Dokumentation üK223 Gruppe 2

Multi-User Applikation objektorientiert realisieren

Anik, Ermal, Kenneth

Inhaltsverzeichnis

1. Projektauftrag

1.1 Projektbeschreibung

Im Projekt „OurSpace“ wird eine Social-Media-Website entwickelt, auf der mehrere Nutzer Blogbeiträge erstellen und verwalten können.

Die Anwendung wird als Full-Stack-Lösung mit React, Spring Boot und PostgreSQL umgesetzt. Dabei liegt der Fokus auf Mehrbenutzerfähigkeit, Sicherheit und einer guten Dokumentation. Administratoren können Kategorien erstellen und Nutzer verwalten, während die Nutzer eigene Blog-Posts veröffentlichen und teilen können

1.2. Funktionale Anforderungen

User Rollen und Privilegien:

- Bestehende Rollen und Autoritäten werden bearbeitet oder erweitert, um untenstehende Anforderungen zu erfüllen und zu testen.
- Die persönlichen Informationen eines Users sind nur für Administratoren oder den User selbst zugänglich.
- Admins können ausserdem andere Benutzer bearbeiten, erstellen und löschen.

Frontend:

- Login-Page, die öffentlich zugänglich ist (bereits vorhanden)
- Eine öffentlich zugängliche Homepage (bereits vorhanden)
- Eine Homepage für alle eingeloggten User
- Eine Admin Page (nur für Admins zugänglich).
- Mindestens eine Komponente, um gruppenspezifische Funktionalitäten im Frontend zu ermöglichen.

Security:

- Jeder REST-Endpoint soll nur mit sinnvollen Autoritäten zugänglich sein. Dies wird mit automatisierten Tests überprüft.
- Es gibt Bereiche des Front-Ends die nur für eingeloggte Benutzer zugänglich sind.
- Es gibt Bereiche des Front-Ends die nur für Admins zugänglich sind.
- Der Authentifizierung-Mechanismus wird mit JSON-Web-Tokens implementiert. (bereits vorhanden)

Gruppenspezifischer Auftrag:

Blog Posts

- Erstellen Sie ein Blog Model, das die Information eines Blogeintrags enthält (Text, Titel, Kategorie, Autor)
- Jeder Eintrag in Blog kann eindeutig zu einem User als Autor:in zugeordnet werden.
- Erstellen Sie Endpoints in ihrer Applikation, um typische CRUD-Operationen an Blogs durchzuführen.Die GET-Methode soll Pagination und Sorting nutzen.
- Nur der/die Autor:in oder ein Administrator soll ein Blog bearbeiten oder löschen können.
- Selbst unauthentifizierte Benutzer sollen Blogs (mit GET-Methode) lesen können.

1.3 Nicht funktionale Anforderungen

Implementation

- Daten werden in einer PostgreSQL Datenbank persistiert, das OR-Mapping wird mit JPA realisiert.
- Ein Frontend mit React (Typescript) wird genutzt.
- Ein Backend Spring Boot (Java) wird genutzt.
- Der Sourcecode wird täglich in einem GIT-Repository committed.

