungen).

3.10.3.3.1 Erweiterung: Key-User ohne Admin-Signier-Passwort In der Praxis gibt es oft Nutzergruppen, die Stammdaten pflegen dürfen (FGW/NV/FMK/SW/KF), **ohne** Admin-Rechte zu besitzen und **ohne** das Signier-Passwort für Benutzer/Gruppen/Rechte zu kennen. Dafür braucht es eine **Trennung der Signier-Berechtigungen** („Separation of Duties“).

Es gibt zwei praktikable Muster:

1. **Separater Stammdaten-Signierschlüssel (Role Key)**
   - Eigener Ed25519-Key + eigenes Stronghold-Vault (z. B. `vaults/<db>.masterdata.vault`).
   - Entsperren erfolgt über ein **separates** Passwort, das nur die Stammdaten-Key-User kennen.

- Vorteil: technisch einfach.
- Nachteil: wenn ein Key-User aus der Rolle entfernt wird, muss das Passwort/der Key **rotiert** werden (weil Wissen nicht „zurückgenommen“ werden kann).

2. **Per-User-Signaturen + Admin-Delegation (Capability-Zertifikate) (empfohlen)**
   - Jeder Nutzer besitzt einen **eigenen** Ed25519-Key (Private Key in `vaults/<user_id>.vault`).
   - Ein Admin erteilt die Berechtigung zum Signieren von Stammdaten über ein **Delegations-Zertifikat**, das mit dem DB-Integritätsschlüssel signiert ist (Root-Trust via `<db>.integrity.dbkey.json`).
   - Ein Stammdatensatz wird dann signiert mit:
     - `signed_by_user_id` + `user_signature`
     - optional: `capability_id` (Delegations-Zertifikat)
   - Verifikation:
     1) User-Signatur ist gültig (Public Key ist in der DB hinterlegt und selbst signiert/geschützt).
     2) Delegations-Zertifikat ist gültig und umfasst den Scope (z. B. `masterdata.fgw`, `masterdata.nv`, …).
     3) Optional: Delegation ist nicht widerrufen / Signing-Zeitpunkt liegt vor `revoked_at`.
   - Vorteile:
     - Kein Shared Secret unter Key-Usern.
     - **Bessere Nachvollziehbarkeit** („wer hat unterschrieben“).
     - Entzug der Berechtigung ist über Widerruf/Expiry möglich, ohne sofortige Rotation eines Shared Keys.
   - Nachteil: etwas mehr Implementierungsaufwand (Zertifikate/Widerruf).

3.10.3.3.2 Umsetzung in FMB Log (aktuell) Scopes (Delegations-Bereiche):

- `masterdata.fgw`: Freigabewerte (FGW)
- `masterdata.nv`: Nuklidvektoren (NV)
- `masterdata.fmk`: Freimesskampagnen inkl. Pfad-Reihenfolge und SW/KF (FMK/SW/KF)

Ablauf: Admin delegiert Signierrechte an Key-User

1. Integritätsschutz ist aktiviert und das DB-Key-Zertifikat liegt neben der DB (`<db>.integrity.dbkey.json`).
2. Admin entsperrt den DB-Signierschlüssel einmalig (Administration → Einstellungen → Integritätsschutz → „Entsperren“).
3. Admin vergibt Delegationen (Administration → Einstellungen → Delegationen):
   - Nutzer auswählen
   - Scope(s) erteilen (FGW/NV/FMK)
   - Optional: Ablaufdatum (Expiry) setzen
4. Die Delegation wird als Datensatz in `capability_certs` abgelegt und mit dem DB-Signierschlüssel signiert.

Ablauf: Key-User ändert Stammdaten

1. Key-User meldet sich an.
2. Beim Speichern von FGW/NV/FMK wird der Datensatz mit dem **User-Key** signiert und die zugehörige Delegation (`capability_id`) mitgeschrieben.
3. Die Anwendung nutzt für Berechnungen (insb. Tagesabrechnung/Preview/Dashboard) nur **verifizierte** Stammdaten:
   - User-Signatur gültig
   - User-Public-Key ist DB-signiert (Public-Key-Swap erkennbar)
   - Delegation gültig zum Signaturzeitpunkt (`issued_at`/`expires_at`/`revoked_at`)

Widerruf / Entzug

- Admin kann Delegationen widerrufen. Signaturen, die **nach** `revoked_at` erstellt wurden, gelten als ungültig.
- Signaturen, die **vor** `revoked_at` erstellt wurden, bleiben verifizierbar (Audit-Trail).

3.10.3.4 Phase 4: Tagesabrechnung als signierter Snapshot

Zusätzlich zur laufenden Daten-Integrität wird beim PDF-Export ein **Snapshot-Hash** erzeugt und in der PDF als **QR-Code** ausgegeben.

Umsetzung in FMB Log (aktuell):

- Beim Export wird ein kanonischer Snapshot der tatsächlich exportierten Tabellenzeilen erstellt und als **BLAKE3** gehasht.
- Der Hash wird in `daily_reports.snapshot_hash` gespeichert (DATA-Hash; 32-Byte-BLOB, Anzeige als Hex).
- Die PDF enthält unten rechts in der Fußzeile einen QR-Code mit diesem Snapshot-Hash (und optional weiteren Metadaten, siehe unten).
- Zusätzlich werden neben dem QR-Code kurze **Fingerprints** angezeigt (z. B. `DATA: ABC-DEF-GHI`), damit Werte auch ohne Abtippen langer Hashes manuell gegengeprüft werden können.
- Zusätzlich wird der **PDF-Hash** (BLAKE3 der finalen PDF-Bytes; Spaltenname historisch `daily_reports.pdf_sha256` in der Historie gespeichert. In der Historie kann das Original-PDF über **PDF prüfen...** gegen diesen Hash geprüft werden.
- Wird eine enthaltene Mess-Revision später **ungültig gesetzt** oder wird eine Tagesabrechnung manuell ungültig gemacht (`reports.invalidate`), wird eine **signierte Invalidierung** angelegt (`daily_report_invalidations`, User-Key) und die Tagesabrechnung als **ungültig** markiert (`daily_reports.is_valid = 0`). Dabei werden Export-Markierungen der enthaltenen Messungen zurückgesetzt, sodass eine erneute Abrechnung möglich bleibt.

3.10.3.4.1 Optional: RFC3161-Zeitstempel (FreeTSA) Gerade bei Tagesabrechnungen kann zusätzlich ein RFC3161-Zeitstempel hinterlegt werden (TSA-Signatur über den Snapshot-Hash). Das ist besonders hilfreich, wenn eine Tagesabrechnung **nachweisbar zu einem Zeitpunkt** existiert haben soll.

- Aktivierung durch Admin: `Administration → Einstellungen → Tagesabrechnung → RFC3161-Zeitstempel verpflichtend`
- Verhalten:
  - Ist die Option aktiv, wird beim PDF-Export ein RFC3161-Timestamp bei `https://freetsa.org/tsr` angefordert.
  - Ohne Internetverbindung schlägt der Export fehl (Fail-Closed), damit keine „ungetimestampte“ Abrechnung entsteht.
- Speicherung in der Historie (`daily_reports`):
  - `tsa_provider`
  - `tsa_gen_time`
  - `tsa_snapshot_sha256` (der für TSA verwendete SHA-256-Imprint; siehe Hinweis)
  - `tsa_token_sha256`
  - `tsa_token_base64` (DER-Token als Base64)
- QR-Code in der PDF:
  - Enthält immer `snapshot_hash`
  - Wenn TSA aktiv: zusätzlich `tsa_token_sha256` (das Token selbst ist zu groß für QR und liegt in der DB-Historie)

In der **Historie** kann FMB Log den TSA-Token zusätzlich technisch prüfen („plausibel“):

- Token lässt sich parsen (TSTInfo)
- Hash-Algorithmus ist SHA-256
- Imprint im Token passt zu `tsa_snapshot_sha256`

Hinweis: Das ist keine vollständige PKI-Vertrauensprüfung der TSA-Signaturkette, erhöht aber die Konsistenzprüfung (Token/Imprint gehören zusammen).

Hinweis (warum TSA weiterhin SHA-256 nutzt)

FreeTSA/RFC3161 erwartet einen Standard-Hash (hier SHA-256) als „Message Imprint“.
Daher wird für TSA zusätzlich `tsa_snapshot_sha256` gespeichert. Der primäre DATA-Hash (`daily_reports.snapshot_hash`) bleibt BLAKE3.

