Modul 223: Multi-User-Applikationen objektorientiert realisieren

Image Gallery, Projektdokumentation

Autoren:

Laux Stefan (SIX, 2. Lehrjahr)

Lusti Sarina (Google, 2. Lehrjahr)

Seiler Yannis (Google, 2. Lehrjahr)

Betreut durch:

Widmer Luca (Noser Young AG, Projektleiter)

Inhaltsverzeichnis

Overview

Ziel

Umsetzen einer Multiuserapplikation mit allgemeinen Grundanforderungen als auch eines gruppenspezifischen Auftrags.
Der gruppenspezifische Auftrag unserer Gruppe ist es, für eine imaginäre Social-Media-Site eine Image Gallery zu erstellen.

Kontext des Auftrages

Es wird eine neue Social-Media-Plattform namens 'OurSpace' entwickelt, welche alle typischen Features moderner sozialer Multiuserapplikationen enthält.
Innerhalb einer Gruppe sollten grundlegende, eher allgemein gehaltene Anforderungen umgesetzt werden, wie auch eine spezifische Subkomponente der Plattform.

Im Falle unseres Teams wäre das eine Image Gallery, welche es ermöglicht, Bild Posts zu erstellen, zu liken, wie auch zu sehen, wer die Posts schon mit einem Like versehen hat.

Zu den Anforderungen der Applikation im Allgemeinen gehört das Erstellen von neuen Usern, die Rechtevergabe, Rollenerstellung und rollenspezifischer Ressourcenzugriff.

Die Anleitung, wie man das Projekt ausführt, findet man im jeweiligen README.md im Backend/Frontend Repository.

Projektanforderungen

Allgemeine Anforderungen

User Rollen und Privilegien

- Admin-Rolle vorhanden
- persönliche Informationen von anderen Usern nur für Admins lesbar
- Admins können User bearbeiten, erstellen, löschen

Frontend

- Login-Page (öffentlich zugänglich)
- Public Homepage
- User Homepage (für eingeloggte User)
- Admin-Page
- mindestens eine gruppenspezifische Page

Security

- Absicherung der REST-Endpoints
    - Sicherstellen von rollenspezifischen Rechten pro Endpoint
- Log-In-limitierte Frontend Bereiche (gewährleisten von Authentifizierung und Autorisierung)

- Rollen-limitierte Frontend Bereiche
- Gewisse Frontend Bereiche sollen nur mit bestimmten Rollen zugänglich sein
- Authentifizierungsmechanismus via JWT

Gruppenspezifische Anforderungen

Kreiere ein Image-Post Modell mit folgenden Vorgaben:

- User kann Bild teilen
    - Informationen über: URL, Autor, Beschrieb, Likes, wer hat geliked
- Endpoints
    - CRUD-Operationen
- Liste aller Bilder von beliebigen Users (nur sichtbar für eingeloggte User)
- Restriktionen
    - Nur der Ersteller oder Admin soll Beiträge anderer User bearbeiten oder löschen können. Ansonsten können User nur Modifikationen an ihren eigenen Posts vornehmen.