Testing

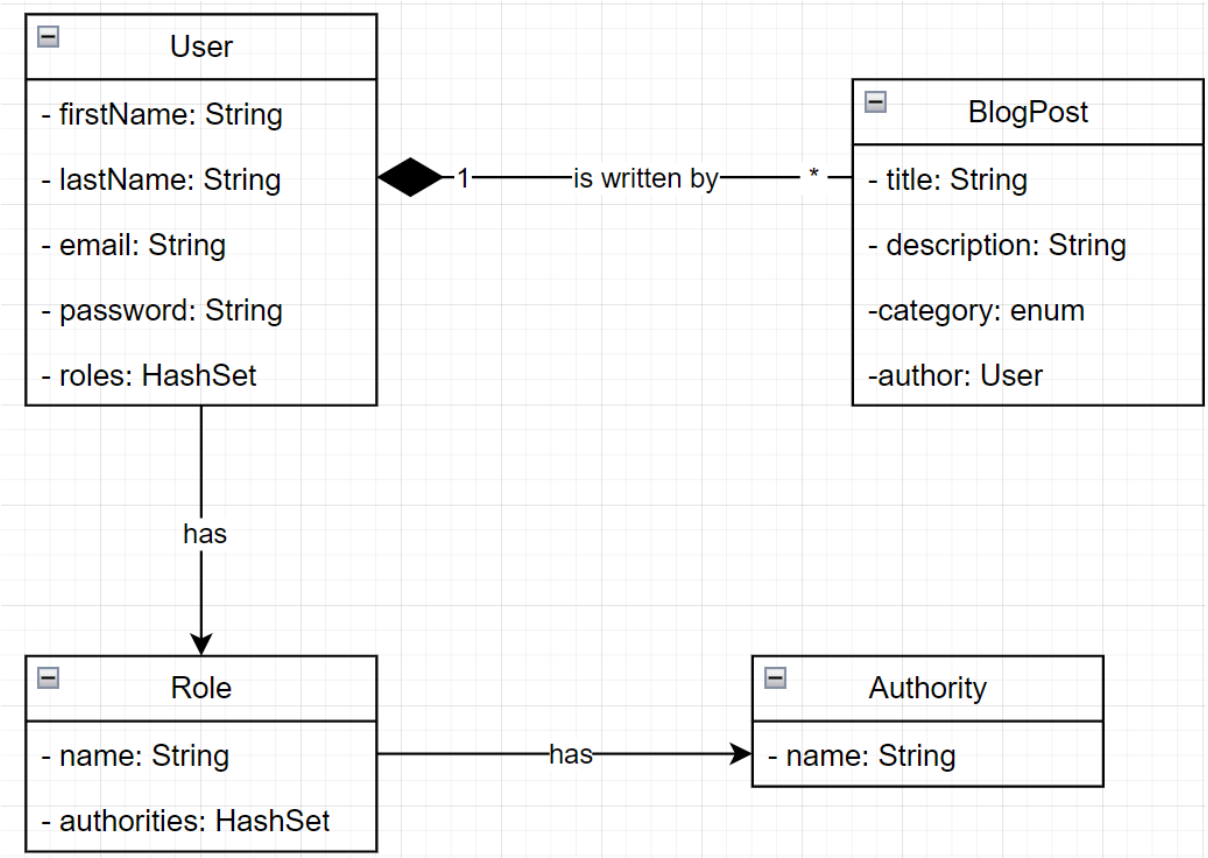
- Generelle Funktionalität aller selbst implementierten Endpoints wird mit Cypress (zwingend), Postman und/oder JUnit getestet.
- Besonderer Wert wird auf das Testen von Zugriffsberechtigungen gelegt.
- Mindestens ein Use-Case wird ausführlicher mit Cypress getestet. Dies beinhaltet im Minimum:
    - Der Endpoint wird mit mehreren Usern & Rollen getestet
    - Mindestens ein Erfolgsfall und ein Error Fall wird getestet.
    - Für diese Fälle werden Use-Cases nach UML-Standard beschrieben.