3.10.4 Hinweise zur CRDT-Synchronisation (CR-SQLite)

Die Signaturen sind normale Spalten/Datensätze und werden über CR-SQLite repliziert. Wichtig ist:

- Primärschlüssel müssen non-NULL sein; Signatur-Spalten sollten **nullable** sein (Initialzustand kompatibel).
- Bei Konflikten („Last-writer-wins“) ist die Signatur nur für den finalen Datensatz gültig – das ist gewollt: ein Konflikt ohne passende Signatur wird als „ungültig“ sichtbar.

3.11 Audit (Administration → Audit)

FMB Log enthält eine Audit-Ansicht, mit der ein Admin den aktuellen Integritätsstatus prüfen kann.

Der Audit umfasst (je nach Einstellung) u. a.:

- **Security-Tabellen** (DB-Signatur/Integrität): Benutzer, Gruppen, Gruppenrechte, Delegations-Zertifikate, Admin-Einstellungen
- **Stammdaten** (User-Signaturen + Delegation): FGW/NV/FMK inkl. SW/KF-Tabellen
- **Messdaten** (User-Signaturen): Mess-Revisionen
- **Protokoll-Integrität**: BLAKE3-Prüfung der komprimierten Protokoll-Bytes (Hub-Packfiles)

Empfehlung: Audit nach Updates, nach Key-Rotation und bei Verdacht auf Datenmanipulation ausführen.

3.12 Audit-Trail Export (Administration → Audit-Trail Export)

Neben dem kryptografischen Audit gibt es einen **Audit-Trail** als „Betriebsprotokoll“: Die Anwendung schreibt wichtige Aktionen (z. B. aus der Administration und aus den Stammdaten) als Ereignisse in die Tabelle `audit_trail`. Diese Einträge sind für externe Prüfungen gedacht und können als CSV exportiert werden.

Der Audit-Trail ist **kein** Ersatz für den Audit-Check (Signatur-/Hash-Verifikation), sondern ergänzt ihn:

- **Audit-Trail** beantwortet: *Wer hat wann was geändert?* (Change-Log / Nachvollziehbarkeit)
- **Audit** beantwortet: *Sind die aktuellen Daten unverändert und verifizierbar?* (Integritätsprüfung)

3.12.1 Export nutzen

1. Öffnen Sie **Administration → Audit-Trail Export**.
2. Wählen Sie optional einen Zeitraum („Von/Bis“) und/oder nutzen Sie die Suche.
3. Klicken Sie auf **CSV exportieren** und speichern Sie die Datei.

Die CSV enthält u. a. `event_at`, `user_id`, `username`, `action`, `entity_table`, `entity_id` sowie `before_json`/`after_json` (Vorher/Nachher als JSON-Snapshot).

3.12.2 Typische Aktionen (Beispiele)

- Benutzerverwaltung: `users.create`, `users.update`, `users.delete`, `users.reset_password`
- Gruppen/Rechte: `groups.create`, `groups.update`, `groups.delete`, `group_permissions.update`, `user_groups.update`
- Stammdaten: `fgw.update`, `fmks.create/update/delete`, `nuclide_vectors.create/update/delete`
- Tagesabrechnung: `daily_reports.invalidate`

Hinweis

Sensible Geheimnisse (z. B. temporäre Passwörter) werden **nicht** im Audit-Trail gespeichert. Es werden nur die fachlich relevanten Metadaten und Vorher/Nachher-Snapshots protokolliert.

4 Benutzer

Die Benutzerverwaltung befindet sich in der App unter **Administration → Benutzer**. Hier verwalten Sie Accounts und steuern, wer sich anmelden darf. Rechte werden nicht direkt am Nutzer vergeben (außer „Admin“), sondern über Gruppen.

4.1 Überblick

Ein Benutzer kann:

- **aktiv** oder **inaktiv** sein (nur aktive Benutzer können sich anmelden),
- **Administrator** sein (volle Rechte) oder
- Rechte über **Gruppen** erhalten (empfohlenes Rollenmodell).

Inaktivieren statt Löschen ist oft sinnvoll, weil dadurch die Historie (wer hat wann welche Messungen importiert) nachvollziehbar bleibt.

Empfehlung

- Nutzen Sie Gruppen als Rollen (z. B. „Messung“, „Auswertung“, „Key-User“) und vergeben Sie Rechte ausschließlich über Gruppen.

4.2 Benutzer anlegen

Zum Anlegen eines neuen Nutzers geben Sie Benutzername, Anzeigename und ein initiales Passwort ein. Optional können Sie den Nutzer direkt als Admin markieren (nur in Ausnahmefällen; üblich ist die Rechtevergabe über Gruppen).

Nach dem Anlegen kann sich der Nutzer (sofern aktiv) sofort anmelden und sein eigenes Passwort ändern.

4.3 Benutzer bearbeiten

Wählen Sie in der Tabelle einen Benutzer aus. In der Detailansicht können Sie:

- Anzeigename ändern
- Benutzer aktivieren/deaktivieren
- Admin-Status setzen/entfernen

Hinweis: Mindestens ein **aktiver Admin** muss verbleiben.

4.4 Gruppen zuweisen

Weisen Sie dem Benutzer die passenden Gruppen zu. Die wirksamen Rechte ergeben sich aus der Gruppen-Rechte-Matrix (siehe Gruppen & Rechte). Änderungen wirken sofort auf die UI (nach dem nächsten Laden der Seite bzw. nach erneutem Login).

4.5 Passwort zurücksetzen

Admins können in der Detailansicht ein neues Passwort für den ausgewählten Benutzer setzen (**Passwort neu setzen**). Zusätzlich existiert die Seite **Administration → Passwörter zurücksetzen** für Nutzer mit dem Recht `users.reset_passwords`. Das ist hilfreich, wenn ein Key-User Passwörter verwalten darf, aber kein Voll-Admin sein soll.

4.6 Benutzer löschen

In der Detailansicht können Benutzer gelöscht werden. In vielen Organisationen ist jedoch „Inaktivieren“ die bessere Wahl, weil die Historie nachvollziehbar bleibt.

Hinweis: Der eigene Account kann nicht gelöscht werden.

Zusammenfassung

- Nutzer sind aktiv/inaktiv und optional Admin.
- Rechte werden über Gruppen vergeben (best practice).
- Passwort-Resets sind eine separate, sensible Berechtigung.

5 Gruppen & Rechte

FMB Log nutzt ein gruppenbasiertes Berechtigungsmodell: Sie definieren **Gruppen** (Rollen) und vergeben Rechte in einer Matrix. Benutzer erhalten Rechte, indem sie einer oder mehreren Gruppen zugeordnet werden. Dieses Vorgehen ist im Alltag deutlich wartbarer als individuelle Rechte pro Nutzer, weil Rollen (z. B. „Messung“, „Auswertung“, „Key-User“) konsistent abgebildet werden können.

Die Gruppen- und Rechteverwaltung befindet sich in der App unter **Administration → Gruppen & Rechte**.

5.1 Rollenmodell

Ein Benutzer kann als **Admin** markiert werden. Admins besitzen immer alle Rechte und sehen zusätzlich die Administrationsseiten. Für alle anderen gilt: Rechte kommen ausschließlich aus den zugewiesenen **Gruppen**.

Das hat einen praktischen Vorteil: Wenn eine Funktion fehlt (oder zu viel sichtbar ist), lässt sich das fast immer auf die Gruppenzuordnung bzw. die Gruppen-Matrix zurückführen.

5.2 Gruppen verwalten

Gruppen können angelegt, umbenannt, deaktiviert/aktiviert und gelöscht werden. Nur **aktive** Gruppen werden bei der Berechnung der wirksamen Rechte berücksichtigt. In der Praxis ist „Deaktivieren“ häufig die bessere Wahl als „Löschen“, weil sich so die Historie nachvollziehen lässt und die Gruppe später bei Bedarf wieder aktiviert werden kann.

5.3 Rechte (Matrix)

Die Rechte werden pro Gruppe in einer Matrix vergeben. Die UI blendet Funktionen entsprechend ein/aus; zusätzlich werden sensible Aktionen auch serverseitig geprüft.