High-Level Design

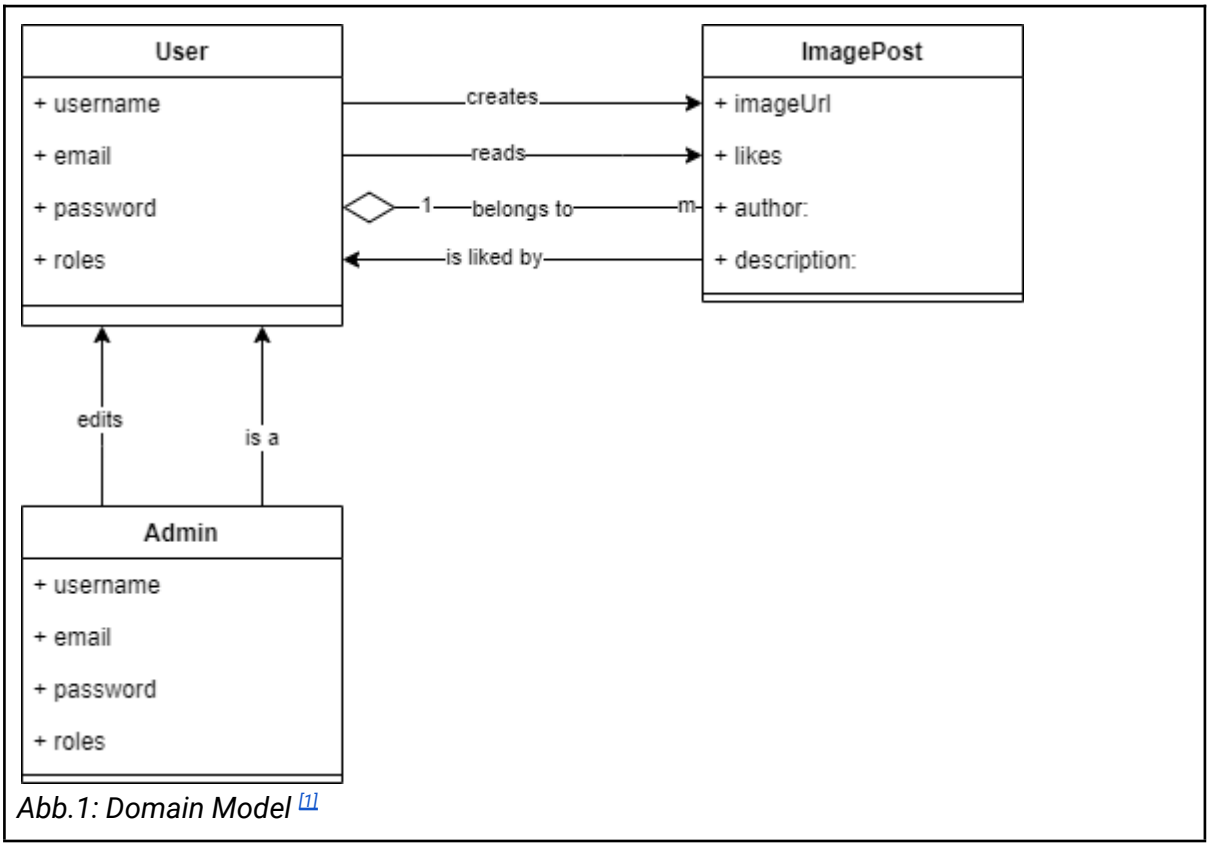


Abb.1: Domain Model [1]

Unsere hauptsächlichen Domänen sind die User von OurSpace, die Image Posts und noch ein spezieller User, welcher die Admin-Rolle hat.

Die User interagieren mit den Image-Posts insofern, dass sie ihre eigenen kreieren, editieren wie auch löschen können. Das Löschen oder Editieren von Posts kann jeweils nur an den eigenen Posts vorgenommen werden, da man nicht in der Lage sein soll, andere Posts zu manipulieren.

Die einzige Interaktion, die mit anderen Posts vorgenommen werden kann, ist das Liken. Nämlich kann man als Benutzer mithilfe eines Like-Buttons einen anderen Post mit “gefällt mir” markieren, was dann im Backend wie auch im Frontend reflektiert wird.
Ausnahmen gelten hierbei für Admin-Users. Diese können nämlich Modifikationen an allen Posts vornehmen, nicht nur an den eigenen, um beispielsweise potenzielle Missstände zu beheben.

Detailliertes Applikationsdesign

Für die Umsetzung unseres Auftrages halten wir uns an folgendes Entity Relationship Diagram.
Zwischentabellen sind nicht aufgeführt, vollständigkeitshalber gilt allerdings zu erwähnen, dass in der Datenbank auch noch folgende Tabellen existieren: `image_like`, `users_role`, `role_authority`.

