High Level Design

cohabit

Version 1.1

Besim Kurteshi, Albin Qerkinaj, Berkant-Yasin Sener,
Veljko Stojadinovic

Dokument-Historie

Version	Status	Datum	Verantwortlicher	Änderungsgrund
1.0	draft	01.12.2025	Albin Qerkinaj	
1.1	initial version	07.12.2025	Besim Kurteshi	

Abkürzungen

Inhaltsverzeichnis

1. Einleitung

1.1. Zweck

Das System "cohabit" dient als Plattform zur Erhöhung der Transparenz auf dem Wohnungsmarkt. Es ermöglicht Mietern (Usern), subjektive Erfahrungen und Bewertungen zu Wohnungen abzugeben, und Anbietern (Vermietern/Verwaltungen), ihre Objekte darzustellen. Ziel ist es, Mietentscheidungen durch authentische Informationen über soziale Faktoren und Umweltbedingungen zu unterstützen.

1.2. Systemüberblick

Bei dem System handelt es sich um eine webbasierte Anwendung. Die Architektur folgt dem Prinzip einer **Single Page Application (SPA)**.

- **Frontend:** Interagiert mit dem Benutzer.
- **Backend:** Stellt eine RESTful API bereit und verarbeitet die Geschäftslogik.
- **Datenbank:** Speichert persistent alle relevanten Daten (Relationale DB).

1.3. Definitionen

- **User:** Ein Mieter oder Interessent, der Wohnungen sucht und bewertet.
- **Anbieter:** Ein Vermieter oder Verwalter, der Wohnungen anlegt und verwaltet.
- **Wohnung:** Das digitale Profil einer Wohneinheit.
- **Bewertung:** Eine strukturierte Meinung (Sterne + Text) eines Users zu einer Wohnung.

1.4. Betriebsumgebung

1.5. Dokumentenüberblick

2. *Komponentenname* Überblick

2.1. Rolle *Komponentenname* im Gesamtsystem

3. 2.1 Komponenten des Systems

Das System besteht aus drei logischen Hauptkomponenten:

1. **Client (Frontend - Svelte):**
   - *Rolle:* Präsentation der Daten, Entgegennahme von Benutzereingaben, Vor-Validierung.
   - *Funktionalität:* Darstellung der Suchergebnisse, Formulare für Login/Registrierung und Bewertungen. Kommunikation via HTTPS/JSON mit dem Backend.
2. **Application Server (Backend - Quarkus):**
   - *Rolle:* Verarbeitung der Geschäftslogik, Sicherheitsüberprüfung (Authentifizierung/Autorisierung via JWT), Schnittstelle zur Datenbank.
   - *Funktionalität:* REST-Endpunkte bereitstellen, Validierung der Daten (Serverseitig), Management der User-Sessions.
3. **Data Storage (Datenbank - PostgreSQL):**
   - *Rolle:* Persistente Speicherung.
   - *Funktionalität:* Speicherung von User-Credentials (gehasht), Wohnungsdaten und Bewertungen unter Wahrung der referenziellen Integrität.