Typische Beispiele für Rechte sind:

- Messungen importieren/ändern/löschen sowie **Messdatum ändern**
- Tagesabrechnungen ungültig machen (`reports.invalidate`)
- FMK und NV anlegen/ändern/löschen
- Freigabewerte ändern (`fgw.update`)
- Passwörter anderer Nutzer zurücksetzen (`users.reset_passwords`)

Die vollständige Liste der Permission Keys befindet sich in der Referenz: Rechte.

Zusammenfassung (Gruppen & Rechte)

- Rechte werden über Gruppen vergeben; Admins haben immer alles.
- „Deaktivieren“ ist oft sinnvoller als „Löschen“ (Historie, Wiederverwendbarkeit).
- Kritische Rechte (FGW ändern, Passwörter zurücksetzen, Messdatum ändern, Tagesabrechnungen ungültig machen) nur gezielt vergeben.

6 Freigabewerte (FGW)

Freigabewerte (FGW) sind die nuklidspezifischen Grenzwerte, gegen die FMB Log gemessene Aktivitäten prüft. Sie sind zentral für die Berechnung von Freigabewerten eines Nuklidvektors und für die Freigabeprüfung in Vorschau und PDF der Tagesabrechnung.

Die FGW werden bewusst **lokal in der SQLite-Datenbank** gespeichert. Dadurch ist die Anwendung nicht von externen Dateien oder Datenbankservern abhängig, und Änderungen können in denselben Prozess (Berechtigungen, Backup, Freigabe) eingebettet werden wie alle anderen Stammdaten.

6.1 Zugriff und Berechtigung

Die Seite **Freigabewerte** ist nur sichtbar, wenn Sie Administrator sind oder die Permission `fgw.update` besitzen. So kann ein Key-User FGW pflegen, ohne Voll-Admin zu sein.

6.2 Einheiten und Struktur

FGW werden pro Nuklid und Freigabepfad gespeichert. Die Einheit hängt vom Pfad ab:

- Oberflächenspezifische Pfade (z. B. OF/3a/4a/5b) in Bq/cm^2
- Alle übrigen Pfade in Bq/g

6.3 Änderungen im Betrieb

Änderungen an FGW wirken sofort auf Berechnungen und Anzeigen. Aus fachlicher Sicht empfiehlt sich daher ein klarer Freigabeprozess (z. B. Vier-Augen-Prinzip) und ein Backup, bevor größere Änderungen vorgenommen werden. So bleiben Tagesabrechnungen auch später nachvollziehbar.

Zusammenfassung (FGW)

- Zugriff nur als Admin oder mit `fgw.update`.
- Einheiten: OF/3a/4a/5b in Bq/cm^2, sonst in Bq/g.
- Änderungen wirken sofort – daher Freigabeprozess und Backups nutzen.

7 Betrieb & Datenbank

FMB Log speichert alle Daten in einer einzelnen SQLite-Datei. Dadurch ist kein Datenbankserver notwendig und die Anwendung kann auch offline betrieben werden.

7.1 Standard-Datenbankpfad

Standardmäßig liegt die Datenbank im Programmordner unter `resources/fmblog.db`. Beim ersten Start legt die Anwendung das Unterverzeichnis `resources` an (falls es fehlt) und erstellt die Datenbankdatei beim Öffnen automatisch.

Dieser Standard ist bewusst gewählt: Die Datenbank ist damit leicht auffindbar, einfach zu sichern und kann (sofern gewünscht) auch auf einen gemeinsam genutzten Pfad gelegt werden.

7.2 Stub-DB und Updates

Für Neuinstallationen oder Updates kann eine **Stub-Datenbank** mitgeliefert werden. Beim ersten Start wird dann (falls noch keine Datenbank existiert) `resources/fmblog.stub.db` aus den gebündelten Ressourcen nach `resources/fmblog.db` kopiert.

Wichtig: Eine vorhandene Datenbank wird dabei **niemals überschrieben**. Damit bleiben Bestandsdaten bei Updates erhalten.

7.3 Datenmodell (ER)

Eine vereinfachte Darstellung des SQL-Schemas als ER-Diagramm finden Sie hier:

7.4 Netzlaufwerk und Mehrbenutzer

SQLite kann grundsätzlich auf Netzlaufwerken funktionieren, allerdings hängt die Stabilität stark von Dateisperren/Locking, SMB-Konfiguration (z. B. OpLocks) und der Latenz/Zuverlässigkeit des Shares ab. In Mehrbenutzer-Szenarien kommt es typischerweise nicht auf „viele Leser“, sondern auf **gleichzeitige Schreibzugriffe** an.

Empfehlung: Validieren Sie den Mehrbenutzerbetrieb zunächst in einer Testumgebung und definieren Sie organisatorisch, wer wann Änderungen schreibt (z. B. Import zentral, Auswertung verteilt). Wenn regelmäßig Sperren oder Zeitüberschreitungen auftreten, ist ein lokaler DB-Pfad pro Nutzer die robustere Variante.

7.5 Synchronisation (Hub-DB und lokale Replica)

Für den Mehrnutzerbetrieb nutzt FMB Log eine **lokale Replica-DB** (pro Windows-Benutzer) und synchronisiert diese regelmäßig mit einer **Hub-DB** (z. B. im gemeinsamen Datenordner / Netzlaufwerk):

- **Hub-DB:** entspricht dem in den Einstellungen gesetzten DB-Pfad (oder dem Standard unter `resources/fmblog.db`)
- **Lokale Replica:** wird im Benutzerprofil unter `AppData/Local/<app-id>/replicas/…` angelegt
- **Sync:** läuft automatisch (z. B. beim Fokuswechsel und alle ~10 s)

7.5.1 Warum lokale Replica + CRDT?

Ein reiner SQLite-Mehrbenutzerbetrieb über ein Netzlaufwerk ist oft fragil (Dateisperren, OpLocks, Latenz). Die Replica-Architektur reduziert dieses Risiko deutlich:

- **Offline-fähig:** Die Anwendung bleibt auch bei temporär nicht verfügbarem Hub nutzbar (Arbeit in der lokalen Replica).
- **Weniger Lock-Druck auf dem Share:** Nutzer schreiben hauptsächlich lokal; der Hub wird kurz/periodisch synchronisiert.
- **Konfliktbehandlung:** Gleichzeitige Änderungen können deterministisch zusammengeführt werden (CRDT/CR-SQLite).

7.5.2 Wie funktioniert CR-SQLite (CRR) in FMB Log?

FMB Log verwendet CR-SQLite, um Tabellen als **CRR (Conflict-free Replicated Relations)** zu betreiben. Vereinfacht:

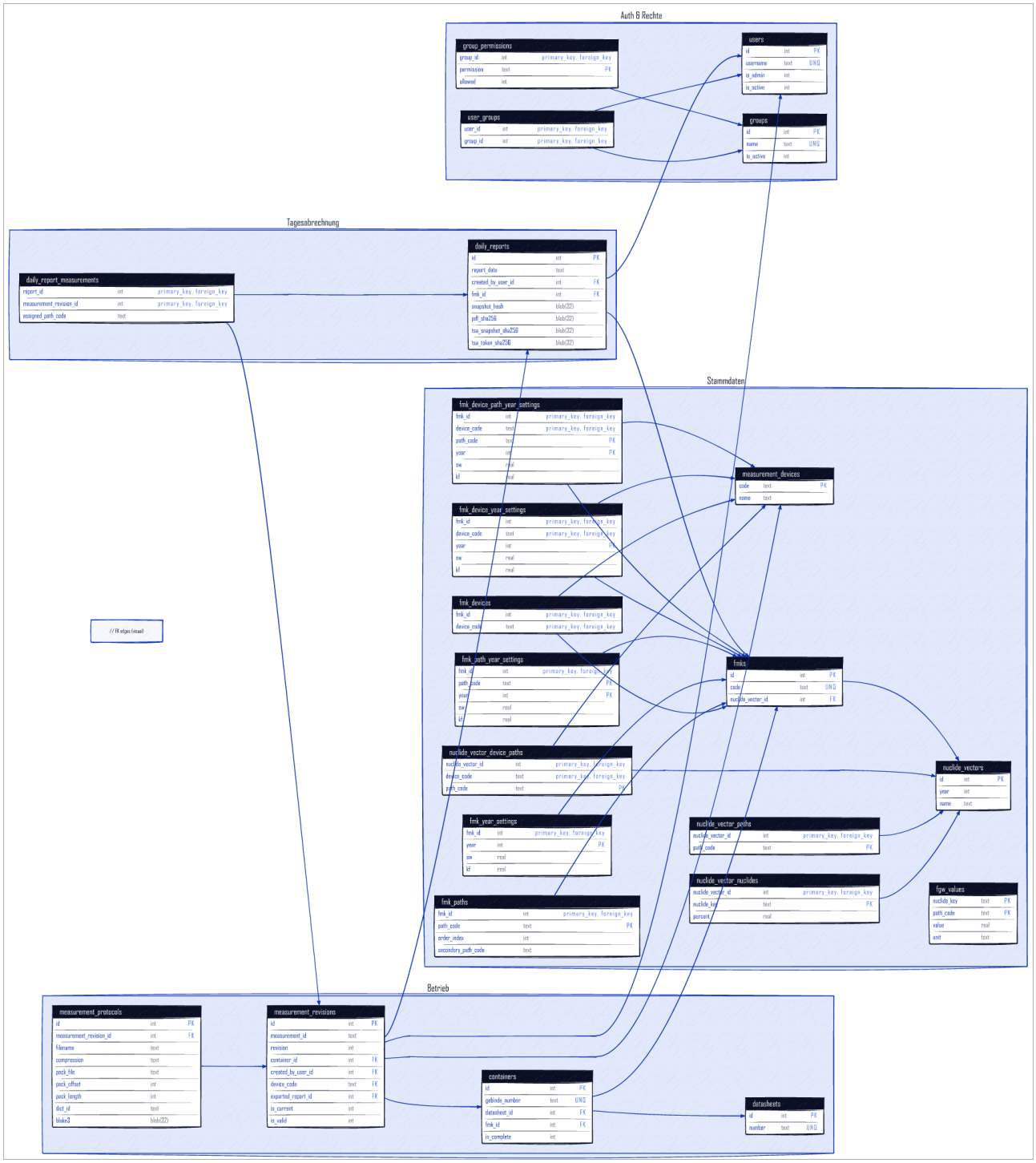

Figure 3: SQL-Schema (ER)

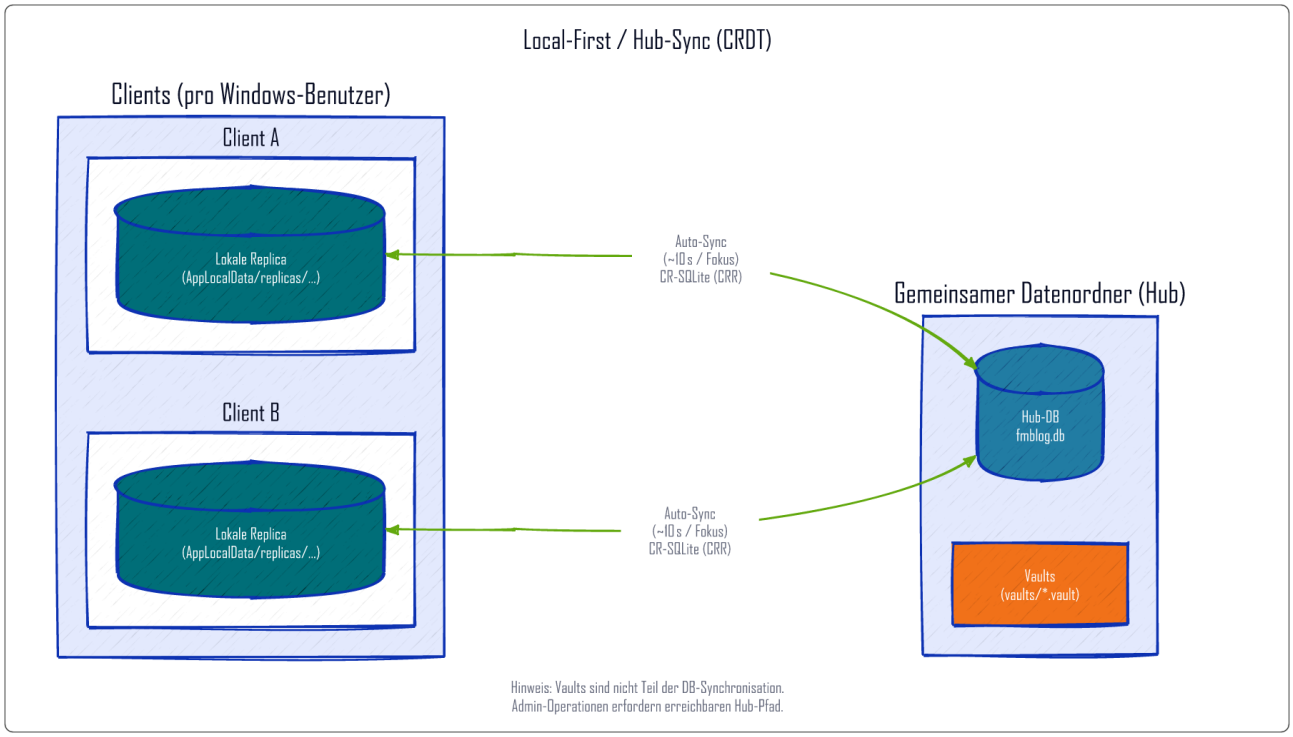

Figure 4: Local-First / CRDT (Übersicht)

- Jede DB-Instanz hat eine eigene **Site-ID**.
- Änderungen werden von CR-SQLite als **Change-Events** erfasst (`crsql_changes`).
- Beim Sync werden neue Change-Events zwischen lokaler Replica und Hub ausgetauscht und angewendet.

Wichtig: Das Ergebnis ist **eventual consistency**. Nach erfolgreichem Sync konvergieren alle Replicas (deterministisch) auf denselben Zustand – ohne dass die Anwendung dazu File-Locks "manuell" koordinieren muss.

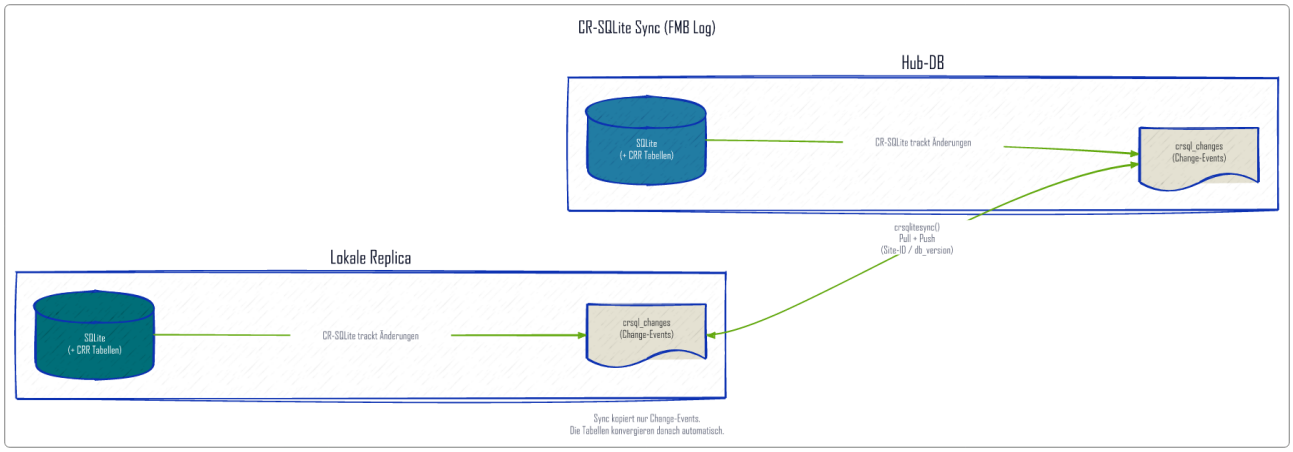

Figure 5: cr-sqlite Architektur

7.5.3 Sync-Ablauf (automatisch)

Der Sync läuft automatisch und "best effort":

- Beim Start wird ein initialer Sync versucht.
- Zusätzlich bei Fokuswechsel (zurück zur App) und periodisch alle ~10 s.
- Technisch werden erst **Änderungen aus dem Hub in die lokale Replica gezogen** (Pull) und danach **lokale Änderungen in den Hub geschrieben** (Push).