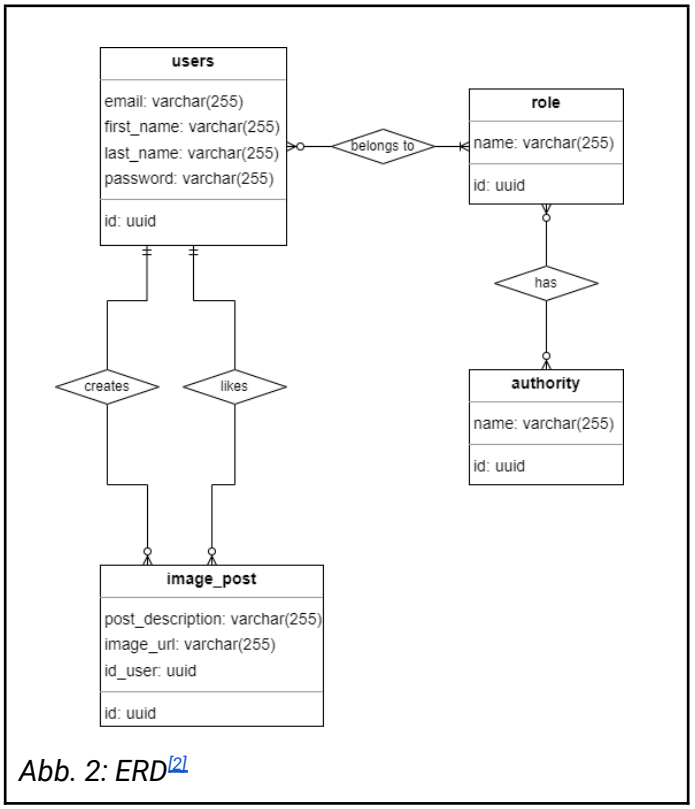


Abb. 2: ERD[2]

Innerhalb unseres Codes sind die Tabellen anhand von Models repräsentiert. Diese haben auch Einschränkungen auf ihren Feldern um Datenbankprinzipien (ACID) so gut wie möglich zu gewährleisten. *(Beispiel ImagePost: imageUrl darf nicht leer sein beim Erstellen.)*

Rechtevergabe

Um die oben erwähnte Sicherheit zu gewährleisten, ist es nötig, verschiedene Rechte zu vergeben, die in bestimmten Rollen vorhanden sind.
Laut Auftragsdefinition ist es besonders wichtig, dass eine Admin-Rolle besteht, welche bestimmte zusätzliche Rechte umfasst. Diese zusätzlichen Rechte wären die folgenden:

- User bearbeiten (Authority: USER_MODIFY)
    - Bezieht sich auf die verschiedenen Informationen eines Users. Dies beinhaltet Vorname, Nachname, E-Mail-Adresse, Passwort und Rollen.
- User löschen (Authority: USER_DELETE)
- User erstellen (Authority: DEFAULT)

Standardmässig hat jeder User DEFAULT Rechte, welche es nicht einfach erlauben, andere Userdaten in irgendeiner Form zu verändern. Das Einzige, was jeder User kann, ist das Erstellen von anderen Usern.

Hat man allerdings auch die Rechte USER_MODIFY und USER_DELETE, so hat man alle Autoritäten eines Admins, was dementsprechend unter die Admin-Rolle fällt.

Dies spiegelt sich wider in der Aufteilung der Rollen und Autoritäten. Die Rollen, die vorhanden sind, haben grundsätzlich die gleichen Namen wie die oben erwähnten Rechte, allerdings sind diese auf die jeweiligen limitiert.

Anders ist das beim Admin, welcher als einzige Rolle alle anderen Autoritäten umfasst.