Multiuserfähigkeit

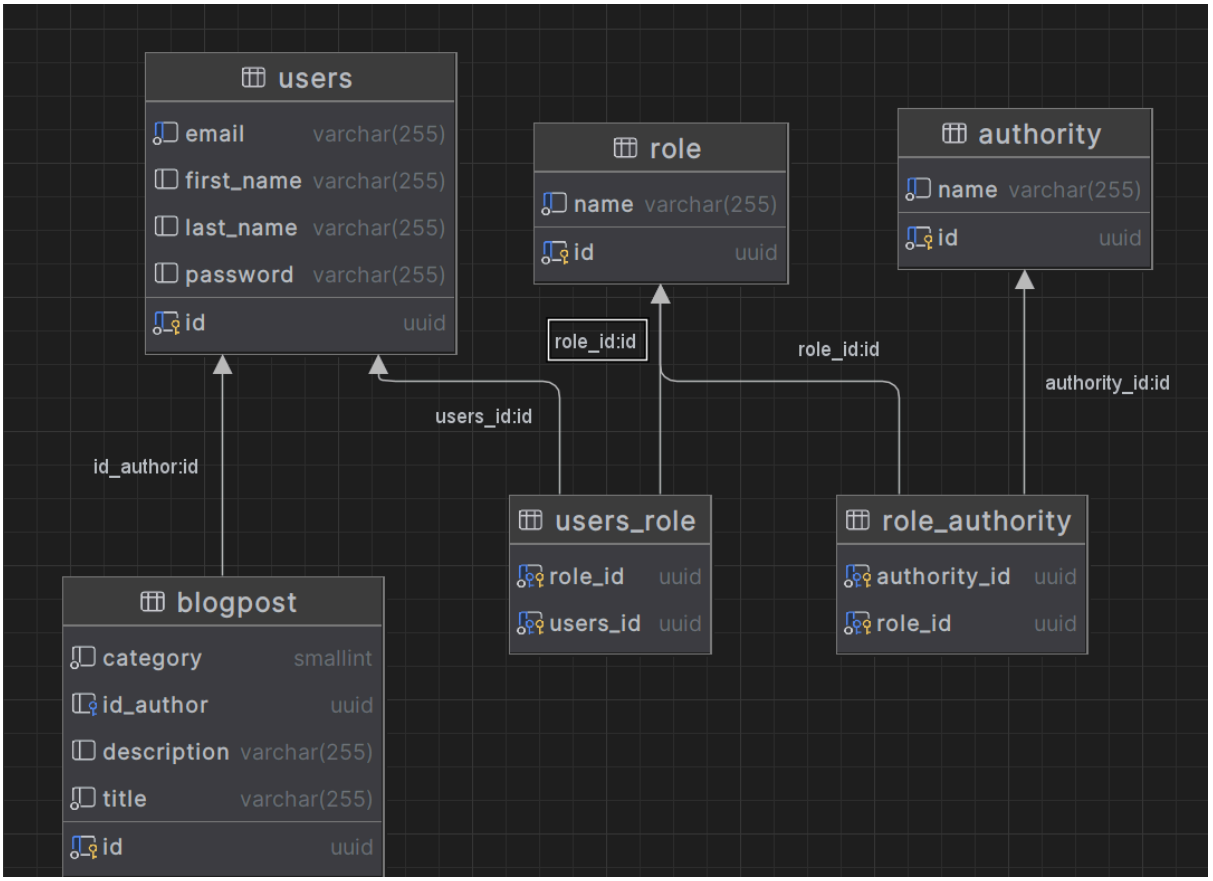
- Aspekte der Multiuserfähigkeit, wie z.B. Einhaltung der ACID-Prinzipien werden berücksichtigt

2. UML

2.1. Domain Modell



3. ERD


users
email varchar(255)
first_name varchar(255)
last_name varchar(255)
password varchar(255)
id uuid
role
name varchar(255)
id uuid
authority
name varchar(255)
id uuid
role_id:id
role_id:id
authority_id:id
users_id:id
id_author:id
users_role
role_id uuid
users_id uuid
role_authority
authority_id uuid
role_id uuid
blogpost
category smallint
id_author uuid
description varchar(255)
title varchar(255)
id uuid

4. Testing Strategien

Cypress:

Wir werden Cypress verwenden, um die Funktionalität des Frontends via End-to-End Testing zu testen. Dabei wird sichergestellt, dass das Frontend korrekt mit dem Backend interagiert sowie ob das mit den User-Berechtigungen korrekt funktioniert.

