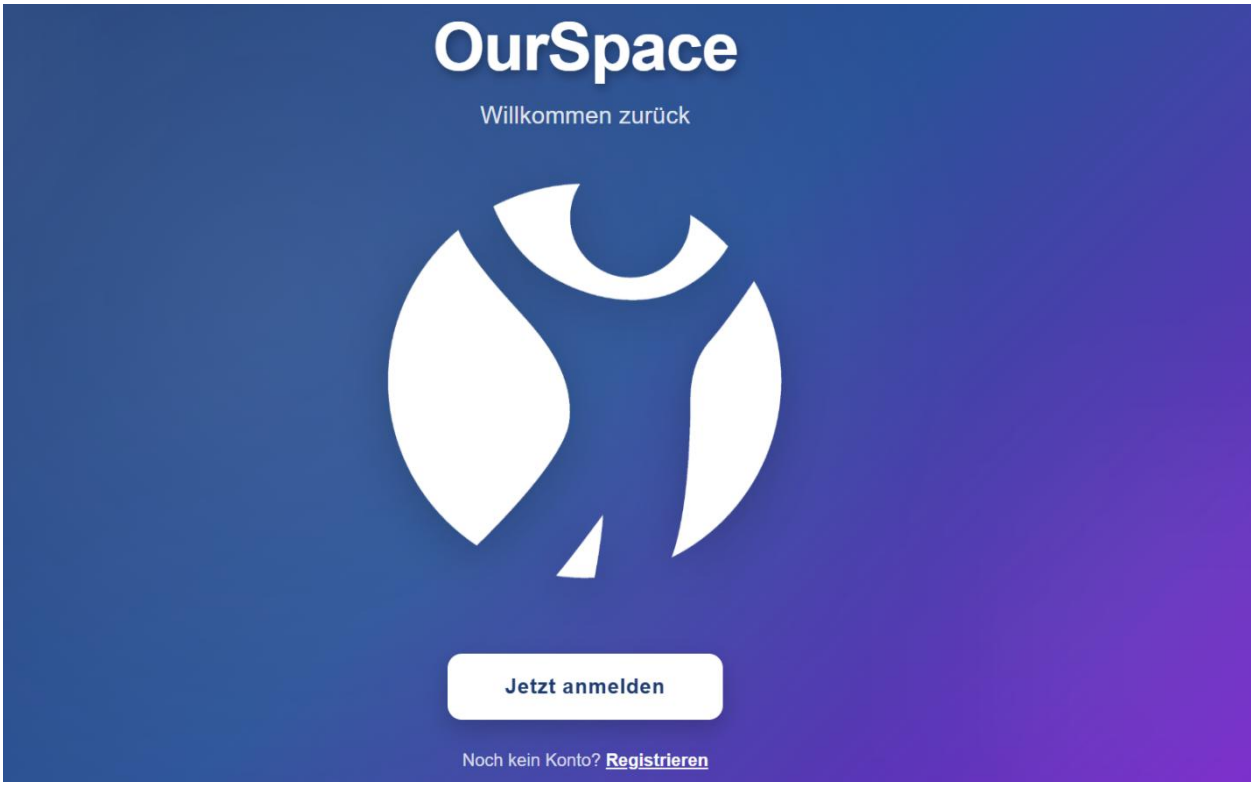


Projektauftrag üK 223

Thema: Modul 223 – Multi-User-Applikationen objektorientiert realisieren

Dokumentinformationen

Dateiname: OurSpace-Dokumentation_Daphne-David-Gino.pdf
Speicherdatum: 16.01.2026

Autoreninformationen

Autor: Daphne McNamara, David Tarlos und Gino Wenger

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Sinn und Zweck

Das vorliegende Dokument beschreibt die Dokumentation zum Projektauftrag für den üK 223. Das Ziel des Projektes ist die Entwicklung einer Multi-User-fähigen Full-Stack-Komponente für die Social-Media-Plattform «OurSpace».