3.1. Funktionalität

4. Detailliertes Design

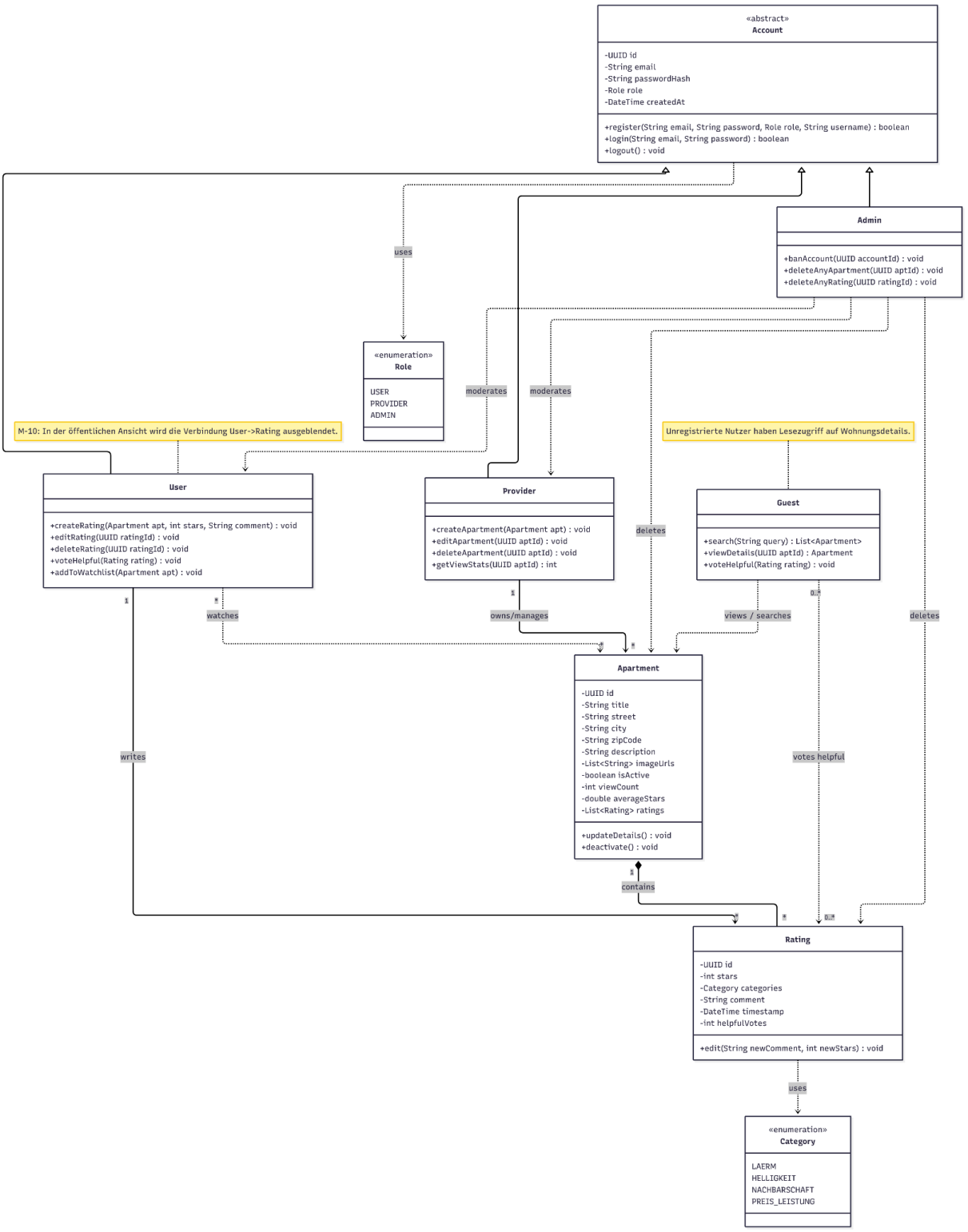
4.1. Architektur

Die Komponenten bilden eine klassische **3-Tier-Architektur** (Schichtenarchitektur):

- Der **Client** sendet Anfragen an die API.
- Der **Application Server** verarbeitet diese Anfrage stateless.
- Die **Datenbank** liefert die Rohdaten zurück an den Server, der diese als DTOs (Data Transfer Objects) an den Client sendet.