7.5.4 Einschränkungen (wichtig)

Auch mit CRDT/CR-SQLite gibt es Grenzen und "Betriebsregeln":

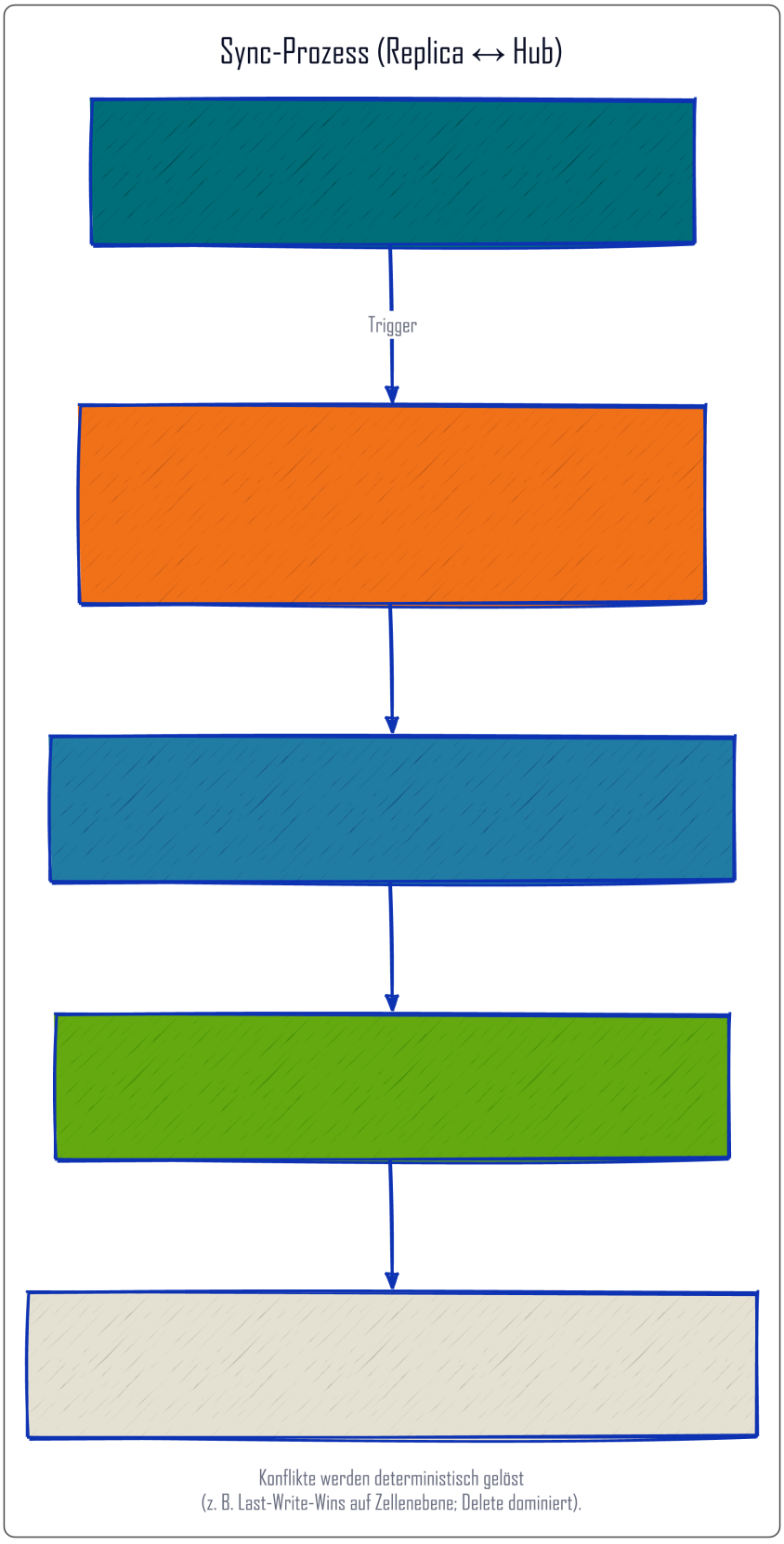

Figure 6: Sync-Prozess

1. **Konflikte sind möglich (und werden aufgelöst)**
   Wenn zwei Nutzer *dieselbe* Entität gleichzeitig bearbeiten, greift eine deterministische Merge-Regel. In der Praxis bedeutet das oft "Last-Write-Wins" auf Feld-/Zellenebene.
   Empfehlung: Kritische Stammdaten (FGW/NV/FMK) organisatorisch nur durch wenige Key-User ändern lassen.

2. **Schema-Constraints für CRR-Tabellen**
   CR-SQLite stellt Anforderungen an das Schema (u. a. nicht-nullable Primary Keys; keine `NOT NULL`-Spalten ohne Default-Wert).
   Daraus folgt: IDs sind in FMB Log als **stabile, eindeutige TEXT-IDs** ausgelegt (statt Auto-Increment), und Schema-Änderungen erfolgen über App-Migrations.

3. **Netzwerk-/Share-Qualität bleibt entscheidend**
   Der Hub ist weiterhin eine SQLite-Datei. Wenn das Share instabil ist (Locking, hohe Latenz), kann es trotz Replica-Ansatz zu Sync-Fehlern kommen. Die App bleibt dann lokal nutzbar, aber Änderungen können erst später verteilt werden.

4. **Große BLOBs kosten Sync-Zeit**
   Messprotokolle werden bewusst **nicht** als BLOB in der CRDT-SQLite synchronisiert. Stattdessen werden sie zstd-komprimiert in einem Protokoll-Archiv **neben der Hub-DB** gespeichert und in der DB nur referenziert (BLAKE3, Archivdatei, Offset/Länge, optional `dict_id`).
   Ergebnis: Der CR-SQLite-Sync bleibt schnell, weil er nur Metadaten synchronisiert.

   Ablage im Hub-Ordner (neben `<db>.db`):

   - `protocols/<site-id-hex>/pack-*.bin` (Packfiles, append-only; max. ~100 Einträge oder ~1 MB pro Pack)
   - `protocols/<site-id-hex>/state.json` (merkt sich das aktuelle Packfile)
   - DB-Referenzen in `measurement_protocols` (`pack_file`, `pack_offset`, `pack_length`, `blake3`, `dict_id`, …)
   - Hinweis: Die Spalte `sha256` existiert nur für Abwärtskompatibilität und ist für neue Imports i. d. R. leer (maßgeblich ist `blake3`).

   Offline-Import: Wenn der Hub nicht erreichbar ist, wird das Protokoll im **lokalen Cache** abgelegt und die DB enthält zunächst eine „ausstehende" Referenz (`pack_file=''`, `pack_length=0`). Zusätzlich wird lokal eine Outbox geführt (`protocol_upload_outbox`), die Upload-Versuche, Backoff und Fehlerstatus verwaltet (Status ist auf der Import-Seite in „Ausstehende Uploads" sichtbar). Nach dem nächsten erfolgreichen Sync lädt der importierende Client das Protokoll automatisch in das Hub-Archiv hoch und aktualisiert die Referenz. Bis dahin können andere Clients die Messung sehen, das Protokoll aber noch nicht öffnen.

7.5.5 Besonderheit: Vault-Dateien sind nicht Teil der DB-Synchronisation

Passwörter und Signierschlüssel werden zusätzlich in **Stronghold-Vaults** gespeichert:

- Nutzer-Vaults: `vaults/<user_id>.vault`
- Integritäts-Vault: `vaults/<db>.integrity.vault`

Diese Dateien liegen **neben der Hub-DB** im gemeinsamen Datenordner und werden **nicht** über die DB-Synchronisation repliziert. Daher gilt:

- Kritische Admin-Operationen (Benutzer-/Gruppen-/Rechteverwaltung, Integritätsschlüssel) sollten nur durchgeführt werden, wenn der Hub-Pfad erreichbar ist.
- Wenn der Hub-Pfad nicht erreichbar ist, sind diese Aktionen in der UI deaktiviert (die Fach-/Messdaten können weiterhin über die lokale Replica bearbeitet werden).

7.6 Backup

Die sicherste Backup-Strategie ist, die relevanten Dateien zu kopieren, wenn die Anwendung geschlossen ist. Für den Betrieb empfiehlt sich eine regelmäßige Sicherung (z. B. täglich) und eine Aufbewahrung mehrerer Generationen (Rotation), damit auch ältere Tagesabrechnungen nachvollziehbar bleiben.