Im Rahmen des Projekts implementiert unsere Gruppe die Image Gallery, welche es Benutzern erlaubt, Bilder zu veröffentlichen, zu bearbeiten, zu löschen und zu liken.

2 Domänenmodell

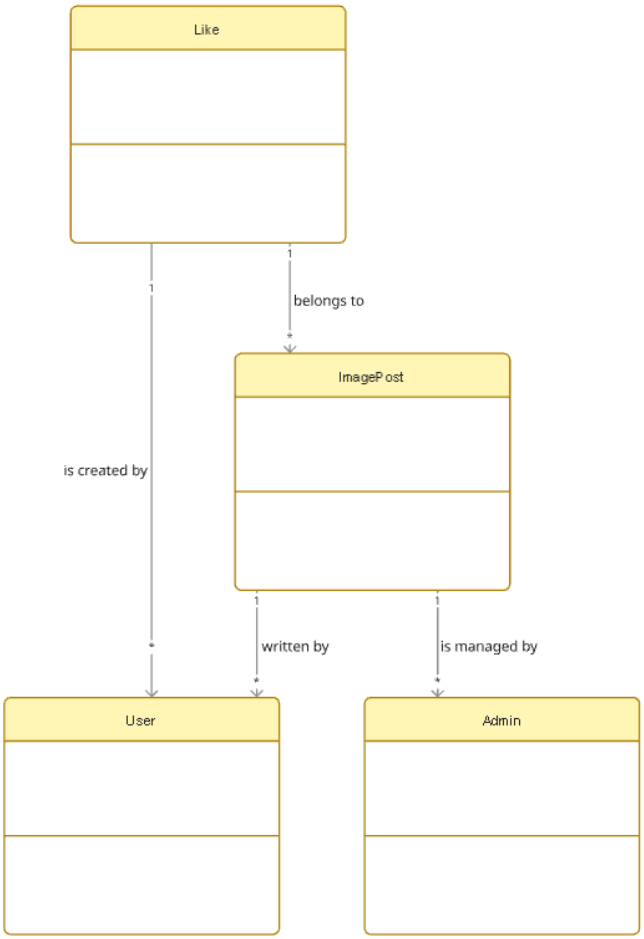


Abbildung 1: Domain Model Image Gallery

Das Domänenmodell zeigt die zentralen fachlichen Objekte der Anwendung sowie deren Beziehungen zueinander. Es dient dem grundlegenden Verständnis der Domäne und verzichtet bewusst auf technische Details oder Implementierungsaspekte.