4.2. UML Diagramme

4.2.1. Klassendiagramm



4.2.2. Aktivitätsdiagramme

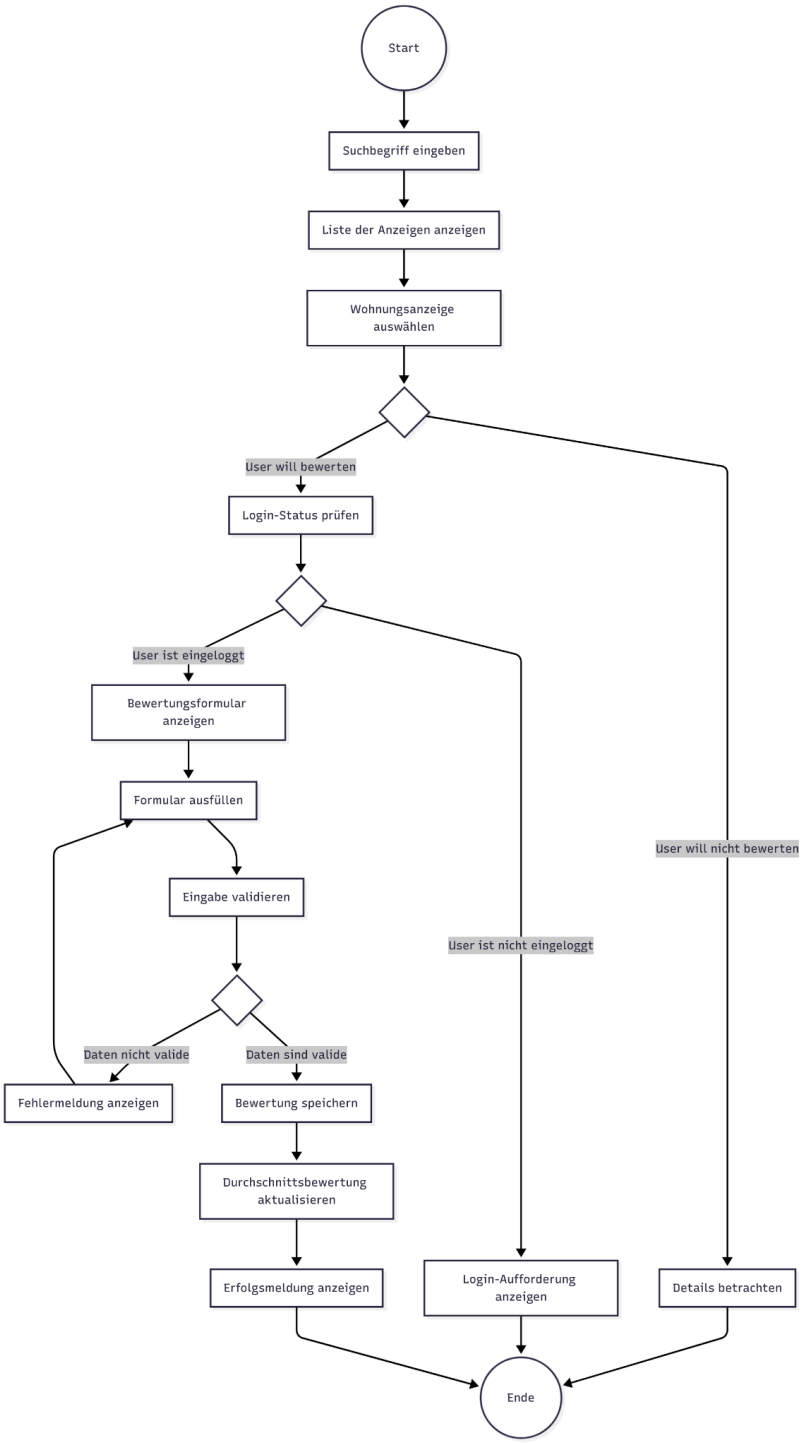


Abbildung 4.2.2.1 Aktivitätsdiagramm - User bewertet Wohnungsanzeige

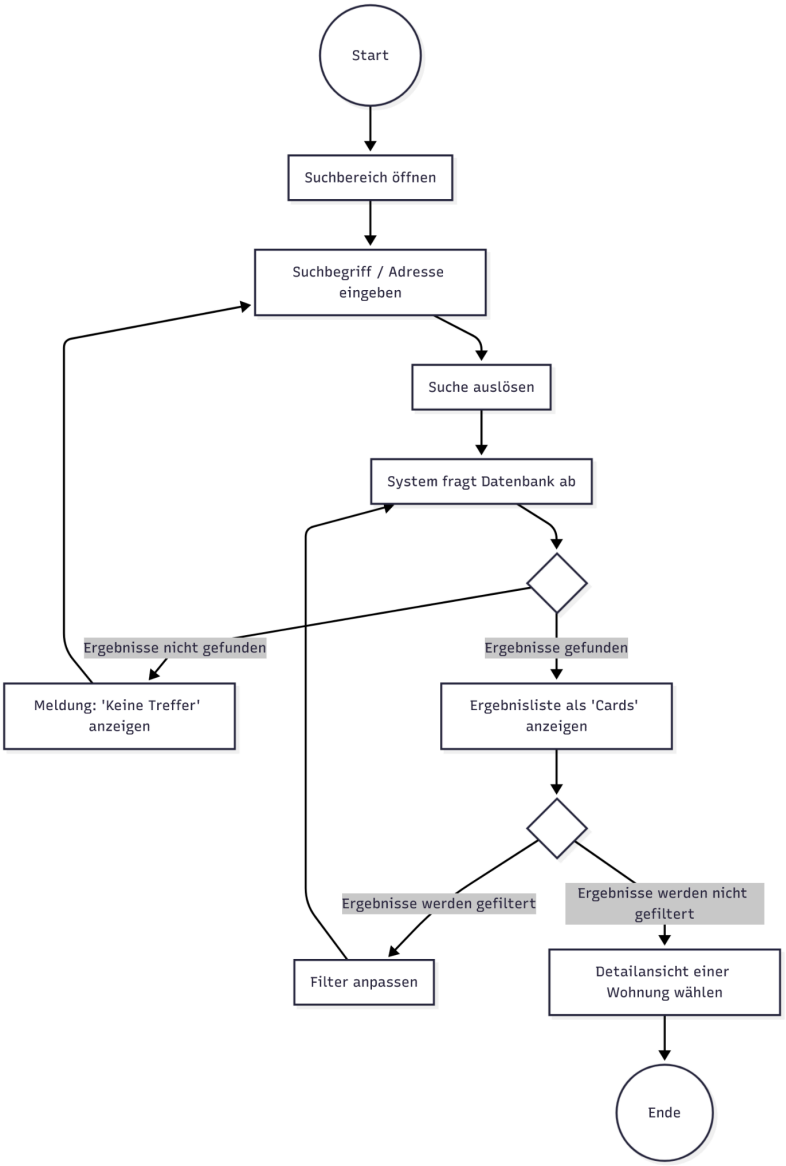


