

17.09 - 19.09.2025

Mert Bal, Leon Wulff, Azar Wirth

MULTI-USER APPLICATION Noser Young

O U R S P A C E

ÜK 223

Inhaltsverzeichnis

<u>Einleitung</u>	s. 3
<u>To-Do Planung</u>	s. 4
<u>Use Case Definitionen</u>	s. 5 - 9
• <u>UC1 User erstellt neuen Blogpost</u>	5
• <u>UC2 User bearbeitet oder löscht eigenen Blogpost</u>	6
• <u>UC3 Admin bearbeitet oder löscht beliebigen Blogpost</u>	7
• <u>UC4 Nutzer lesen Blogposts mit Pagination und Sortierung</u>	8
• <u>UC5 Zugriffskontrolle bei Änderungen</u>	9
<u>Domain Model</u>	s. 10
<u>ERD (Entity Relationship Model)</u>	s. 10
<u>Sequence Diagram</u>	s.11
<u>Use Case Diagram</u>	s.11
<u>Mockup</u>	s. 12 - 13
<u>Testing</u>	s. 14
<u>Zusammenfassung</u>	s. 15



Einleitung

Dies ist die Dokumentation für das üK-Projekt 223

Unsere Anforderungen waren im Allgemeinen diese:

Frontend:

- Bestehende Rollen und Autoritäten nutzen und erweitern.
- Nur Admins können andere Benutzer erstellen, bearbeiten und löschen.
- Eine Admin Seite erstellen welche nur für Admins zugänglich ist
- Gruppenspezifische Funktionalität im Frontend ermöglichen
- Eine Homepage für eingeloggte User

Security:

- Gruppenspezifische Endpoints sollten nur mit sinnvollen Berechtigungen zugänglich sein.
- Es gibt Bereiche des Frontends, die nur für eingeloggte Benutzer zugänglich sind.
- Es gibt Bereiche des Frontends, die nur für Admins zugänglich sind.

Use Cases (gruppenspezifisch):

- **UC1:** User erstellt neuen Blogpost (Text, Titel, Kategorie, Autor).
- **UC2:** User bearbeitet oder löscht eigenen Blogpost.
- **UC3:** Administrator bearbeitet oder löscht beliebigen Blogpost.
- **UC4:** Alle Nutzer (auch anonyme/uneingeloggte Gäste) lesen Blogposts mit Pagination und Sortierung.
- **UC5:** System stellt sicher, dass nur der Autor eines Blogposts oder ein Admin Änderungen durchführen darf.

In dieser Dokumentation finden sie:

- To-Do Planung
- Use Case Definitionen
- Domain Model
- ERD
- Use Case Diagram
- Mockup

README's können in den entsprechenden Repositories gefunden werden.



T0-D0 Planung

- Mert
- Leon
- Azar

Backend

- ~~Entities erstellen (Blogpost, Authority (fertig machen))~~
- ~~Beziehungen zwischen Entities erstellen~~
- ~~Service Methoden implementieren für Blogpost CR~~
- ~~Service Methoden implementieren für Blogpost UD~~
- ~~Validierung für Admin Änderungen~~
- ~~Security~~
- ~~Pagination / README~~
- Swagger Doku
- Code Überarbeiten

Frontend

- Security / Roles / Service
- ~~Mockup für die Blogpost page~~
- ~~Update Readme~~
- Admin Ansicht/ allgemein Ansicht
- Blogpage (einzel) / Author posts
- Edit/Create (Form) blog page
- Navbar (navigierung) / Routing
- Homepage (Gast/Eingeloggt)
- Admin Page (User Bearbeitung)
- Register Page (Logik)

Testing

- Swagger Endpoints und Doku
- Cypress
- Postman

Dokumentation

- ~~Domain Model~~
- ~~ERD~~
- ~~Sequence Diagram (ein Usecase)~~
- ~~Use Case Definitionen (zu allen)~~
- ~~Use Case Diagram~~
- ~~README schreiben~~
- ~~Testing Doku (JUnit, Cypress, Postman & Swager)~~
- ~~Einleitung~~
- Zusammenfassung

