OURSPACE

GROUPS

ük-223 Dokumentation

Thema: **Multiuser-Applikationen**

Autoreninformationen

Autoren: Leona Istrefi, Ramis Yousaf

Inhaltsverzeichnis

1. Ausgangslage

Wir wurden beauftragt, die Social-Media-Website “OurSpace” zu entwickeln. Unser Teil im Rahmen dieses Projektes war es, Groups zu implementieren. Auf dieser Website sollen mehrere User Blog-Posts erstellt werden können. Die Posts sind zugänglich für die Öffentlichkeit. Die Page wird verwaltet von Admins, die Kategorien erstellen und verwalten können. Diese Kategorien können dann von Usern zu Blog-Posts zugeteilt werden. User können Teil einer Gruppe werden.

2. Auftrag

Ziel dieses Projekt ist es, eine Full-Stack-Komponente mithilfe von React, SpringBoot und PostgresSQL zu erstellen, die die unten aufgeführten Bedingungen erfüllen. Dabei wird auf Multiuser-fähigkeit und Sicherheit sowie Dokumentation der Arbeit geachtet. Als Vorgabe erhalten Sie ein funktionierendes rudimentäres Full-Stack Projekt. Dieses Projekt enthält Funktionalitäten, um bestehende User einzuloggen und ermöglicht das Erstellen, Bearbeiten und Löschen von Usern. Login Funktionalität sowie einfaches Routing wurden ebenfalls implementiert.

2.1. Pflicht-Anforderungen

2.1.1. Funktionale Anforderungen:

User Rollen & 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

- Im Minimum enthält die Applikation:
    - Login-Page, die öffentlich zugänglich ist
    - Eine öffentlich zugängliche Homepage
    - 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 autorisierten 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.

3. Gruppenspezifische Aufgabe

3.1. Groups

- Erstellen Sie ein Group Model, das Informationen über eine Gruppe von Usern enthält (Mitglieder, Gruppenname, Motto, Logo).
- Jeder User kann nur maximal in einer Gruppe gleichzeitig Mitglied sein.
- Erstellen Sie Endpoints in Ihrer Applikation, um typische CRUD-Operationen an Groups durchzuführen.
- Erstellen Sie einen Endpoint, der alle Mitglieder einer Gruppe aufgelistet.
- Benutzen Sie dafür Pagination.
- Nur ein Administrator soll eine Gruppe erstellen, bearbeiten oder löschen können.
- Nur Administratoren oder Mitglieder einer jeweiligen Gruppe können Informationen der Gruppe aufrufen