3 Entity Relationship Diagramm (ERD)

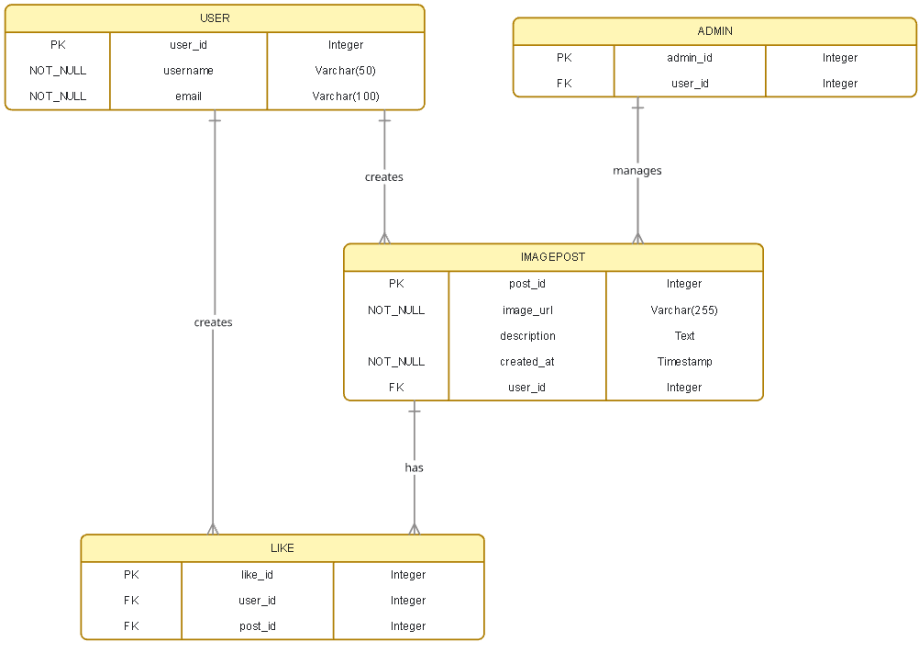


Abbildung 2: ERD Image Gallery

Das Entity Relationship Diagramm beschreibt die relationale Struktur der Datenbank. Es zeigt die verwendeten Tabellen, deren Attribute sowie die Primär- und Fremdschlüssel zur Abbildung der Beziehungen zwischen den Entitäten.

4 Use-Case-Beschreibungen (UC1-UC5)

4.1 UC1: User erstellt einen Image-Post

Actor(s)	Eingeloggter User
Description	Der User erstellt einen neuen Beitrag in der “meine Posts”, indem er eine Bild-URL und optional eine Beschreibung hinzufügt.
Preconditions	- Der User ist eingeloggt. - Eine Bild-URL ist vorhanden und gültig. - Optional: Beschreibung ist vorhanden (Validierung erforderlich).
Postconditions	- Der neue Image-Post wird in der Datenbank gespeichert. - Der Image-Post wird in der Gallery angezeigt. - Danach erschient eine Erfolgsmeldung - Der Image-Post erscheint in „Meine Posts“.
Normal Course	1. Der User loggt sich ein. 2. Der User wird zur Gallery-Seite weitergeleitet. 3. Der User klickt auf „Meine Posts“. 4. Der User klickt auf „+“ (unten rechts), um einen neuen Post zu erstellen. 5. Der User gibt die Bild-URL und optional die Beschreibung ein. 6. Der User klickt auf „Speichern“. 7. Das System validiert die Eingaben. 8. Das System speichert den Post in der Datenbank. 9. Der neue Post wird sowohl in „Meine Posts“ als auch in der allgemeinen Gallery angezeigt.
Alternative Courses / Exceptions	- Bild-URL ist ungültig → Fehlermeldung anzeigen. - Beschreibung überschreitet die maximale Länge → Fehlermeldung anzeigen. - Pflichtfelder sind leer → Fehlermeldung anzeigen.

Tabelle 1: UC1

4.2 UC2: User bearbeitet eigenen Image-Post

Actor(s)	Eingeloggter User (eigene Posts), Admin (alle Posts)
Description	User ändert Bild-URL oder Beschreibung eines eigenen Image-Posts über einen Dialog.
Preconditions	- User ist eingeloggt - Post existiert - Eingeloggter User: Post gehört dem User - Admin: darf beliebige Posts bearbeiten
Postconditions	- Änderungen werden in der DB gespeichert - Gallery zeigt aktualisierte Daten - Erfolgsmeldung wird angezeigt
Normal Course	1. User navigiert zu "Meine Posts" (/gallery/my-posts) 2. User klickt auf das Bearbeiten-Icon (Stift) eines Posts 3. System öffnet einen Dialog mit den aktuellen Daten (Bild-URL, Beschreibung) 4. User ändert Bild-URL und/oder Beschreibung 5. User klickt auf "Speichern" 6. System validiert die Eingaben 7. System speichert die Änderungen 8. System schliesst den Dialog 9. System zeigt Erfolgsmeldung: "Post erfolgreich aktualisiert" 10. System aktualisiert die Post-Liste
Alternative Courses / Exceptions	- Bild-URL ist leer → Fehlermeldung: "Bild-URL ist ein Pflichtfeld" - Bild-URL ist ungültig (kein http/https) → Fehlermeldung: "Bitte gib eine gültige URL ein (z.B. https://example.com/bild.jpg)" - Beschreibung > 200 Zeichen → Fehlermeldung: "Beschreibung darf maximal 200 Zeichen lang sein"