Use Case Definitionen

UC1

Name	User erstellt neuen Blogpost
Rolle(n)	User
Beschreibung	Ein angemeldeter Nutzer möchte einen neuen Blogpost erstellen, der danach veröffentlicht wird.
Vorbedingungen	<ul style="list-style-type: none">• User ist im System angemeldet.• User verfügt über Schreibrechte.
Nachbedingungen	<ul style="list-style-type: none">• Der neue Blogpost ist in der Blogübersicht sichtbar.
Ablauf	<ol style="list-style-type: none">1. User klickt auf "Create Blogpost".2. System zeigt das Eingabeformular.3. User gibt Titel, Inhalt und Kategorie ein.4. User klickt auf "publish".5. System speichert den Blogpost in der Datenbank.6. System bestätigt die erfolgreiche Erstellung.
Ausnahmen	<ul style="list-style-type: none">• User gibt keine gültigen Daten ein → System zeigt Fehlermeldung.• Serverfehler beim Speichern → System zeigt Fehler und bietet erneut Speicherung an.
Priorität	Hoch



UC2

Name	User bearbeitet oder löscht eigenen Blogpost
Rolle(n)	User
Beschreibung	Ein Nutzer kann seine eigenen Blogposts bearbeiten oder löschen.
Vorbedingungen	<ul style="list-style-type: none"> • User ist im System angemeldet. • User verfügt über Bearbeitungs- und Löschrechte.
Nachbedingungen	<ul style="list-style-type: none"> • Post ist aktualisiert oder gelöscht • Änderungen sind in der Datenbank gespeichert
Ablauf	<ol style="list-style-type: none"> 1. User öffnet einen eigenen Post 2. System zeigt Optionen „Edit“ und „Delete“ 3. <ol style="list-style-type: none"> a. User klickt „Edit“, nimmt Änderungen vor, klickt „Save“ b. Oder User klickt „Delete“ und bestätigt die Löschung 4. System aktualisiert oder entfernt den Post
Ausnahmen	<ul style="list-style-type: none"> • User ist nicht der Autor oder Admin → keine Bearbeitung/Löschung möglich • Serverfehler beim Speichern/Löschen → System zeigt Fehler und bietet erneut Speicherung an.
Priorität	Mittel



UC3

Name	Administrator bearbeitet oder löscht beliebigen Blogpost
Rolle(n)	Administrator
Beschreibung	Der Admin kann jeden Blogpost bearbeiten oder löschen, unabhängig vom Autor.
Vorbedingungen	<ul style="list-style-type: none">• Admin ist eingeloggt• Admin hat volle Zugriffsrechte
Nachbedingungen	<ul style="list-style-type: none">• Änderungen werden im ganzen System übernommen
Ablauf	<ol style="list-style-type: none">1. Admin wählt einen beliebigen Post2. System zeigt Optionen zur Bearbeitung und Löschung3. Admin wählt Aktion und bestätigt4. System aktualisiert oder entfernt den Post
Ausnahmen	<ul style="list-style-type: none">• Fehler beim Bearbeiten/Löschen → Fehlermeldung• Datenbankfehler → Retry-Mechanismus
Priorität	Mittel



UC4

Name	Nutzer lesen Blogposts mit Pagination und Sortierung
Rolle(n)	Administrator, User, Besucher
Beschreibung	Alle Nutzer können vorhandene Blogposts lesen mit Pagination und Sortierung.
Vorbedingungen	<ul style="list-style-type: none">• Blogposts sind vorhanden
Nachbedingungen	<ul style="list-style-type: none">• Nutzer können Posts durchsuchen, filtern und lesen
Ablauf	<ol style="list-style-type: none">1. Nutzer navigiert zur Blogübersicht2. System zeigt Liste mit Standard-Sortierung (z. B. nach Datum, Name, etc.)3. Nutzer kann Sortierkriterien auswählen4. Nutzer verwendet Pagination, um weitere Seiten ansehen zu können
Ausnahmen	<ul style="list-style-type: none">• Keine Blogposts vorhanden → Hinweis „Keine Inhalte vorhanden“• Serverfehler → Fehlermeldung anzeigen
Priorität	Hoch