Abbildung 4.2.2.2 Aktivitätsdiagramm - User sucht Wohnungsanzeige

4.2.3. Sequenzdiagramme

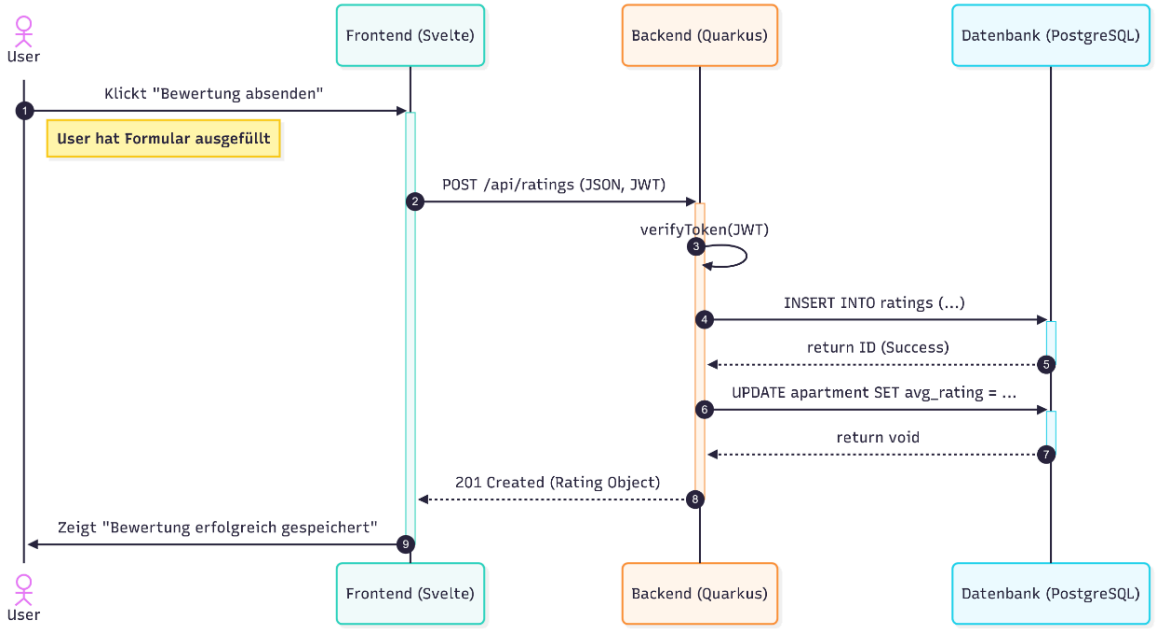


Abbildung 4.2.3.1 Sequenzdiagramm - User bewertet