4. Nicht funktionale Anforderungen:

4.1. Implementation

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

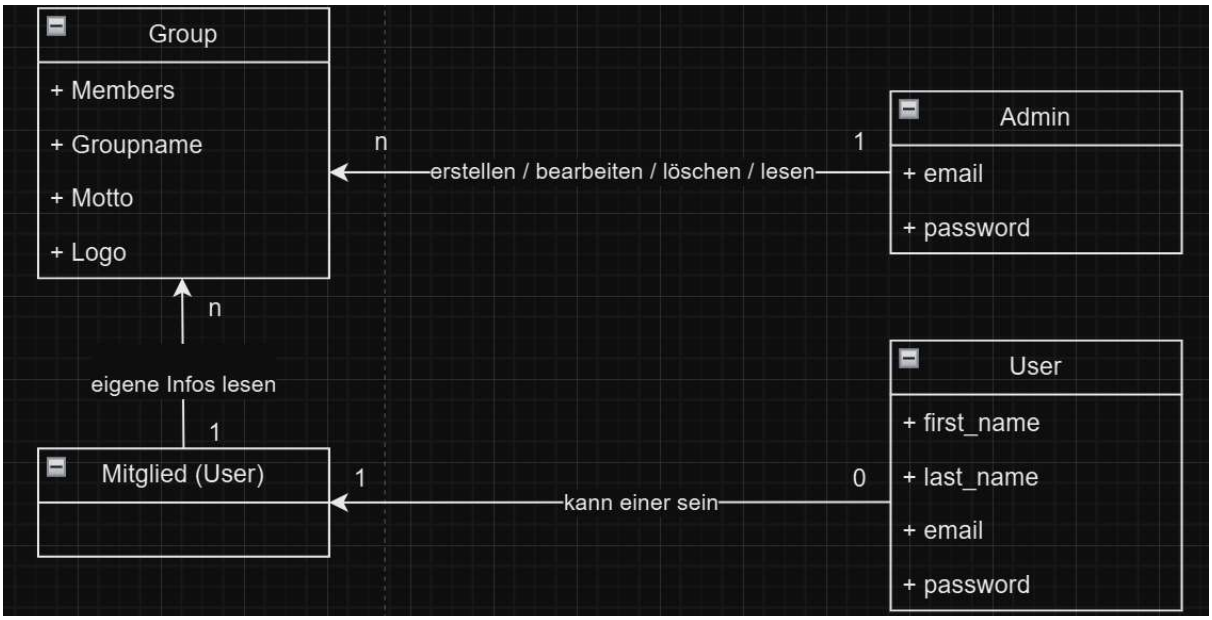
4.2. Testing

- 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 Errorfall wird getestet.
    - Für diese Fälle werden Use-Cases nach UML-Standard beschrieben.

4.3. Multiuserfähigkeit

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

5. Domänenmodell



5.1. User:

- Jeder **User** kann einer **Gruppe** beitreten (über die **Mitglied**-Entität), aber nur einer Gruppe gleichzeitig.
- Ein **User** hat Attribute wie `Vorname`, `Nachname`, `Email`, und `Passwort`.

5.2. Mitglied:

- Ein **Mitglied** ist ein **User**, der zu einer Gruppe gehört.
- Ein **Mitglied** kann nur in einer Gruppe sein und hat das Recht, eigene Gruppeninformationen zu sehen.

5.3. Group:

- Eine **Gruppe** hat mehrere **Mitglieder** und Informationen wie `Gruppenname`, `Motto` und `Logo`.
- Nur ein **Admin** kann Gruppen erstellen, bearbeiten, löschen und lesen.

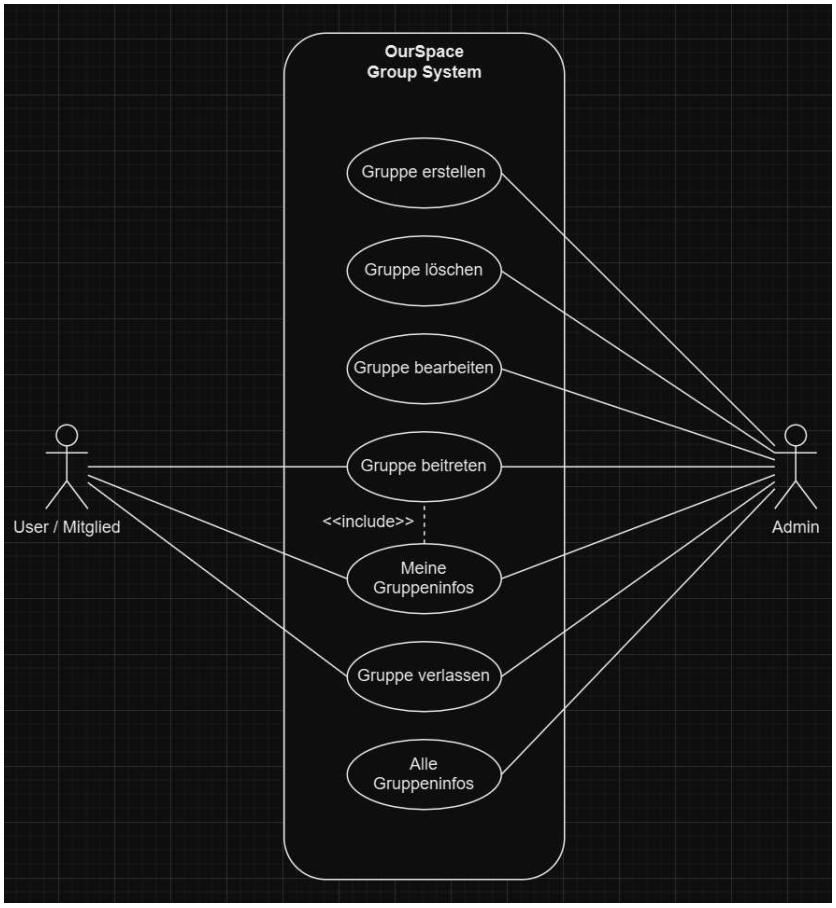
5.4. Admin:

- Der **Admin** verwaltet Gruppen und hat erweiterte Rechte: Er kann Gruppen erstellen, bearbeiten, löschen und deren Informationen lesen.
- Der **Admin** kann mehrere Gruppen verwalten.