Backend-Struktur

Die Backend-Struktur hält sich an die REST-Vorgaben durch das Umsetzen des geschichteten Systems.
Um das Einschränken der verschiedenen Autoritäten und Rollen darzustellen, wird folgendes Farbschema fortlaufend verwendet:

Authority	**Role**
DEFAULT	DEFAULT
USER_MODIFY	USER_MODIFY
USER_DELETE	USER_DELETE
<<alle oben erwähnten>>	ADMIN
<<ADMIN + User, der auf seine eigenen Ressourcen zugreift>>	

APIs

Damit das Frontend mit den Image-Posts interagieren kann, sollen verschiedene Endpoints erstellt werden:

GET	**PUT**	**POST**	**DELETE**
getImagePostById getImagePostsByAuthor getAllImagePosts	updateImagePostById	createImagePost	deleteImagePostById

Für die ImagePost GET-Requests wird ausserdem Like-basiertes Sorting implementiert. Ebenfalls wichtig ist, dass die Datentransferobjekte, die das Backend mit den Requests erhält, per `@Validated` Annotation auf ihre Korrektheit geprüft werden.

Die Endpoints für den User sind:

GET	**PUT**	**POST**	**DELETE**
retrieveById retrieveAll	updateById	register registerWithoutPassword	deleteById

Security

Alle Endpoints sind von seitens Backend her geschützt. Dies geschieht einerseits mithilfe von funktionierender Authentifizierung und entsprechender Autorisierung.

Authentifizierung wird gewährleistet durch den bereits implementierten JWT-Mechanismus.

Passwörter werden auf der Datenbank entsprechend Industriestandards verschlüsselt abgespeichert, um so erhöhte Sicherheit zu gewährleisten.

Die **Autorisierung** stellen wir sicher, indem wir jeden Endpoint mittels identitäts- und rollenbasierter Restriktionen schützen.
Für Aktionen, die von Admins als auch Usern, die auf ihre eigenen Ressourcen zugreifen wollen, ausgeführt werden, wird ebenfalls die Annotation `@PreAuthorize` verwendet.
In dieser wird eine Hilfsmethode aufgerufen, die sicherstellt, ob die angefragte Ressource wirklich zum eingeloggten User gehört.

Dokumentation

Die ganzen Endpoints wurden zusätzlich im Backend via Swagger dokumentiert. Diese Dokumentation umfasst eine kurze Beschreibung sowie die Rolle(n) und Autoritäten, welche man braucht, um diese Operation auszuführen. Finden tut man diese Dokumentation bei laufendem Backend unter folgender URL: /docs/swagger-ui/index.html.