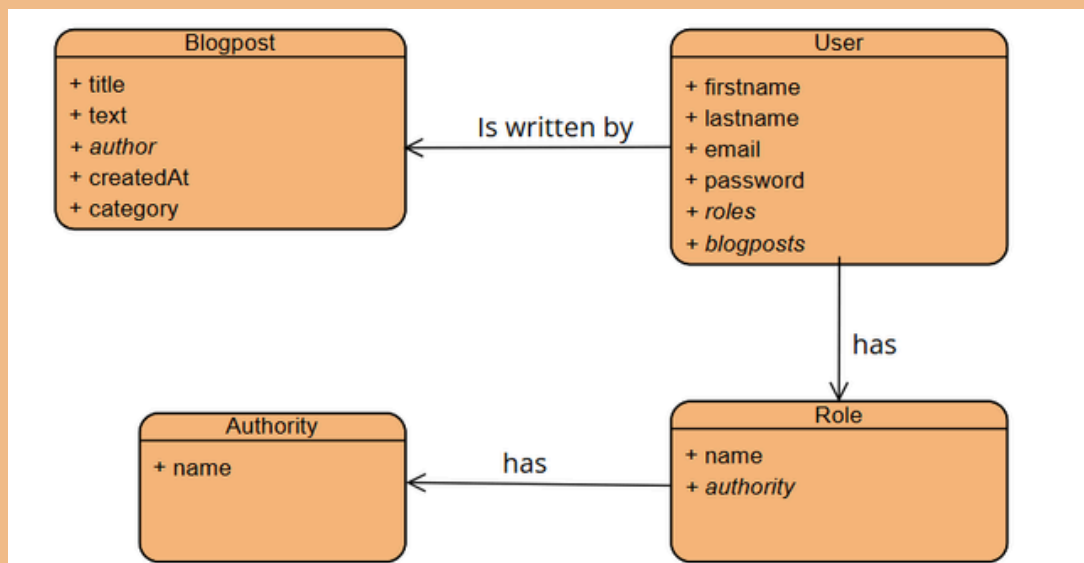


UC5

Name	Zugriffskontrolle bei Änderungen
Rolle(n)	User (Autor), Administrator
Beschreibung	Das System verhindert, dass nicht berechnigte Nutzer Blogposts bearbeiten oder löschen.
Vorbedingungen	<ul style="list-style-type: none">• Nutzer ist eingeloggt• Aktion zum Ändern eines Blogposts wird ausgelöst
Nachbedingungen	<ul style="list-style-type: none">• Unautorisierte Änderungen werden verhindert
Ablauf	<ol style="list-style-type: none">1. Nutzer beginnt Bearbeitungs- oder Löschvorgang2. System prüft, ob Nutzer berechnigt ist (Autor oder Admin)3.<ol style="list-style-type: none">a. Falls ja: Zugriff wird gewährtb. Falls nein: Zugriff wird verweigert, Fehlermeldung wird angezeigt
Ausnahmen	<ul style="list-style-type: none">• Session abgelaufen → Weiterleitung zum Login• Fehlerhafte Rechteverwaltung → Sicherheitsvorfall, Logging
Priorität	Sehr hoch