Tabelle 2: UC2

4.3 UC3: User löscht eigenen Image-Post

Actor(s)	Eingeloggter User (eigene Posts), Admin (alle Posts)
Description	User löscht eigenen Image-Post aus der Galerie nach Bestätigung.
Preconditions	- User ist eingeloggt - Post existiert - Eingeloggter User: Post gehört dem User - Admin: darf beliebige Posts löschen
Postconditions	- Post ist aus der DB gelöscht - Gallery zeigt Post nicht mehr - Erfolgsmeldung wird angezeigt
Normal Course	1. User navigiert zu "Meine Posts" (/gallery/my-posts) 2. User klickt auf das Löschen-Icon (Mülleimer) eines Posts 3. System öffnet Bestätigungsdialog mit: - Titel: "Post löschen?" - Text: "Bist du sicher, dass du diesen Post löschen möchtest? Diese Aktion kann nicht rückgängig gemacht werden." 4. User klickt auf "Löschen" 5. System löscht den Post 6. System schliesst den Dialog 7. System zeigt Erfolgsmeldung: "Post erfolgreich gelöscht" 8. System aktualisiert die Post-Liste
Alternative Courses / Exceptions	- User klickt "Abbrechen" → Dialog wird geschlossen, Post bleibt bestehen - User klickt ausserhalb des Dialogs → Dialog wird geschlossen, Post bleibt bestehen - Server-Fehler beim Löschen → Fehlermeldung: "Fehler beim Löschen des Posts"

Tabelle 3: UC3

4.4 UC4: Eingeloggte User sehen Image-Posts mit Pagination

Actor(s)	Eingeloggter User, Admin
Description	User kann alle Image-Posts in einer Galerie ansehen, mit Pagination, Sortierung und Filterung nach Autor.
Preconditions	- User ist eingeloggt
Postconditions	- Actor sieht aktuelle Image-Posts- Ausgewählte Sortierung/Filter werden angewendet
Normal Course	1. User wird zur Galerie navigiert (/gallery) 2. System lädt alle Posts und Autorenliste 3. System zeigt Header mit: - Titel "Galerie" - Anzahl Bilder in der Sammlung - Tabs: "Alle Posts" / "Meine Posts" 4. System zeigt Filter-Bereich mit: - Sortierungs-Dropdown (Standard: "Neueste zuerst") - Autor-Filter-Dropdown (Standard: "Alle Autoren") 5. System zeigt Posts als Karten-Grid (12 Bilder pro Seite) mit: - Bild - Beschreibung - Erstellungsdatum - Like-Button 6. System zeigt Pagination am Seitenende 7. User kann durch Seiten navigieren, sortieren oder filtern
Alternative Courses / Exceptions	- Keine Posts vorhanden → Empty State: "Noch keine Posts vorhanden" mit Hinweis "Sei der Erste und teile ein Bild!" - Fehler beim Laden → Error-Alert mit Fehlermeldung und "Erneut versuchen"-Button

Tabelle 4: UC4

4.5 User liked oder entfernt Like von einem Image-Post

Actor(s)	Eingeloggter User
Description	User kann Likes auf Image-Posts setzen oder entfernen (Toggle-Funktion).
Preconditions	- User ist eingeloggt - Post existiert
Postconditions	- Like-Status wird in der DB aktualisiert - Like-Icon zeigt aktuellen Status an
Normal Course	1. User befindet sich in der Galerie (/gallery) 2. User bewegt Maus über das Herz-Icon (Tooltip: "Gefällt mir") 3. User klickt auf das Herz-Icon eines Posts 4. System sendet Toggle-Request an Backend (POST /api/image-posts/{id}/like) 5. System aktualisiert den Post mit der Server-Antwort 6. Like-Icon wechselt den Status: - Nicht geliked → Geliked: Herz wird ausgefüllt (rot) - Geliked → Nicht geliked: Herz wird nur umrandet (grau)
Alternative Courses / Exceptions	- Netzwerkfehler / Server nicht erreichbar → Fehler wird in Konsole geloggt, Icon-Status bleibt unverändert - User ist nicht mehr eingeloggt (Token abgelaufen) → Backend gibt 401 zurück, Fehler wird geloggt