Programmfluss anhand eines DELETE-Request Beispiels

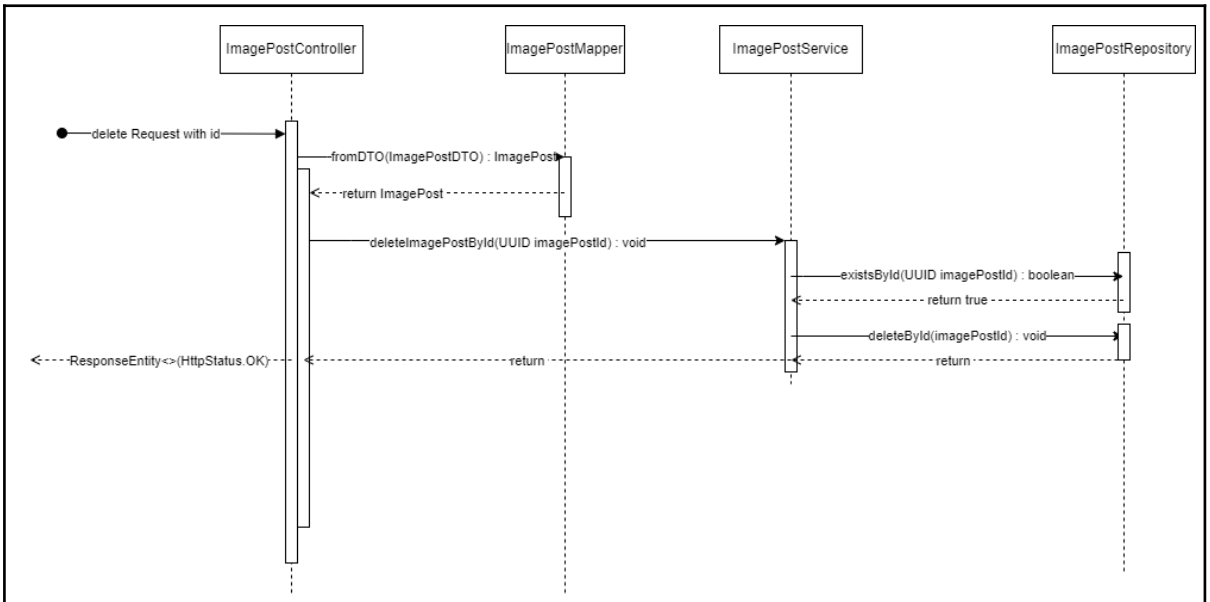


Abb. 3: Sequenzdiagramm[3]

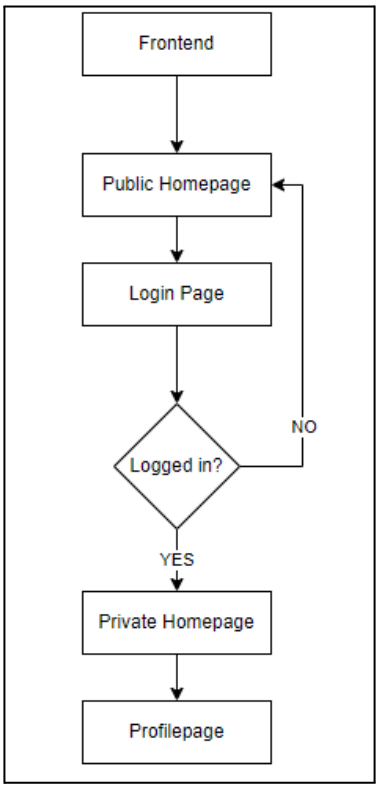
Im obenstehenden Diagramm (Abb. 3) ein DELETE-Request für einen Image-Post zu sehen, welcher von einem gültigen User ausgeführt wird.
Empfangen wird das ImagePostDTO im ImagePostController, wird umgewandelt in eine Entität und an den Service mitsamt der Pfadvariable (ID) übergeben. Nachdem die Überprüfung der Gültigkeit der ID erfolgreich war, wird der Eintrag gelöscht und der erfolgreiche Statuscode an das Frontend zurückgesendet.

Logging

Um Aktionen unserer Applikation zu beobachten, werden bei allen wichtigen Backend Aktionen entsprechende Logs in eine Datei geschrieben.
Die Datei ist zu finden unter `M223_Backend-YaSei-StLau-SaLus/log/log.out`.
Um das Logging zu implementieren, verwendeten wir Log4J2 und hielten jeweils in den Services fest, wenn eine Aktion durchgeführt wurde (Loglevel entsprechend dem Ergebnis der Operation).
Damit die Logs in der Datei `log.out` landen, müssen zusätzliche Konfigurationen in unserer `application.properties` Datei als auch in `logback.xml` vorgenommen werden.

Frontend Struktur

Aufbau der Seite



Die Homepage, welche öffentlich ist, hat fast keinen Inhalt ausser einen Log-in-Button. Auf der Log-in-Seite kann man sich mit seinen Zugangsdaten anmelden und bekommt so seinen Token, mit welchem bestimmt wird, welche Berechtigungen man hat.
Die Homepage, welche nur für eingeloggte User besteht, ist ähnlich aufgebaut wie Instagram. Es gibt einen Feed in der Mitte, in welchem alle Posts angezeigt werden, auf der linken Seite kann man einen Post hinzufügen und auf der rechten Seite kann man Profildetails des eingeloggten Benutzers sehen.
Die Hauptfunktion unserer Webanwendung ist es, Posts mit einem Bild hochzuladen und zu liken. Ausserdem sollte man sehen, wer alles einen Post geliked hat.

