Systemdokumentation Lunchify

Stand: Mai 2025

Inhalt

1. Architekturübersicht

Lunchify besteht aus drei Schichten:

1. **Präsentation**
    - **JavaFX-Client** (Modul javafx.controls, javafx.fxml)
    - FXML-Views, Controller-Klassen, CSS-Styles
2. **Geschäftslogik**
    - Controller- und Service-Klassen (z. B. InvoiceDAO, UserDAO)
    - BCrypt für Passwort-Hashing
3. **Datenhaltung**
    - **Supabase Postgres** mit Storage
    - Tabellen, Enums, RLS

Kommunikation: JDBC (DB) und HTTPS (Storage).

2. Datenbankmodell

2.1. Tabelle: benutzer

Spalte	Typ	Beschreibung
id	int8 PK	Eindeutige Benutzer-ID
name	text	Anzeigename des Users
email	text UQ	Login-E-Mail
rolle	text	Enum (admin / user)
passwort	text	BCrypt-Hash
must_change_password	bool	Flag: Passwort-Änderung beim Login erforderlich

2.2. Tabelle: rechnung

Spalte	Typ	Beschreibung
id	int8 PK	Eindeutige Rechnungs-ID
user_id	int8 FK → benutzer	Zugehöriger Benutzer
file_url	text	Pfad/Dateiname im Supabase-Storage
type	text	Enum RESTAURANT / SUPERMARKET
invoice_amount	numeric	Ursprünglicher Rechnungsbetrag (€)
reimbursement_amount	numeric	Berechneter Rückerstattungsbetrag (€)
status	text	Enum SUBMITTED, APPROVED, REJECTED
upload_date	date	Datum des Uploads

2.3. Tabelle: benutzer_rechnung_starred

Spalte	Typ	Beschreibung
id	int8 PK	Eindeutiger Eintrag
benutzer_id	int8 FK → benutzer	User, der gefavorit hat
rechnung_id	int8 FK → rechnung	Gefavoritete Rechnung
starred_date	timestamp	Zeitstempel des Favorisierens

2.4. Tabelle: notification

Spalte	Typ	Beschreibung
id	int8 PK	Eindeutige Notifikation-ID
user_id	int8 FK → benutzer	Empfänger
message	text	Benachrichtigungstext
timestamp	timestamp	Versand-/Erstellzeit
erstatteter_betrag	numeric	Betrag der rückerstatteten Rechnung, falls zutreffend

2.5. Tabelle: refund_config

Spalte	Typ	Beschreibung
category	text PK	Enum RESTAURANT / SUPERMARKET
amount	numeric	Max. Rückerstattung in €

3. Supabase Storage

- Bucket**: rechnung**
- Dateipfade**: <user>_<timestamp>_<originalname>.png/jpg/...**
- Öffentliche URL**:**
  - https://<PROJECT>.supabase.co/storage/v1/object/public/rechnung//<trimmed-filename>
- RLS: Nur eigener Bucket; öffentliche Leserechte für Bilder zulässig.

4. Sicherheitskonzepte

- Passwort: BCrypt hashing (BCrypt.hashpw, checkpw)
- Default-Passwort: default123 → sofort ändern
- RLS (Row-Level Security):
- Tabelle rechnung: policy select on rechnung for web using (user_id = auth.uid());
- Tabelle benutzer_rechnung_starred: analog
- Roles: Postgres-Enum benutzer_rolle; Access via ?::benutzer_rolle

4.1. RLS-Policies (Auszug)

```
ALTER TABLE rechnung ENABLE ROW LEVEL SECURITY;
CREATE POLICY select_own_invoices
  ON rechnung
  FOR SELECT
  USING (user_id = auth.uid());

ALTER TABLE benutzer_rechnung_starred ENABLE ROW LEVEL SECURITY;
CREATE POLICY select_own_starred
  ON benutzer_rechnung_starred
  FOR SELECT
  USING (benutzer_id = auth.uid());
-- Admin-Rollen haben separate Policies (broad access)
```

4.2. GUI-Komponenten & Styling

- FXML + eigene CSS-Klassen
- dashboard.css: Grid-Layouts, Farbvariablen (--primary, --accent)
- controlsfx: DatePicker-Skin, Tooltips

4.3. Tests & Qualitätssicherung

- Unit Tests (JUnit5)
  - InvoiceDAO (CRUD-Operationen gegen H2-InMemory)
  - reimbursement = min(invoiceAmount, cap)
  - UserDAO.validateLogin() (Mocked JDBC)
- Integrationstests
  - HTTP-Upload + Download (mit lokalem Supabase-Emulator)
- UI-Smoke Tests
  - TestFX: Login → Dashboard öffnen → Navigation
- Code Coverage
  - Ziel ≥ 80 %

4.4. Prozesse & Abläufe

- **Neuen User anlegen (Admin)**
  - Insert in benutzer mit Flag must_change_password = TRUE
- **Erstlogin**
  - validateLogin() liest must_change_password, zeigt Dialog
  - updatePasswordAndClearFlag() setzt Flag auf FALSE
- **Rechnung hochladen (User)**
  - Datei in Storage
  - Insert in rechnung, Status=SUBMITTED
- **Favorisieren (User)**
  - Toggle über DAO benutzer_rechnung_starred
- **Benachrichtigung**
  - On status change: Insert in notification
  - (Später) E-Mail-Versand