5.5. Beziehungen:

- **User** können nur als **Mitglied** einer **Gruppe** beitreten und ihre eigenen Informationen sehen.
- **Admins** haben volle Kontrolle über die Verwaltung von Gruppen.

6. Use-Case-Diagramm



Dieses Diagramm zeigt, welche Aktionen ein User (Mitglied) und ein Admin in unserem Gruppen-Management-System durchführen können.

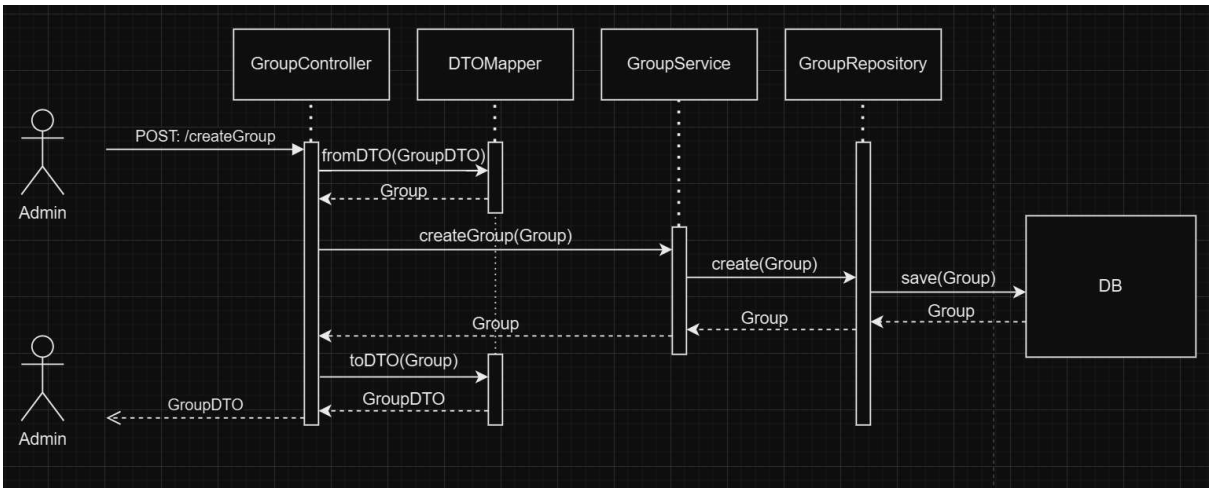
Admin:

- **Gruppen erstellen**: Der Admin kann neue Gruppen erstellen.
- **Gruppen bearbeiten**: Der Admin kann bestehende Gruppen verändern.
- **Gruppen löschen**: Der Admin kann Gruppen entfernen.
- **Alle Gruppeninfos anzeigen**: Der Admin hat Zugriff auf die Informationen aller Gruppen.

User / Mitglied:

- **Gruppe beitreten**: Ein User kann einer Gruppe beitreten.
- **Eigene Gruppeninfos anzeigen**: Ein User kann nur die Infos seiner eigenen Gruppe sehen (dargestellt durch die <<include>>-Beziehung, da ein User einer Gruppe beitreten muss, um die Infos zu sehen).
- **Gruppe verlassen**: Ein User kann eine Gruppe verlassen.

7. Sequenz-Diagramm



In diesem Sequenzdiagramm wird der Prozess beschrieben, wie ein Admin eine neue Gruppe erstellt:

1. **Admin** sendet eine POST-Anfrage an den **GroupController** mit den Gruppendaten (z.B. Gruppenname, Motto, Logo).
2. Der **GroupController** ruft den **DTOMapper** auf, um die empfangenen Daten von einem **GroupDTO** in eine **Group**-Entität umzuwandeln.
3. Der **GroupController** leitet die **Group**-Entität an den **GroupService** weiter, um die Gruppe zu erstellen.
4. Der **GroupService** ruft das **GroupRepository** auf, das die Gruppe in der Datenbank speichert.
5. Nach erfolgreicher Speicherung gibt das **GroupRepository** die gespeicherte **Group**-Entität zurück an den **GroupService**.
6. Der **GroupService** nutzt den **DTOMapper**, um die **Group**-Entität wieder in ein **GroupDTO** umzuwandeln.
7. Schliesslich sendet der **GroupController** die **GroupDTO** als Antwort zurück an den **Admin**, um zu bestätigen, dass die Gruppe erfolgreich erstellt wurde

8. Use-Case Beschreibung

Das ist die Use-Case Definition aus der Sicht des Admins für das Erstellen einer neuen Gruppe im OurSpace Group System:

Actor:	Admin
Description:	Der Admin möchte eine neue Gruppe erstellen, indem er Informationen wie Gruppenname, Motto und Logo angibt.
Preconditions:	• Der Admin muss eingeloggt sein und die Berechtigung haben. • Der Gruppenname muss einzigartig sein.
Postconditions:	• Eine neue Gruppe wird erstellt. • Die Gruppe ist sichtbar und der Admin kann ihr beitreten.