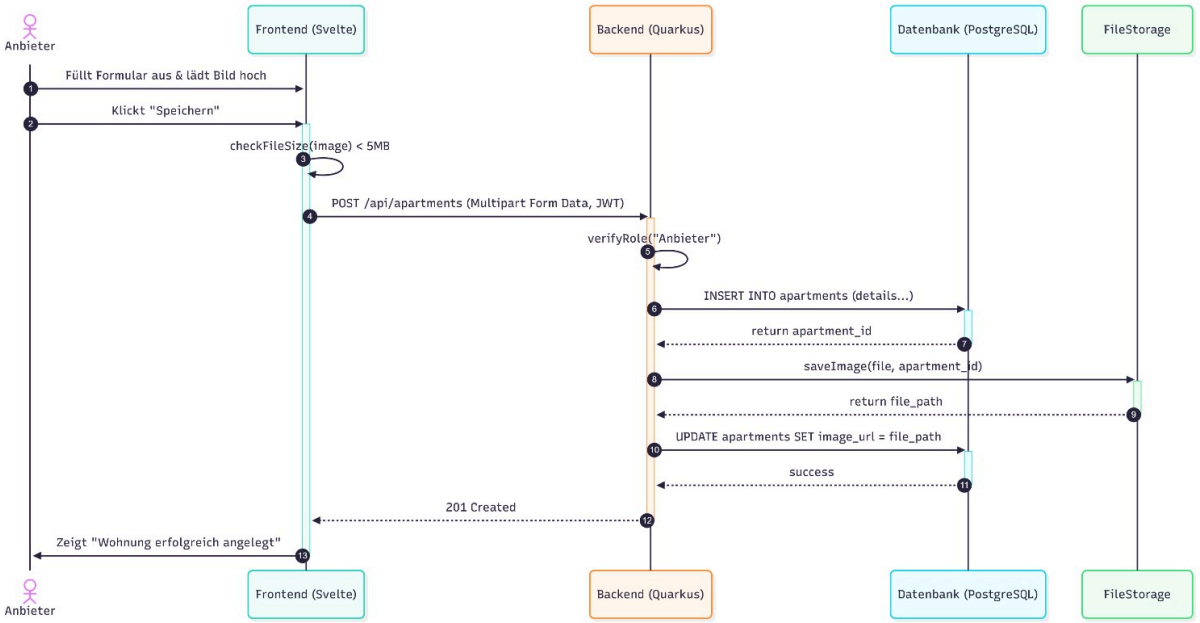


Abbildung 4.2.3.2 Sequenzdiagramm - Anbieter ersteltl Wohnungsanzeige

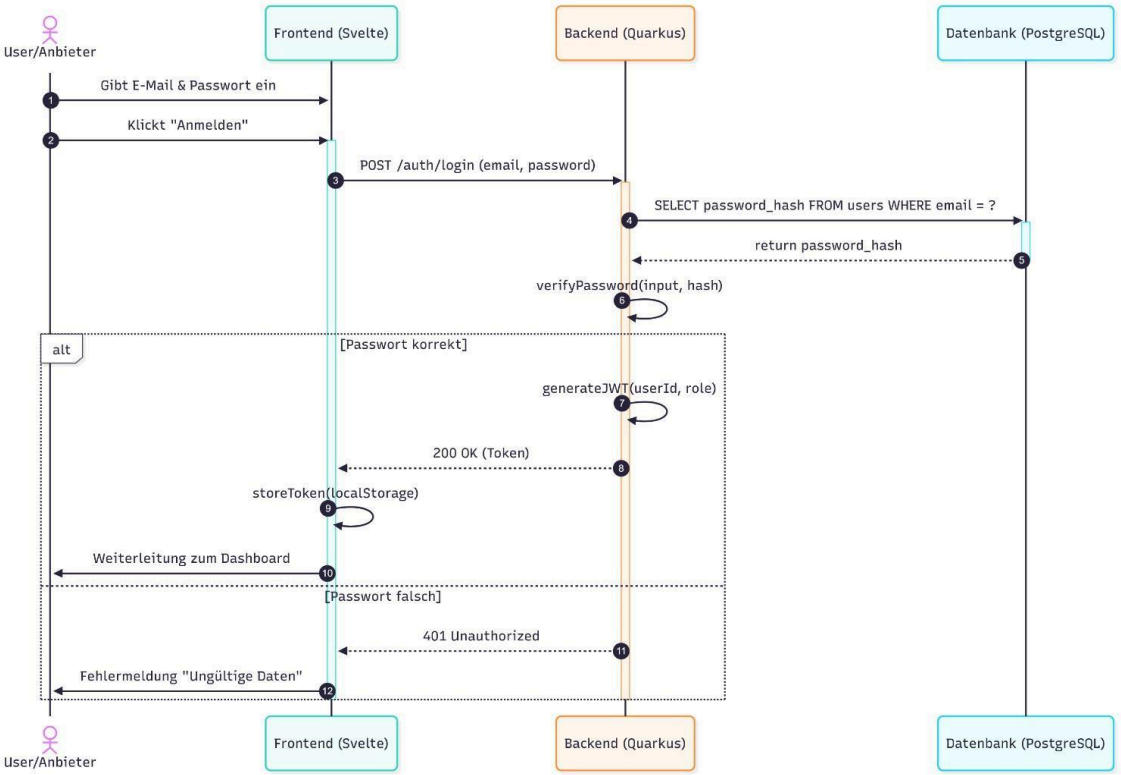


Abbildung 4.2.3.3 Sequenzdiagramm - User/Anbieter loggt sich ein

5. Anhang