# Social Media Backend Projekt

Modul 295

Alen Arandjelovic

30.01.2026

# Inhaltsverzeichnis

# I - Informieren

## Projekt-Idee-Pitch

Dieses Projekt ist ein Backend für eine einfache Social-Media-Plattform.
Benutzer können erstellt werden, Beiträge verfassen und kommentieren.
Alle Funktionen werden über eine REST-API bereitgestellt und ohne Frontend über HTTP-Requests getestet.
Die Daten werden validiert, persistent gespeichert und sind vollständig testbar.

## User Stories

### User Story 1: User erstellen

**Als** neuer Benutzer
**möchte ich** einen User erstellen können
**damit** ich Beiträge und Kommentare verfassen kann.

**Akzeptanzkriterien:**

1. Ein User kann über POST /users erstellt werden.
2. Pflichtfelder (z. B. Username, E-Mail) müssen vorhanden sein.
3. Ungültige oder fehlende Felder werden mit einer passenden Fehlermeldung (HTTP 400) abgewiesen.
4. Bei erfolgreicher Erstellung wird der neue User mit einer eindeutigen ID zurückgegeben.
5. Der User wird persistent in der Datenbank gespeichert.

---

### User Story 2: Post erstellen

**Als** registrierter User
**möchte ich** einen Post erstellen können
**damit** ich Inhalte mit anderen teilen kann.

**Akzeptanzkriterien:**

1. Ein Post kann über POST /posts erstellt werden.
2. Ein Post ist immer einem existierenden User zugeordnet.

3. Pflichtfelder (z. B. Textinhalt) müssen vorhanden sein.
4. Existiert der angegebene User nicht, wird der Post abgewiesen (HTTP 404).
5. Der Post wird in der Datenbank gespeichert und kann über GET /posts abgerufen werden.

---

## User Story 3: Kommentar zu einem Post hinzufügen

**Als** User
**möchte ich** einen Kommentar zu einem Post hinzufügen
**damit** ich auf Beiträge reagieren kann.

**Akzeptanzkriterien:**

1. Ein Kommentar kann über POST /posts/{postId}/comments erstellt werden.
2. Der referenzierte Post muss existieren.
3. Der Kommentar enthält einen Text und einen zugehörigen User.
4. Fehlt der Post oder der User, wird eine passende Fehlermeldung zurückgegeben.
5. Der Kommentar ist nach dem Erstellen über einen GET-Endpoint abrufbar.

# P- Planen

## Klassenplan

Dies ist zunächst der Plan, welche Klassen vorgesehen sind und welche Attribute sie enthalten sollen.

**User**

- Id, username, email, createdAt

**Post**

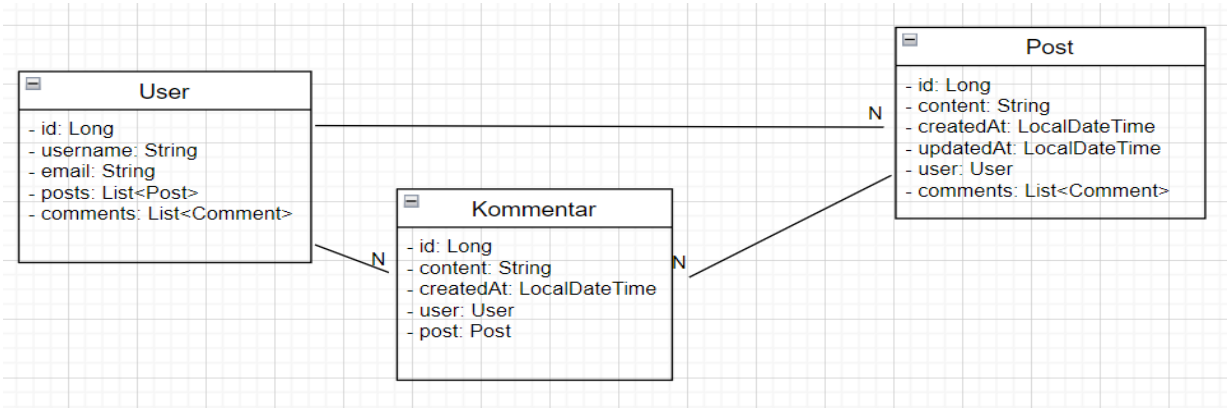
- Id, userId, text, createdAt

**Comment**

- Id, postId, userId, text, createdAt

# E- Entscheiden

## Klassendiagram




# R- Realisieren

## Arbeitsprotkol:

**23.01.2026 – Projektstart**

Heute habe ich die Projektidee und User Stories formuliert und erstellt. Darauf habe ich das Klassendiagramm erstellt. Dann habe ich den Projektordner erstellt. Anschließend habe ich das GitHub-Repository angelegt und lokal auf der VM mit dem Projektordner verbunden. mit Sven's Schirmer Hilfe die Spring-Boot zip Datei erstellt und anschliessend auf der VM entpackt und so schon mal die Basis Projektstruktur vorbereitet.

**23.01.2026 – Projektstart**

Heute hab ich den Code Finalisiert und alle Dokumentationsteile Fertig geschrieben wie den Testplan, Testprotokol, Rest-Api Schnittstellen usw. Dazu Habe ich die Dokumentationsdatei verschönert und übersichtlicher gemacht mit einem Titelblatt und einem Inhaltsverzeichniss.

# K- Kontrollieren

## Projekt-Testplan

Folgende Http methode (wie GET,POST,DELETE etc) in den Testschritte werden mithilfe von Insomnia ausgeführt.