Sichern Sie im Hub-Ordner mindestens:

- die DB-Datei (`<db>.db` bzw. `fmblog.db`)
- den Ordner `protocols/` (Protokoll-Archive/Packfiles)
- den Ordner `vaults/` (Stronghold-Vaults für Pepper/Integritätsschlüssel)

Zusammenfassung (Datenbank)

- Eine Datei: SQLite `resources/fmblog.db` (Standardpfad).
- Optional: Stub-DB `resources/fmblog.stub.db` wird nur kopiert, wenn noch keine DB existiert.
- Netzlaufwerke sind möglich, aber lock-/latency-abhängig → vorher testen.
- Mehrnutzerbetrieb: lokale Replica + CR-SQLite Sync (eventual consistency).
- Protokolle liegen als Packfiles im Hub-Ordner (`protocols/…`), nicht als BLOB in der DB.
- Backups am besten bei geschlossener Anwendung.

8 Technische Dokumentation

Dieses Kapitel ergänzt den Admin Guide um technische Details für Betrieb, IT-Integration und Fehlersuche. Es beschreibt, **wo** FMB Log Daten ablegt, **welche** Dateien „außerhalb der Datenbank“ existieren und **wie** Signaturen/Authentifizierung technisch funktionieren.

> Zielgruppe: Admins/Key-User und IT-Betreiber (z. B. wenn FMB Log auf mehreren Arbeitsplätzen gegen einen gemeinsamen Hub betrieben wird).

8.1 1) Datenablage: `resources` vs. Hub-Datenordner vs. AppLocalData

FMB Log unterscheidet drei Speicherbereiche:

1. **Programm-Ressourcen (`resources/` im Programmordner)**
   Enthält gebündelte Dateien, die zur App gehören (Extensions, Dictionaries, Logo, Stub-DB).
2. **Hub-Datenordner (gemeinsam, z. B. Netzlaufwerk)**
   Enthält die **Hub-DB** sowie **Vaults** und **Protokoll-Archive**. Dieser Ordner ist die „Quelle der Wahrheit“ für den gemeinsamen Betrieb.
3. **AppLocalData (pro Windows-Benutzer, lokal)**
   Enthält die **lokale Replica-DB** und den **lokalen Protokoll-Cache**, damit die App offline weiter nutzbar bleibt.

Typische Struktur (vereinfacht):

- **Programmordner** (lokal)
  - `resources/`
    - `extensions/crsqlite.dll` (CR-SQLite Extension)
    - `zstd-dicts/rpt-v1.dict` (optional, Zstd-Dictionary für RPT-Kompression)
    - `Logo_EWN_RGB.png` (Logo für Tagesabrechnung)
    - `fmblog.stub.db` (Stub-DB als „Baseline“; wird nur kopiert, wenn noch keine DB existiert)
- **Hub-Datenordner** (gemeinsam)
  - `<db>.db` (Hub-DB; Name frei wählbar)
  - `<db>.integrity.pub.json` (DB-Public-Key)
  - `<db>.integrity.dbkey.json` (DB-Key-Zertifikat; Root-signiert)
  - `vaults/`
    - `<user_id>.vault` (Pepper + User-Signierschlüssel)
    - `<db>.integrity.vault` (DB-Signierschlüssel; Admin-Entsperren)
  - `protocols/`
    - `<site-id-hex>/pack-*.bin` (Packfiles für komprimierte Protokolle)
    - `<site-id-hex>/state.json` (merkt „aktuelles“ Packfile)
- **AppLocalData** (lokal)
  - `replicas/fmblog.<hash>.db` (Replica-DB)
  - `protocols-cache/<hub-hash>/` (lokaler Protokoll-Cache; Dateien nach Protokoll-BLAKE3 benannt)

Zusammenfassung (Speicherorte)

- `resources/` enthält „App-Ressourcen“ (Extension/Dictionaries/Logo/Stub), nicht die produktiven Daten.
- Produktive Daten liegen im Hub-Datenordner: Hub-DB + `vaults/` + `protocols/` + Integritätsdateien.
- AppLocalData ist pro Nutzer lokal: Replica-DB + Cache (offline-fähig).

Stub-DB aktualisieren (bei Schema-Änderungen)

Die Stub-DB ist die „Baseline“ für neue DB-Dateien (Hub/Replica). Wenn sich das Schema durch neue Migrationen ändert, sollte die Stub-DB aktualisiert werden:
`pnpm -s db:stub:update`
Wichtig: Das Skript muss auf **Windows** ausgeführt werden (nicht in WSL), da es die lokal installierten Build-Tools/Abhängigkeiten nutzt.

8.2 2) Messprotokolle: zstd-Kompression, Packfiles und lokaler Cache

Messprotokolle werden aus Performance- und Sync-Gründen **nicht** als BLOB in der CR-SQLite-Datenbank synchronisiert. Stattdessen:

1. Die Protokoll-Bytes werden **zstd-komprimiert** (optional mit Dictionary, z. B. `rpt-v1.dict`).
2. Die komprimierten Bytes werden als **Packfile-Eintrag** in den Hub geschrieben (append-only).
3. In der DB wird nur eine **Referenz** gespeichert (`measurement_protocols`):
   - `pack_file`, `pack_offset`, `pack_length` (wo liegt das Protokoll im Packfile?)
   - `blake3` (Integritäts-Hash über die komprimierten Bytes; gespeichert als 32-Byte-BLOB, angezeigt als Hex)
   - `dict_id` (welches Dictionary wurde verwendet, falls vorhanden?)
4. Zusätzlich wird das Protokoll lokal im **Protokoll-Cache** abgelegt, damit es offline gelesen werden kann.

8.2.1 Offline-Import (Hub nicht erreichbar)

Offline-Import ist erlaubt. In diesem Fall:

- Die DB enthält zunächst eine „ausstehende“ Referenz (`pack_file=''` und/oder `pack_length=0`).
- Das Protokoll liegt bereits im lokalen Cache des Import-Clients (unter dem `blake3`-Namen).
- Zusätzlich wird lokal eine Outbox geführt (`protocol_upload_outbox`), die Upload-Versuche, Backoff und Fehlerstatus verwaltet.
- Sobald der Hub wieder erreichbar ist, lädt der Import-Client das Protokoll automatisch in den Hub hoch (Outbox-Flush) und ergänzt `pack_file/offset/length`. Der Status ist in der UI auf der Import-Seite unter „Ausstehende Uploads“ sichtbar und kann dort manuell per „Jetzt synchronisieren“ angestoßen werden.

Wichtig: Wenn ein Protokoll **noch nicht** im Hub archiviert wurde, können andere Clients die Messung sehen, aber das Protokoll erst öffnen, nachdem der Import-Client wieder online war und der Upload erfolgt ist.

8.2.2 Cache-Reset

Der lokale Protokoll-Cache ist ein Performance-/Offline-Feature. Ein „Cache leeren“ löscht nur lokale Dateien. Beachten Sie:

- Der Cache ist als **LRU** mit einem Größenlimit ausgelegt (aktuell ~100 MB pro Hub). Bei Bedarf werden ältere, nicht mehr benötigte Einträge automatisch entfernt.
- Wurde ein Protokoll offline importiert und noch nicht in den Hub hochgeladen, darf der Cache **nicht** geleert werden, da sonst der spätere Upload nicht mehr möglich ist. Prüfen Sie vorher „Ausstehende Uploads“ (Import-Seite): wenn dort ausstehende/pausierte Uploads angezeigt werden, darf der Cache nicht geleert werden. Zusätzlich zeigt die Seite „Cache leeren“ an, ob noch ausstehende Uploads existieren; in diesem Fall ist „Cache leeren“ deaktiviert.

8.2.3 Dictionary-Training (optional)

Da RPT-Protokolle oft sehr ähnlich sind, kann ein Zstd-Dictionary den Speicherbedarf und die I/O-Last deutlich reduzieren. Das Dictionary kann bei Bedarf neu trainiert werden:

- `pnpm -s dict:train:rpt`
  Trainiert `src-tauri/res/zstd-dicts/rpt-v1.dict` aus den Beispieldateien unter `./RPT`.