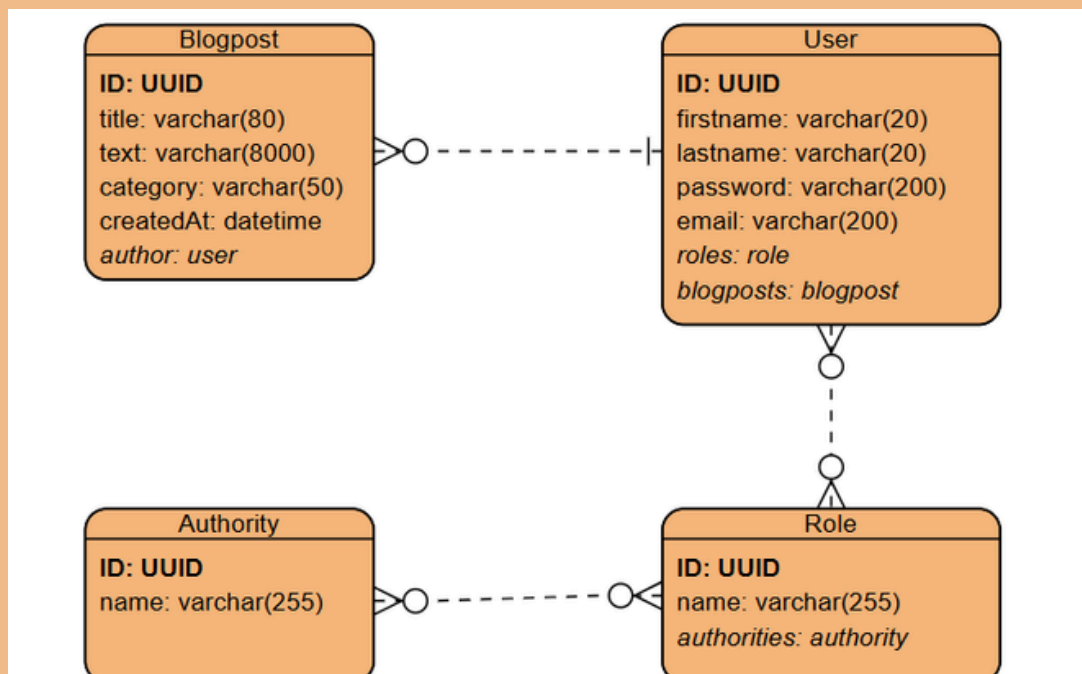


Domain model



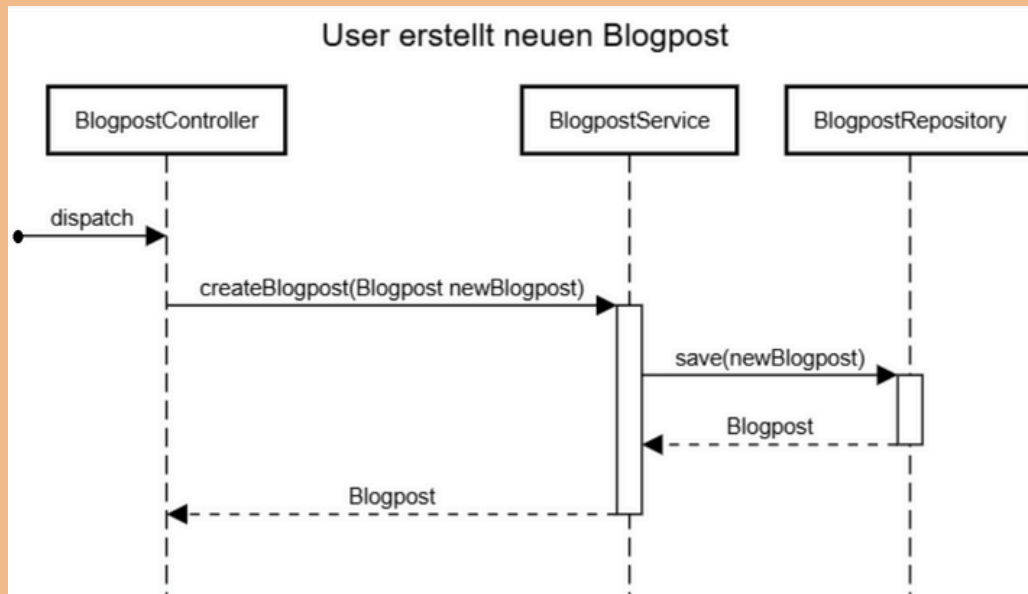
Hier werden mit wenigen Details definiert, wie die unterschiedlichen Entitäten miteinander kommunizieren. Ebenso wird gezeigt welche Attribute die jeweiligen Entitäten haben und mit welchen Entitäten sie verbunden sind per foreign key

ERD (Entity Relationship Diagram)