Unser spezifischer Gruppenauftrag ist es, eine Image Gallery zu machen. Wir haben nicht extra dafür eine Seite gemacht, sondern haben diese Gallery in die Mitte der Homepage eingefügt, damit sie mehr im Mittelpunkt unseres Projekts steht.
(Abb. 4: Frontend Struktur)

Use Case Diagramm

Auf diesem Diagramm ist nochmals ersichtlich, welche Funktionen normale User machen können und welche der Admin dieser Webanwendung. Jedoch sind nicht alle möglichen Funktionen abgebildet.

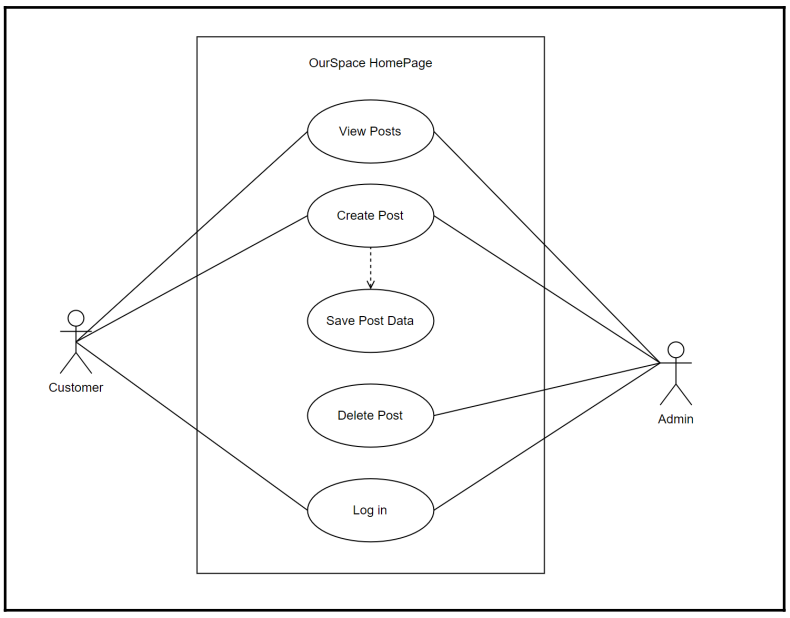


Abb. 5: Use Case Diagramm[6]

Zugriffs-Limitierungen Frontend

Für Benutzer von OurSpace unterscheidet sich das UI grundlegend anhand folgender Parameter:

- Rollen & Autorisierungen
- Beziehung zu einer bestimmten Ressource
    - *Benutzer sieht seine eigenen Profildaten allerdings nicht die anderer*

Wir haben dir Routes mittels Private Routes geschützt, damit nicht unberechtigte Benutzer auf die geschützten Seiten zugreifen können.

Seite	**Rolle**
Homepage öffentlich (/)	Alle
Login (/login)	Alle
Homepage privat (/home)	ADMIN, USER
User (/users/all)	ADMIN
User (/users/{id})	ADMIN, USER

Verbindung der Subkomponenten

Damit alle Komponenten (Frontend, Backend und Datenbank) reibungslos miteinander interagieren können, muss natürlich auch eine gute Verbindung gewährleistet sein. Dies haben wir auf verschiedene Wege gewährleistet.

Das Frontend haben wir mittels sogenannten Services mit dem Backend verbunden. Services stellen in unserem Fall die Funktionalität bereit, um Requests an das Backend zu senden, welche beispielsweise dazu dienen, einen User einzuloggen oder Ähnliches. Um diese Requests auszuführen, verwenden wir die Axios-Library, welche einen Grossteil der HTTP-Funktionalität bereits bereitstellt.