Zusammenfassung (Protokolle)

- Protokolle liegen als zstd-Bytes in `protocols/...` (Hub) – nicht als DB-BLOB.
- Integrität wird über `measurement_protocols.blake3` geprüft (BLAKE3 über komprimierte Bytes).
- Offline-Import: Protokoll zuerst lokal, Upload in den Hub wird später nachgeholt.
- Cache nur leeren, wenn keine „pending“ Protokolle mehr existieren.

8.3 3) CR-SQLite/CRDT: Warum es gebraucht wird (technisch)

CR-SQLite behandelt Tabellen als **CRR** und protokolliert Änderungen in `crsql_changes`. Daraus ergibt sich:

- Die App kann lokal schreiben (Replica), ohne File-Locks im Hub „dauerhaft“ zu halten.
- Der Sync ist „best effort“ und führt zu **eventual consistency**: nach erfolgreichem Sync konvergieren alle Replicas deterministisch.

Technisch wichtig sind Schema-Constraints von CR-SQLite:

- Primary Keys müssen non-NULL sein.
- `NOT NULL`-Spalten benötigen Defaults (Forward/Backward-Compatibility).
- Schema-Änderungen erfolgen über App-Migrations.

Die CR-SQLite Extension wird als Datei `resources/extensions/crsqlite.dll` gebündelt und vom SQL-Plugin geladen.

Zusammenfassung (CR-SQLite)

- Lokale Replica reduziert Locking-Probleme auf Netzlaufwerken.
- CR-SQLite stellt Anforderungen an das Schema → IDs/Defaults sind wichtig.
- Extension liegt als `crsqlite.dll` im App-`resources`-Ordner.

8.4 4) Authentifizierung: Passwort + Pepper + Stronghold (technisch)

FMB Log nutzt zwei Ebenen:

- **Passwort-Hash in der DB:** Argon2id über `password + pepper` (Pepper ist pro Nutzer zufällig).
- **Pepper in Stronghold:** Das Pepper liegt nicht in SQLite, sondern in `vaults/<user_id>.vault`.

Login-Ablauf (vereinfacht):

1. Aus eingegebenem Passwort + `user_id` wird ein **Vault-Key** abgeleitet (Argon2id, 32-Byte-Key).
2. Mit diesem Key wird die Stronghold-Vault entschlüsselt.
3. Wenn Entschlüsselung fehlschlägt → Passwort falsch.
4. Wenn Entschlüsselung gelingt → Pepper wird gelesen → Hash wird geprüft.

Zusammenfassung (Auth)

- Hash-Swap-Angriffe via DB werden durch user-spezifisches Pepper verhindert.
- Pepper liegt in Stronghold-Vault, nicht in SQLite.
- Vault-Key ist an `user_id + password` gebunden.

8.5 5) Signaturen & Delegation: was wird womit signiert?

FMB Log signiert verschiedene Datenklassen, um Manipulationen in einer „Datei-DB“ zu erkennen.

8.5.1 5.1 Root-Trust und DB-Signierschlüssel (Admin)

- **Root Public Key** ist **fest in der App** eingebaut (Compile-Time).
- Der **DB-Public-Key** liegt als Datei neben der DB: `<db>.integrity.pub.json`.
- Das **DB-Key-Zertifikat** `<db>.integrity.dbkey.json` ist Root-signiert und muss vorhanden sein (Fail-Closed).
- Der **DB-Private-Key** liegt in Stronghold: `vaults/<db>.integrity.vault` und wird per Signier-Passwort entsperrt.

Damit werden u. a. signiert:

- `users`, `groups`, `user_groups`, `group_permissions`
- `capability_certs` (Delegations-Zertifikate)
- Admin-Security-Einstellungen

8.5.2 5.2 User-Signaturen (Key-User, ohne Admin-Signier-Passwort)

Jeder Nutzer besitzt zusätzlich einen **User-Signierschlüssel** (Ed25519) im eigenen Vault:

- `vaults/<user_id>.vault` enthält neben Pepper auch den User-Signing-Key.
- Der Key-User kann Stammdaten signieren, ohne den DB-Signier-Key zu kennen.

Damit ein Key-User wirklich „nur in seinem Scope“ signieren kann, wird die Berechtigung über **Capability-Zertifikate** delegiert:

- Tabelle: `capability_certs`
- Signiert vom DB-Signier-Key (Admin-Aktion)
- Gültigkeit: `issued_at`, optional `expires_at`, optional `revoked_at`

Ein Key-User darf z. B. Stammdaten nur signieren, wenn ein gültiges Zertifikat für den Scope existiert (z. B. `masterdata.fgw`, `masterdata.nv`, `masterdata.fmk`).

Zusammenfassung (Signaturen)

- Root-Trust: Root Public Key ist in der App eingebaut, DB-Key wird per Zertifikat an Root gebunden.
- Admin-Key (DB-Key) signiert Security-kritische Daten und Delegationen.
- Key-User signieren Stammdaten per User-Key + Admin-Delegation (Capability-Zertifikat).

8.6 6) Hash-Algorithmen: BLAKE3 vs. SHA-256

In FMB Log werden zwei Hash-Algorithmen bewusst parallel genutzt:

- **BLAKE3** (schnell, parallelisierbar):
  Für Integrität/Signaturen und Protokoll-Integrität im Betrieb (Audit, Packfiles, Audit-Cache).
- **SHA-256** (Standard für externe Systeme):
  Für RFC3161-TSA (FreeTSA erwartet SHA-256) – der Imprint (`tsa_snapshot_sha256`) und der Token-Hash (`tsa_token_sha256`) bleiben SHA-256.

Zusammenfassung (Hashes)

- BLAKE3: intern, schnell (Audit/Protokolle/Signatur-Workflows).
- SHA-256: extern kompatibel (TSA).

8.7 7) Tagesabrechnung: Snapshot, QR-Code, TSA (technisch)

Beim PDF-Export wird ein **Snapshot** der exportierten Inhalte gebildet und gehasht:

- `daily_reports.snapshot_hash`: BLAKE3 des Snapshots (DATA-Hash)
- `daily_reports.pdf_sha256`: BLAKE3 der finalen PDF-Bytes (PDF-Hash; Spaltenname historisch)
- `daily_report_invalidations`: signierte Invalidierungs-Events (wer/wann/warum; referenziert Report + Snapshot/PDF-Hash zum Zeitpunkt der Invalidierung)
- Optional (TSA):
  - `tsa_provider`, `tsa_gen_time`, `tsa_snapshot_sha256`, `tsa_token_sha256`, `tsa_token_base64`

Hinweis: Hashes (32 Byte) und Signaturen (Ed25519, 64 Byte) werden in SQLite als **BLOB** gespeichert (I/O-sparend). Im Frontend werden BLOBs als **Base64-Strings** transportiert (keine JSON-Byte-Arrays), in der UI werden Hashes aber als **Hex/Fingerprint** angezeigt.

Die PDF enthält in der Fußzeile:

- einen QR-Code mit kompakter Payload (u. a. Snapshot-Hash, optional TSA-Token-Hash)
- daneben kurze Fingerprints (`DATA: …` und optional `TSA: …`) für manuelle Gegenprüfung

Zusammenfassung (Tagesabrechnung)

- Snapshot-Hash ist BLAKE3 (schnell, in QR/Fingerprint als DATA).
- PDF-Hash ist BLAKE3 (schnell, als Referenz in der Historie).
- Für TSA wird zusätzlich `tsa_snapshot_sha256` gespeichert (RFC3161-kompatibler Imprint).
- QR-Code + Fingerprints ermöglichen schnelle Verifikation in der Historie/Audit.

8.8 8) Audit & Performance (technisch)

Der Audit prüft Security-Signaturen, Delegationen, Stammdaten-Signaturen, Messdaten-Signaturen und Protokoll-Integrität. Um die Laufzeit zu reduzieren, nutzt FMB Log ein lokales Cache-Konzept:

- Tabellen: `audit_cache`, `known_keys`, `known_deps`
- Enthält BLAKE3-basierte „Deps/Message/Signature“ Hashes pro Scope/Entity.
- `audit_cache` ist **lokal** (nicht CRR) und wird nicht in den Hub synchronisiert.
- Zusätzlich wird `row_db_version` gespeichert, damit sich Audits auf geänderte Zeilen beschränken können.
- Um Platz/I/O zu sparen, werden wiederkehrende Hashes (Verify-Key und Deps) dedupliziert und in `audit_cache` nur als FK (`*_id`) referenziert.