Tabelle 5: UC5

5 Use-Case-Diagramm

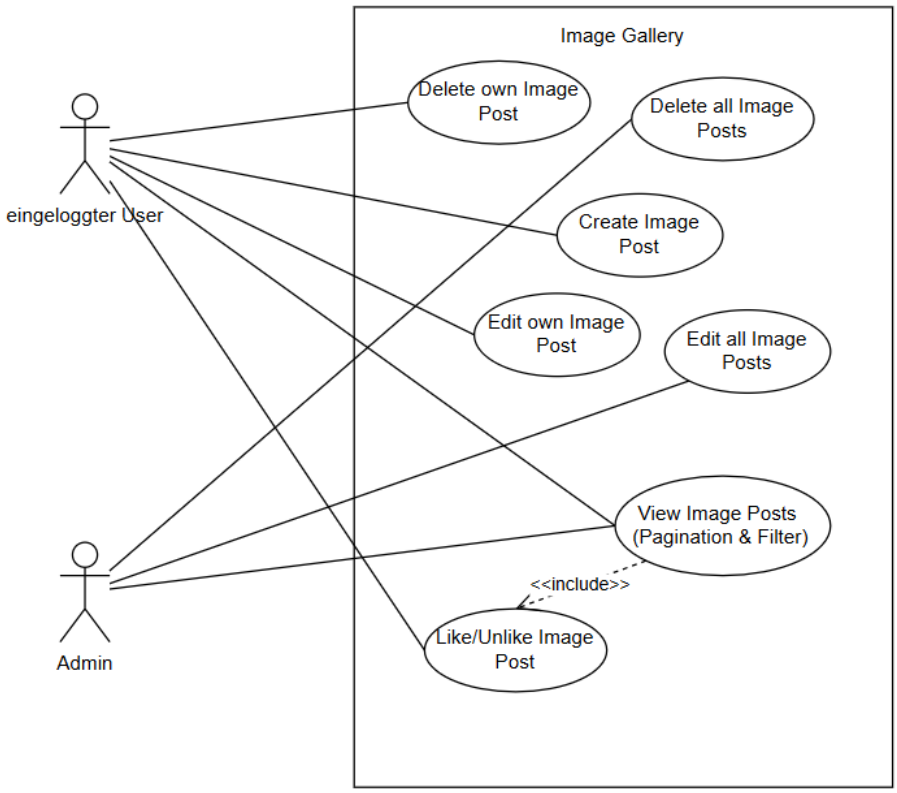


Abbildung 3: Use-Case-Diagramm Image Gallery

Das Use-Case-Diagram gibt einen Überblick über die vom System bereitgestellten Funktionen und zeigt, welche Benutzerrollen mit welchen Use Cases interagieren können. Es dient als Übersicht der funktionalen Anforderungen.

Auf der linken Seite sieht man die external Roles: eingeloggter User und Admin.

Eingeloggter User kann:

- Image Posts erstellen
- Eigene Posts bearbeiten und löschen
- Image Posts anderer User ansehen, liken oder entliken