Normal Course:	1. Der Admin navigiert zur Gruppenerstellungs-Seite.
	2. Er gibt die notwendigen Informationen (Gruppenname, Motto, Logo) ein.
	3. Der Admin klickt auf “Gruppe erstellen”.
	4. Das System prüft den Gruppennamen:
	  ○ Wenn der Name bereits existiert, erscheint eine Fehlermeldung(siehe Exceptions).
	  ○ Wenn der Name eindeutig ist, wird die Gruppe erstellt.
	5. Das System zeigt eine Bestätigung an.
	6. Die neue Gruppe ist in der Liste sichtbar.

Alternative Courses:	1. **Gruppenname existiert bereits:**
	  ○ Der Admin gibt einen neuen Namen ein und bestätigt. Das System prüft den neuen Namen, und der Ablauf wird fortgesetzt.
	2. **Fehlende Eingaben:**
	  ○ Das System zeigt eine Fehlermeldung, wenn Informationen wie Gruppenname oder Motto fehlen.

Exceptions:	• Wenn der Gruppenname bereits existiert, fordert das System den Admin auf, einen anderen Namen zu wählen.

9. Testing Strategie

In unserem Projekt werden wir verschiedene Testarten verwenden. Alle Tests werden in unserer Git Repository abgelegt.

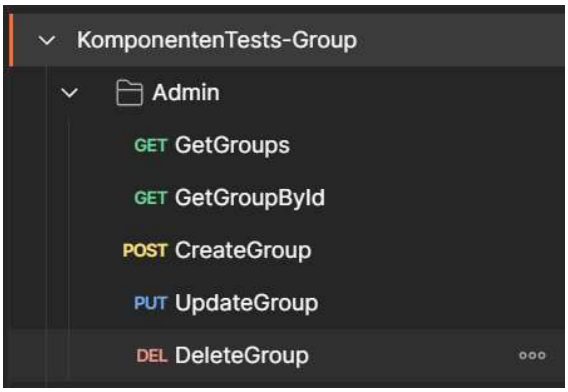
9.2. Funktionale Tests

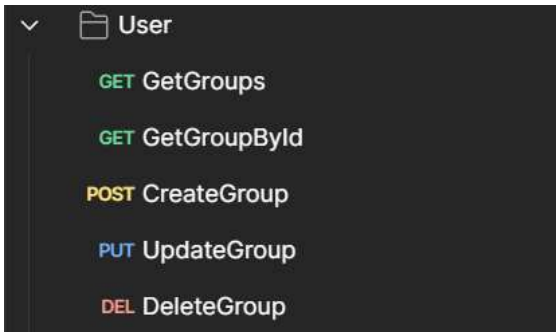
9.2.1 E2E Tests mit Cypress

Die End-to-End (E2E) Tests werden mit dem Tool Cypress durchgeführt. Diese Tests simulieren den gesamten Prozess, den ein Admin durchläuft, um eine neue Gruppe zu erstellen (siehe Use-Case-Beschreibung). Zusätzlich werden verschiedene Simulationen mit unterschiedlichen Usern durchgeführt, um sicherzustellen, dass die Rollen und Berechtigungen im gesamten System korrekt umgesetzt sind.

9.2.2 Komponententests

Unsere Komponententests werden mit Postman durchgeführt, wobei die einzelnen Endpoints getestet werden. Dabei überprüfen wir die Rechte und Berechtigungen von Usern und Admins, um sicherzustellen, dass Rollen und Autoritäten korrekt zugeordnet sind. Diese Tests helfen uns zu bestätigen, dass sich die Applikation in verschiedenen Szenarien mit unterschiedlichen Benutzerrollen korrekt verhält.