Zusammenfassung (Audit)

- Audit ist vollständig, aber kann teuer sein → Cache + Inkrementalität reduziert Laufzeit.
- `audit_cache` ist pro Client lokal und wird nicht mit dem Hub synchronisiert.

8.9 9) Phasenzeiten & Performance-Log

Bei langen Operationen zeigt FMB Log **Phasenzeiten** im UI an und schreibt dieselben Informationen zusätzlich als strukturierte Einträge (`[perf]`) in die Diagnose-Logdatei.

Betroffene Aktionen:

- **Administration → Audit** (prüft Security/Stammdaten/Messdaten/Protokolle)
- **Administration → Einstellungen → Stammdaten neu signieren**
- **Tagesabrechnung → PDF exportieren**

Die Logzeilen enthalten JSON mit `action`, `ok`, `total_ms` und `phases` (je Phase: `name`, `ms`). So können Sie bei Performance-Problemen nachvollziehen, ob z. B. **I/O (Hub/Protokolle)**, **Hashing** oder **DB-Zugriffe** dominieren.

9 Troubleshooting

9.1 App schließt sich ohne Meldung / „Crash“

Im Production-Build läuft Rust mit `panic = "abort"`. Das bedeutet: Ein Panic beendet die Anwendung sofort, ohne dass ein Fehlerdialog angezeigt wird.

FMB Log schreibt deshalb eine Logdatei mit Crash-Hinweisen und Backtrace:

- Öffnen über **Administration → Einstellungen → Diagnose → Log öffnen**
- Pfad (Windows): `%LOCALAPPDATA%\\com.ewn.fmblog\\logs\\fmb-log.log`

Wenn die Anwendung beim Start erkennt, dass der letzte Start unerwartet endete, erscheint zusätzlich ein Hinweis auf der Anmeldeseite.

9.2 „Diese Seite funktioniert nur in der Tauri-Desktop-App“

Einige Funktionen (Dateidialoge, Zugriff auf Ressourcenpfade, PDF-Export, Datenbankzugriff) benötigen Tauri-APIs. Wenn Sie die Web-UI im Browser starten (z. B. mit `pnpm dev`), sind diese APIs nicht verfügbar und die Anwendung zeigt entsprechend einen Hinweis.

Für Desktop-Tests:

- `pnpm tauri dev`

9.3 Export/Dateidialoge funktionieren nicht

Dateidialoge und das Öffnen/exportieren von Dateien funktionieren nur in der Desktop-App und sind zusätzlich durch das Tauri-Permissions-System abgesichert. Wenn Sie eine Meldung wie „not allowed“ oder „Permission … not found“ sehen, prüfen Sie:

- ob Sie die Funktion in der Desktop-App ausführen (nicht im Browser),
- ob das passende Plugin (z. B. `opener`) aktiviert ist,
- ob die Berechtigung in `src-tauri/capabilities/default.json` erlaubt ist (z. B. `opener:allow-open-path`).

9.4 Netzwerk-DB: sporadische Locks/Fehler

Wenn die Datenbank auf einem Netzlaufwerk liegt, können sporadische Sperren oder Zugriffsfehler auftreten. Ursache ist meist das Zusammenspiel aus SMB-Locking, OpLocks und gleichzeitigen Schreibzugriffen. Typische Maßnahmen:

Prüfen:

- Schreibrechte im Zielordner und Stabilität der Verbindung
- Share/SMB-Konfiguration (Locking/OpLocks)
- ob mehrere Nutzer gleichzeitig schreiben (z. B. paralleler Import)

Wenn Fehler reproduzierbar unter Last auftreten, ist ein lokaler DB-Pfad pro Nutzer die robustere Variante.

9.5 Protokoll kann nicht geladen werden

Wenn beim Öffnen eines Messprotokolls eine Fehlermeldung erscheint, sind die häufigsten Ursachen:

1. **Hub nicht erreichbar** (Netzlaufwerk offline / Pfad falsch)
   Protokolle werden im Hub-Ordner als Packfiles unter `protocols/…` gespeichert. Ist der Hub nicht erreichbar, kann die App Protokolle nur laden, wenn sie bereits im lokalen Cache liegen.
2. **Offline-Import auf anderem Rechner**
   Wurde eine Messung importiert, während der Hub nicht erreichbar war, ist das Protokoll zunächst nur auf dem importierenden Rechner im lokalen Cache verfügbar. Der Upload in `protocols/…` wird automatisch nach dem nächsten erfolgreichen Sync nachgeholt. Bis dahin können andere Clients die Messung sehen, das Protokoll aber nicht öffnen.
3. **Lokaler Cache wurde geleert**
   Wenn ein Protokoll noch nicht in das Hub-Archiv hochgeladen war und der lokale Cache gelöscht wurde, kann es ggf. nicht mehr automatisch hochgeladen werden. In diesem Fall muss das Protokoll erneut importiert werden (oder aus einem Backup wiederhergestellt werden).

9.6 Import hängt / Import beendet nicht

Wenn ein Import scheinbar „endlos“ läuft, zeigt die Import-Seite unter dem Button **Importieren** den aktuellen Schritt (z. B. „Schritt: Protokoll speichern…“). Das hilft, den Engpass einzugrenzen.

FMB Log verwendet außerdem Timeouts, damit ein Import nicht unbegrenzt hängen bleibt. Falls eine Zeitüberschreitung auftritt, enthält die Fehlermeldung den betroffenen Schritt.

Typische Ursachen je Schritt:

- **„Protokoll speichern...“**: Zugriff auf den Hub-Ordner (Schreibrechte/SMB/Antivirus) oder Initialisierung der zstd-Kompression.
- **„Gebinde signieren...“ / „Signatur (neu/alt)...“**: Stronghold-Vault nicht erreichbar (liegt im Hub-Ordner) oder falsches Signier-/Vault-Passwort.
- **„Messung speichern...“**: lokale Replica-DB ist blockiert (z. B. durch parallele Aktionen) oder Hub/Sync verursacht Locking unter Last.

Für die Diagnose ist die Logdatei hilfreich:

- Öffnen über **Administration → Einstellungen → Diagnose → Log öffnen**
- Pfad (Windows): `%LOCALAPPDATA%\\com.ewn.fmblog\\logs\\fmb-log.log`

Kurzcheck

- Browser-Modus (`pnpm dev`) eignet sich für UI-Layout, nicht für Tauri-Funktionen.
- Desktop-Tests immer mit `pnpm tauri dev`.
- „not allowed“ deutet meist auf fehlende Tauri-Permissions hin.

10 Rechte (Permission Keys)

Die Anwendung nutzt Gruppenrechte. Admins besitzen immer alle Rechte.

Key	Bedeutung
`measurements.import`	Messungen einlesen
`measurements.update`	Messungen ändern
`measurements.delete`	Messungen löschen
`measurements.update_date`	Messdatum ändern
`reports.invalidate`	Tagesabrechnungen ungültig machen
`fmk.create`	FMK anlegen
`fmk.update`	FMK ändern
`fmk.delete`	FMK löschen
`nv.create`	NV anlegen
`nv.update`	NV ändern
`nv.delete`	NV löschen
`fgw.update`	Freigabewerte ändern
`users.reset_passwords`	Passwörter anderer Nutzer zurücksetzen

11 Rechner & Formeln

11.1 1) Freigabewert für einen Nuklidvektor

Für einen NV und einen Pfad gilt (vereinfachte Darstellung):

$$FGW_{NV} = \frac{1}{\sum_i \frac{p_i}{f_{i,p}}}$$

- p_i: Anteil des Nuklids i im NV (0…1)
- $f_{i,p}$: Freigabewert aus FGW für Nuklid i und Pfad p

11.2 2) SW/KF

- **SW** reduziert den Freigabewert: $FGW_{\text{eff}} = FGW_{NV} \cdot SW$
- **KF** erhöht die Aktivität: $OG_{\text{eff}} = OG/KF$

11.3 3) Bestehenslogik (OG)

Eine Messung besteht für einen Pfad, wenn:

$$OG_{\text{eff}} \leq FGW_{\text{eff}}$$

11.4 4) Einheit/Umrechnung

Umrechnungen zwischen Bq/g und Bq/cm^2 benötigen eine Umrechnung g/cm^2 (z. B. aus Flächenmasse).