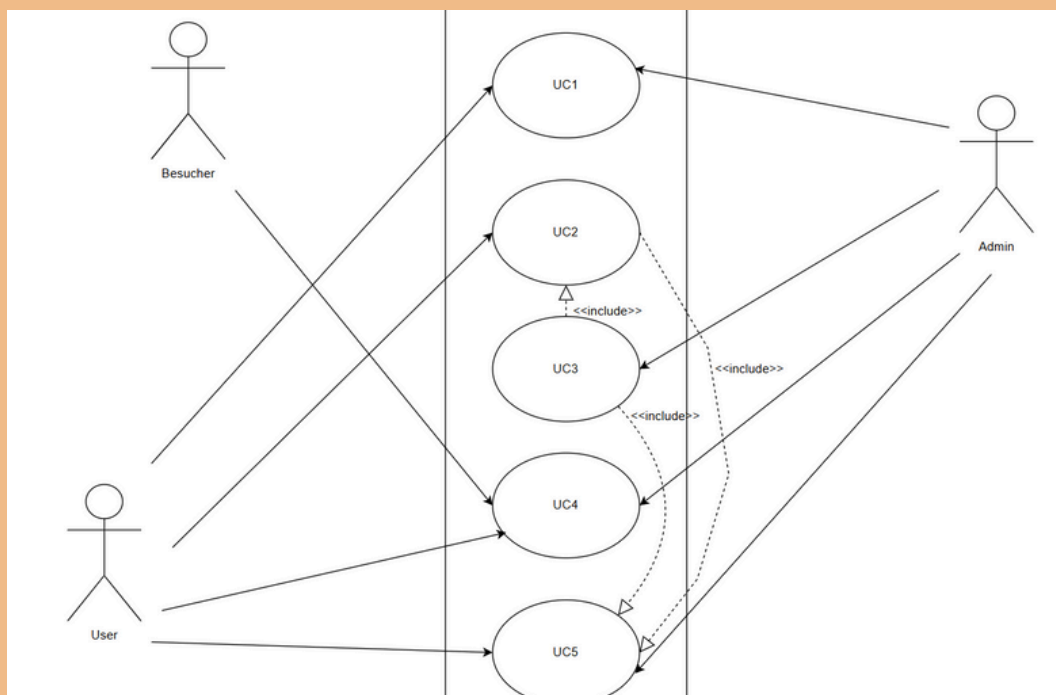
Die in fett geschriebenen Variablen (z.B. **ID: UUID**) sind primary keys und die kursiv geschriebenen Variablen sind foreign keys (z.B. *blogposts: blogpost*). Hierbei werden die Beziehungen zwischen den Entitäten dargestellt, welche Attribute mit welchen Datentypen verwendet werden. Ebenso wird gezeigt welche Primary und foreign keys es gibt

Sequence Diagram



Der User löst über den BlogpostController die Methode createBlogpost() aus, welche den BlogpostService aufruft. Dieser speichert den neuen Blogpost über das BlogpostRepository in der Datenbank und gibt das erstellte Objekt zurück.

Use Case Diagram



Im Use Case Diagram können User Blogposts erstellen (UC1) und eigene bearbeiten/löschen (UC2), Admins auch fremde (UC3). Alle, auch Gäste, dürfen lesen (UC4). UC5 stellt sicher, dass nur Autor oder Admin Änderungen machen dürfen – daher wird UC5 in UC2 und UC3 per include eingebunden.

Mockup



Nur zugänglich als eingeloggter Admin

Nutzen:

- alle Blogs editieren
- alle Blogs löschen können

Elemente:

- Navbar
- BlogCards
- BlogsGrid
- CreateBtn



Zugänglich für alle (Gast, Nutzer und Admin)


Nutzen:

- alle Blogs ansehen
- Blogs erstellen (falls eingeloggt)

Elemente:

- Navbar
- BlogCards
- BlogsGrid
- CreateBtn

Access: Author/Admin

BLOGPOSTS Home Blogs 

Title


select genre

Text

1

Cancel Save

Access: Author

BLOGPOSTS Home Blogs 

Username

TITLE

FULL TEXT

hh.dd.mm.yy

2

Delete Edit

1.Create/Update form

Elemente:

- Navbar
- Blogform
- SaveBtn
- CancelBtn

2.Single Blog view

Elemente:

- Navbar
- BlogCard
- EditBtn
- CancelBtn

Access: All

BLOGPOSTS Home Blogs 

Username

TITLE

FULL TEXT

hh.dd.mm.yy

3

3.Single View blogpost

Elemente:

- Navbar
- Blogform

Testing

Postman

Mit Postman werden alle verfügbaren Endpoints (UC 1-5) unseres Blogposts getestet (Siehe Backend Readme), inklusive vereinzelter User Endpoints, zum Testen und Erhalten eines Bearer Tokens.

Wir haben uns für Postman entschieden, da wir direkte Endpoints, mit ständig ändernden Variablen testen können, sowie unterschiedliche Edgecases und erfolgreiche fail tests.

Cypress

Mit Cypress testen wir den UC 1 & 2, um einen visuellen Ablauf und End-To-End Test haben, wie es wäre, wenn ein User einen Blogpost erstellt, ihn bearbeitet und dann löscht. Dazu werden wir noch diverse Edgecases und Fail-Tests überprüfen.

Wir haben uns dafür entschieden, damit wir nicht nur auf die Unit-Tests angewiesen sind und auch visuelle Tests haben, die den ganzen Bereich von Backend, Datenbank und Frontend abdeckt. Dadurch können wir überprüfen, ob etwas nicht im Backend korrekt läuft oder Funktionalitäten wie das Klicken von Buttons nicht funktioniert.

Swagger

Mit Swagger testen wir die UC 1-5, sozusagen alle Endpoints, um wie bei JUnit direkten Zugang zu den Endpoints zu haben.

Wir benutzen Swagger, da wir direkt Zugang zu den Endpoints haben und Erklärungen hinzufügen, wie genau die Endpoints aufgebaut sind und funktionieren. Zudem können direkt die Variablen eingesetzt und getestet werden und bieten direkt Zugang auf die Response.



Zusammenfassung

Im Folgenden gehe ich auf diverse Punkte des Projektes ein. Dabei werde ich erläutern, wie wir im Team zusammen gearbeitet haben, wie wir uns Projekt aufgebaut und strukturiert haben.

Ebenso die Verwendung von Testing-Strategien und die allgemeine Lösung.

Wir haben die Aufgaben so aufgeteilt, das jeder Bereich fair abgedeckt ist. Hierbei haben wir auch darauf geachtet, das auch Stärken sowie Schwächen eingebunden werden, um etwas neues zu lernen, aber auch seine Stärken zum Test zu setzen.

Im Backend haben wir die Entität Blogpost erstellt, den haben wir die Attribute author, createdAt, titel, text und category definiert.

Basierend auf dem haben wir unsere Endpoints aufgebaut und die Logik implementiert. Dafür haben wir Klassen, enums und Interfaces verwendet. Zum testen der Funktionen, haben wir uns entschieden alle Testing-Funktionen zu verwenden, da wir fanden, dass die eine gute Entscheidung sei, um sicher zu gehen, dass es auf jede Art und Weise klappt. Dafür haben wir Postman, JUnit, Cypress und Swagger verwendet.

Im Frontend haben wir uns an das Atomic Design gehalten, um unser Projekt übersichtlich zu strukturieren. Entsprechend haben wir die Service Methoden hinzugefügt und auch ein Mockup erstellt, um eine Idee der Visualisierung für unser Frontend zu haben. Für ein besseres miteinander, haben wir für die Versionsverwaltung mit git gearbeitet und branches erstellt.

Für unsere Dokumentation haben wir unsere Use-Cases mit Use-Case Beschreibungen, Use-Case Diagramm, Sequenzdiagramm, einem ERD und Domänenmodell versehen. Ebenso haben wir hier die Testing-Strategien erläutert und unsere Planung auf Aufteilung hineingeschrieben, um weiterhin eine Übersicht zu behalten wer was macht.