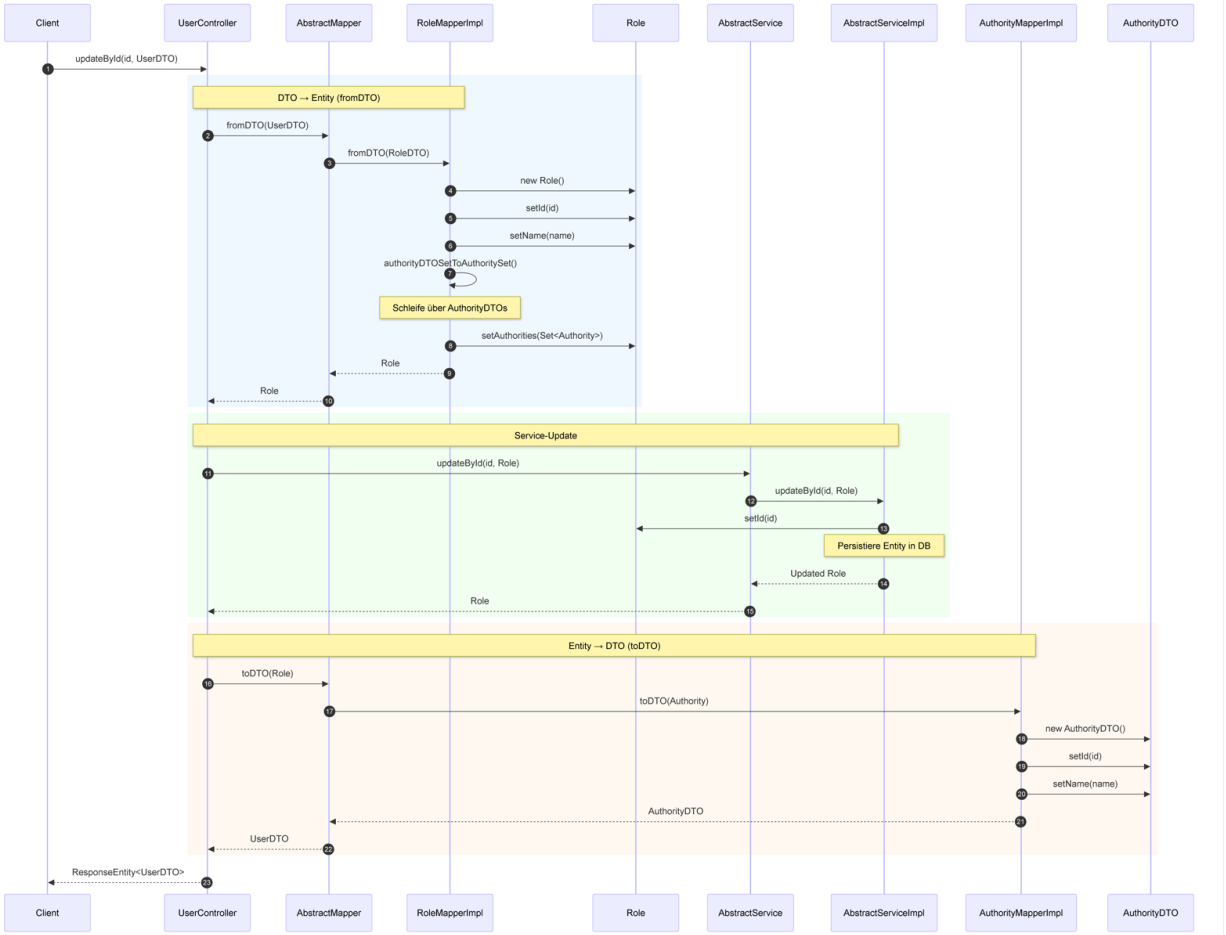
Admin kann:

- Alle vorhandenen Image Posts ansehen, bearbeiten oder löschen

Jedoch kann der Admin keine eigene Image Posts erstellen.

Der Use Case «Like/Unlike Image Post» inkludiert den Use Case «View Image Posts», weil ein Image Post zuerst angezeigt werden muss, bevor er geliked oder entliked werden kann.