9.3. Nicht-funktionale Tests

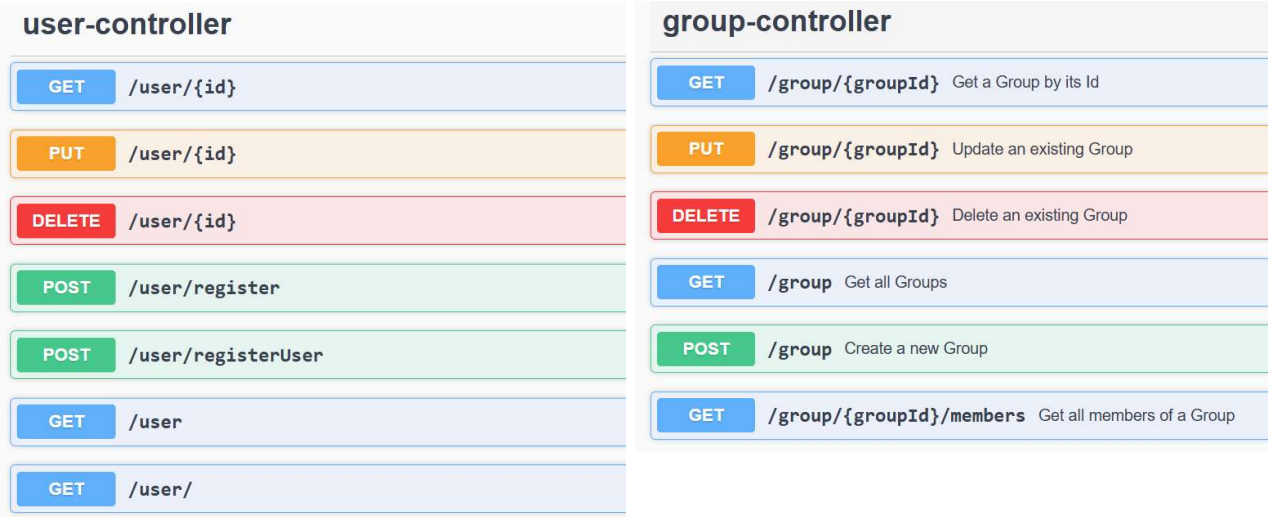
9.3.1 Performance mit Lighthouse

Zur Überprüfung der Performance nutzen wir das Tool Lighthouse. Mit diesen Tests analysieren wir die Leistung der Applikation und identifizieren Bereiche, in denen Verbesserungen erforderlich sind, falls die Performance nicht unseren Erwartungen entspricht.

9.4. Warum Komponententests(Postman) statt JUnit-Tests?

In diesem Projekt haben wir uns entschieden, Komponententests mit Postman durchzuführen, da uns diese Testmethode vertrauter ist. Komponententests bieten eine einfache Möglichkeit, die Endpoints direkt zu testen und ermöglichen uns, die Logik und das Verhalten detailliert zu überprüfen. Obwohl JUnit für Unit-Tests in Java beliebt ist, haben wir aufgrund unserer Erfahrung mit Postman diese Methode bevorzugt, um schneller und effizienter Ergebnisse zu erzielen.

10. Swagger



(war bereits vorhanden)

10.1 Beschreibung der Endpoints des Group-Controllers

GET /group/{groupId}: Ruft eine Gruppe anhand ihrer ID ab. Dieser Endpunkt wird verwendet, um detaillierte Informationen zu einer spezifischen Gruppe zu erhalten.

PUT /group/{groupId}: Aktualisiert eine bestehende Gruppe mit neuen Informationen. Nur Administratoren können diesen Endpunkt verwenden, um Änderungen an Gruppen vorzunehmen.

DELETE /group/{groupId}: Löscht eine Gruppe basierend auf ihrer ID. Dieser Endpunkt ist nur für Administratoren zugänglich und entfernt eine Gruppe dauerhaft aus der Datenbank.

GET /group: Gibt eine Liste aller vorhandenen Gruppen zurück. Administratoren und Benutzer können diesen Endpunkt verwenden, um eine Übersicht über alle Gruppen zu erhalten.

POST /group: Erstellt eine neue Gruppe. Nur Administratoren dürfen diesen Endpunkt nutzen, um neue Gruppen in das System einzufügen.

GET /group/{groupId}/members: Listet alle Mitglieder einer bestimmten Gruppe auf. Diese Liste ist durch Pagination begrenzt und ist für Administratoren sowie für Mitglieder der Gruppe einsehbar.