Um die Daten schlussendlich auch in einer Datenbank zu speichern, haben wir das Backend schlussendlich noch via JPA mit einer PostgreSQL-Datenbank verbunden, welche auf einem Docker Container läuft.

Testing Vorgehen

Um sicherzustellen, dass sich das Projekt auch wie geplant verhält, setzen wir auf eine ausgiebige Teststrategie mithilfe von Postman und Cypress.

Testing Ressourcen

- Cypress
    - Ein Tool, mit dem man gut End-to-End Tests durchführen kann, mit einer ähnlichen Syntax, wie man sich von Unit-Tests gewöhnt ist. Diese sogenannten End-to-End Tests dienen dazu, eine Applikation vom einen “Ende” (Frontend) bis zum anderen “Ende” (Datenbank), indem man einen Prozess testet, der all diese Systeme beinhaltet.
    - Diese Tests sind bei uns im Frontend-Repository vorzufinden.
- Postman
    - Mit Postman kann man hauptsächlich die Funktionalität von verschiedenen Requests testen. Beispielsweise kann man testen, ob ein Request einen gezielten HTTP-Statuscode oder bestimmte Daten zurückgibt.
    - Diese Tests sind bei uns im Backend-Repository vorzufinden.

Test Plan

Das Backend wird grösstenteils auf die Funktion der Endpoints geprüft. Hierbei sollten einzelne Requests, als auch eine Abfolge von Requests funktionieren und sind zu finden unter folgendem Pfad: `M223_Backend-YaSei-StLau-SaLus\tests\postman`.

Use Case

Eine Funktion, die wir ausgiebig testen wollen, ist das Erstellen eines Image-Posts, da dies eine zentrale Funktion unseres Projektes darstellt. Um dies zu tun, haben wir dafür einen Use Case definiert:

Actor(s)	Benutzer
Beschreibung	Ein Benutzer will einen Post hochladen, mit einem Bild und einer Beschreibung.
Voraussetzungen	● Benutzer ist eingeloggt ● Benutzer befindet sich auf der Homepage ● Benutzer hat die beiden Input Felder ausgefüllt
Nachbedingungen	● Benutzer hat auf den “Post Hochladen” Knopf gedrückt

Normaler Verlauf	● Benutzer füllt auf der Homepage die beiden Input Felder aus ● Benutzer drückt auf “Post Hochladen” ● Benutzer sieht seinen Post im Feed
Alternativer Verlauf	● Falls die Beschreibung oder URL mehr als 200 Buchstaben hat, kommt eine Fehlermeldung, dass man eine zu lange Beschreibung/URL geschrieben hat. ● Kein Post wird erstellt.
Exceptions	● MethodArgumentNotValidException: Wenn eingegebene Beschreibung oder URL von Bild zu lang ist.

Nebst diesem wollten wir einen Test mit Fokus auf die Zugriffsberechtigungen vorhanden haben.

Actor(s)	Benutzer
Beschreibung	Ein Benutzer will auf die User-Overview-Page zugreifen.
Voraussetzungen	● Benutzer ist eingeloggt ● Benutzer befindet sich auf der Homepage
Nachbedingungen	● *N/A*
Normaler Verlauf	● Benutzer navigiert auf /users/all ● Benutzer sieht alle User aufgelistet
Alternativer Verlauf	● Benutzer navigiert auf /users/all ● Benutzer hat keine Admin-Berechtigungen und wird auf die Seite /unauthorized weitergeleitet
Exceptions	● *N/A*

Anhang

[0] OurSpace Logo (Titelseite)
[1] Domain Model, selbst erstelltes Diagramm
[2] Entity Relationship Diagram, selbst erstellt
[3] Sequenzdiagramm, selbst erstelltes Diagramm
[4] Zusätzliche Logging-Konfigurationen
[5] Frontend Struktur (High-level), selbst erstelltes Diagramm
[6] Use Case Diagramm