6 Sequenzdiagramm



Der Ablauf beschreibt die Aktualisierung von Benutzerrollen im System. Eine Anfrage mit der Benutzer-ID geht vom Client an den UserController, der die Daten in ein DTO umwandelt. Der AbstractMapper leitet das DTO an den RoleMapperImpl weiter, wo die Rolle mit ihren Eigenschaften (ID, Name, Autoritäten) erstellt wird. Die Autoritäts-DTOs werden in einer Schleife durchlaufen und in AuthoritySet-Objekte umgewandelt. Der AbstractService übergibt die aktualisierten Rollendaten an den AbstractServiceImpl, der die Änderungen in der Datenbank speichert. Die Antwort läuft dann den gleichen Weg zurück: vom AbstractServiceImpl über den RoleMapperImpl zum UserController, der das Ergebnis als Entity-UserDTO an den Client schickt.

7 Testing-Strategie

7.1 Getestete Endpoints

Endpoint	UC	Was wird getestet?	Erwartetes Ergebnis	Tool
POST `/api/image-posts`	UC1	Eingeloggter User öffnet Gallery, füllt Bild-URL und Beschreibung aus und klickt auf «Submit»	Neuer Post erscheint in Gallery, Erfolgsmeldung	Cypress
POST `/api/image-posts`	UC1	Leere Bild-URL oder ungültige Beschreibung	Formular zeigt Validierungsfehler, «Submit»-Button disabled	Cypress
POST `/api/image-posts`	UC1	Uneingeloggter User versucht Post zu erstellen	http 401 Unauthorized	Postman
PUT `/api/image-posts/{id}`	UC2	User bearbeitet eigenen Post	Post wird gespeichert, Erfolgsmeldung	Cypress
PUT `/api/image-posts/{id}`	UC2	User versucht fremden Post zu bearbeiten	Button nicht sichtbar / Fehler 403	Cypress/Postman
PUT `/api/image-posts/{id}`	UC2	Admin bearbeitet fremden Post	HTTP 200, Post aktualisiert	Postman
DELETE `/api/image-posts/{id}`	UC3	User löscht eigenen Post	Post verschwindet, Erfolgsmeldung	Cypress
DELETE `/api/image-posts/{id}`	UC3	User versucht fremden Post zu löschen	Button nicht sichtbar / Fehlermeldung, Status 403	Cypress/Postman
DELETE `/api/image-posts/{id}`	UC3	Admin löscht Post	Post gelöscht, HTTP 200	Postman
GET `/api/image-posts`	UC4	User sieht Gallery mit Pagination	12 Posts pro Seite korrekt angezeigt	Cypress
POST `/api/image-posts/{id}/like`	UC5	User liked fremden Post	Like-Zähler aktualisiert, Icon wechselt	Cypress
POST `/api/image-posts/{id}/like`	UC5	User versucht eigenen Post zu liken	Fehlermeldung / Button nicht verfügbar	Cypress

Tabelle 6: getestete Endpoints

7.2 Ausführlicher Test-Case

Use Case: UC1: User erstellt einen Image Post

Endpoint:

POST

```
/api/image-posts
```

Actor(s): Eingeloggter User, Admin, Nicht eingeloggter User

Normal Course:

1. Eingeloggter User öffnet Image Gallery
2. Klick auf “Create Image Post”
3. Eingabe gültiger Bild-URL und Beschreibung
4. Klick auf «Submit»

Erwartetes Ergebnis: Neuer Post erscheint in Gallery, Erfolgsmeldung angezeigt

Alternative/ Edge Cases:

- Leere Bild-URL -> Validierungsfehler, «Submit»-Button disabled
- Beschreibung länger als 200 Zeichen -> Validierungsfehler, Fehlermeldung
- Ungültige URL -> Fehlermeldung

Negative Cases (Rollen & Berechtigung):

- Nicht eingeloggter User versucht Post zu erstellen -> HTTP 401 Unauthorized
- Admin versucht eigenen Post zu erstellen -> Button nicht vorhanden, Aktion nicht möglich

Test-Tools:

- Cypress für interaktive Frontend-Tests
- Postman für API-Tests, insbesondere Zugriffsrechte