Postman

Wir haben uns entschieden, dass wir die Endpoints mit Postman anstelle von JUnit testen werden. Mithilfe von Postman überprüfen wir, ob wir die korrekten Daten erhalten und ob die User-Berechtigungen für die Bearbeitung der Daten korrekt sind.

Alle getesteten Endpoints

Grün = Voller Zugriff
Gelb = Zugriff nur auf eigene Posts
Rot = Kein Zugriff

Aktion	**User**	**Admin**	**Nicht Authentifizierter User**
GET BY ID	Zugriff auf eigene Posts	Zugriff auf alle Posts, auch von anderen	Kein Zugriff auf eigene Posts
GET All	Zugriff auf alle Posts	Zugriff auf alle Posts	Zugriff auf alle Posts
POST	Kann neue Posts erstellen	Kann Posts für andere Accounts erstellen	Kein Recht, Posts zu erstellen
UPDATE	Kann eigene Posts aktualisieren	Kann Posts anderer Benutzer aktualisieren	Kein Recht, Posts zu aktualisieren
DELETE	Kann eigene Posts löschen	Kann Posts anderer Benutzer löschen	Kein Recht, Posts zu löschen

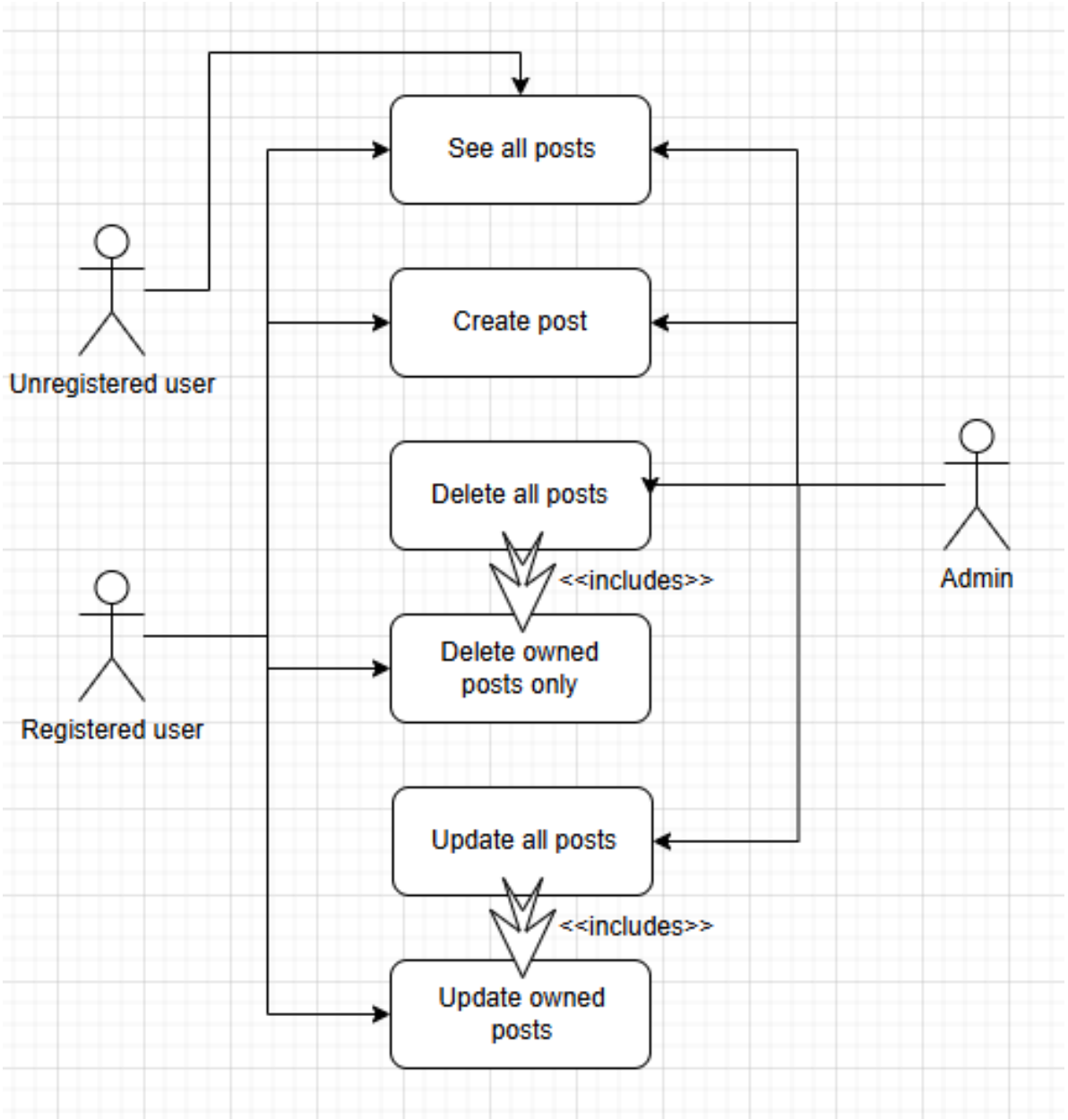
Erwartete Resultate:

Statuscode	**Erwartetes Resultat**
200	Wenn etwas erfolgreich war
201	Wenn etwas erfolgreich erstellt wurde
401	User ist unautorisiert
404	Wenn die Seite nicht gefunden werden kann

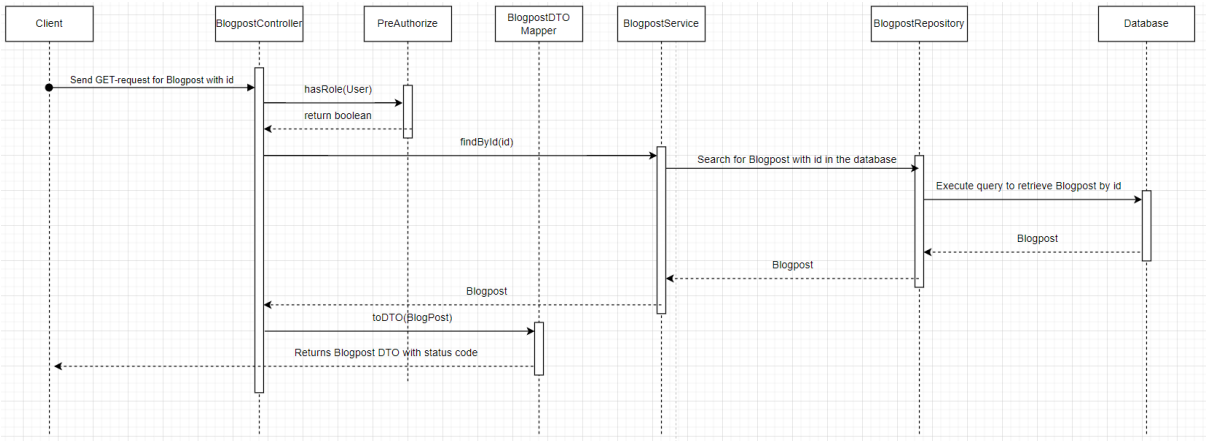
4.1. Use-Case Beschreibung

Actor:	**Registered User**
Beschreibung:	Der registrierte User möchte einen Blogpost erstellen
Voraussetzungen:	● User ist authentifiziert ● User ist autorisiert, um Post zu veröffentlichen
Nachbedingungen:	● Der User sieht den neuen Blogpost auf der Hauptseite
Normaler Verlauf:	● User loggt sich ein ● User klickt auf “New Blogpost” button ● Form öffnet sich und der User füllt alle Felder aus ● User klickt auf “Post” und der Blogpost wird veröffentlicht
Alternativer Kurs.:	● Falls der Blogpost als unvollständig zu veröffentlichen gilt, markiere die fehlenden Felder und teile das dem User mit.
Ausnahmen:	Keine

5. Use-Case Diagramm



6. Sequence Diagramm



7. Swagger