**Testfall 1: User erstellen**

| Test-ID | Testfall | Schritte / Vorgehensschritte | Erwartetes Ergebnis |
|---|---|---|---|
| TC1.1 | User erstellen – gültige Daten | 1. POST /users Body: { "username": "max", "email": "max@example.com" }<br>2. Prüfen, dass HTTP 201 zurückkommt<br>3. GET /users/{id} aufrufen<br>4. Prüfen, dass { "id": 1, "username": "max", "email": "max@example.com" } zurückkommt | User wird erstellt, ID zurückgegeben, persistent – HTTP 201 / 200 |
| TC1.2 | User erstellen – fehlende Pflichtfelder | 1. POST /users Body: { "username": "max" }<br>2. Prüfen, dass HTTP 400 zurückkommt<br>3. Prüfen, dass Fehlermeldung "E-Mail ist erforderlich" erscheint | Fehlermeldung, User wird nicht erstellt – HTTP 400 |
| TC1.3 | User erstellen – ungültige E-Mail | 1. POST /users Body: { "username": "max", "email": "maxexample.com" }<br>2. Prüfen, dass HTTP 400 zurückkommt<br>3. Prüfen, dass Fehlermeldung "E-Mail ungültig" erscheint | Fehlermeldung – HTTP 400 |

**Testfall 2: Post erstellen**

| Test-ID | Testfall | Schritte / Vorgehensschritte | Erwartetes Ergebnis |
|---|---|---|---|
| TC2.1 | Post erstellen – gültiger User | 1. POST /posts Body: { "title": "Mein erster Post", "content": "Inhalt des Posts", "userId": 1 }<br>2. Prüfen, dass HTTP 201 zurückkommt<br>3. GET /posts/{id} aufrufen<br>4. Prüfen, dass Post mit UserID=1 zurückkommt | Post erstellt, User korrekt zugeordnet – HTTP 201 / 200 |
| TC2.2 | Post erstellen – fehlende Pflichtfelder | 1. POST /posts Body: { "userId": 1 }<br>2. Prüfen, dass HTTP 400 zurückkommt<br>3. Prüfen, dass Fehlermeldung "Titel und Content erforderlich" erscheint | Fehlermeldung – HTTP 400 |
| TC2.3 | Post erstellen – nicht existierender User | 1. POST /posts Body: { "title": "Test", "content": "Text", "userId": 999 }<br>2. Prüfen, dass HTTP 404 zurückkommt<br>3. Prüfen, dass Fehlermeldung "User nicht gefunden" erscheint | Fehlermeldung – HTTP 404 |
| TC2.4 | Posts abrufen | 1. GET /posts<br>2. Prüfen, dass HTTP 200 zurückkommt<br>3. Prüfen, dass Post "Mein erster Post" enthalten ist | Liste enthält Post – HTTP 200 |

**Testfall 3: Kommentar erstellen**

| Test-ID | Testfall | Schritte / Vorgehensschritte | Erwartetes Ergebnis |
|---|---|---|---|
| TC3.1 | Kommentar erstellen – existierender Post & User | 1. POST /posts/1/comments Body: { "content": "Nice Post!", "userId": 1 }<br>2. Prüfen, dass HTTP 201 zurückkommt<br>3. GET /posts/1/comments aufrufen<br>4. Prüfen, dass Kommentar vorhanden ist | Kommentar erstellt, abrufbar – HTTP 201 / 200 |
| TC3.2 | Kommentar erstellen – Post existiert nicht | 1. POST /posts/999/comments Body: { "content": "Test", "userId": 1 }<br>2. Prüfen, dass HTTP 404 zurückkommt<br>3. Prüfen, dass Fehlermeldung "Post nicht gefunden" erscheint | Fehlermeldung – HTTP 404 |
| TC3.3 | Kommentar erstellen – User existiert nicht | 1. POST /posts/1/comments Body: { "content": "Test", "userId": 999 }<br>2. Prüfen, dass HTTP 404 zurückkommt<br>3. Prüfen, dass Fehlermeldung "User nicht gefunden" erscheint | Fehlermeldung – HTTP 404 |
| TC3.4 | Kommentare abrufen | 1. GET /posts/1/comments<br>2. Prüfen, dass HTTP 200 zurückkommt<br>3. Prüfen, dass Kommentar "Nice Post!" enthalten ist | Liste enthält Kommentar – HTTP 200 |

## Testprotokoll – Chat-Projekt

| Test-ID | Testfall | Tester | Ergebnis |
|---|---|---|---|
| TC1.1 | User erstellen – gültige Daten, GET prüfen | Alen | bestanden |
| TC1.2 | User erstellen – fehlende Pflichtfelder | Alen | bestanden |
| TC1.3 | User erstellen – ungültige E-Mail | Alen | bestanden |
| TC2.1 | Post erstellen – gültiger User, GET prüfen | Alen | bestanden |
| TC2.2 | Post erstellen – fehlende Pflichtfelder | Alen | bestanden |
| TC2.3 | Post erstellen – nicht existierender User | Alen | bestanden |
| TC2.4 | Posts abrufen – prüfen ob Post vorhanden | Alen | bestanden |
| TC3.1 | Kommentar erstellen – existierender Post & User | Alen | bestanden |
| TC3.2 | Kommentar erstellen – Post existiert nicht | Alen | bestanden |
| TC3.3 | Kommentar erstellen – User existiert nicht | Alen | bestanden |
| TC3.4 | Kommentare abrufen – prüfen ob Kommentar vorhanden | Alen | bestanden |

## REST-API Schnittstellen

### Users

| Methode | Pfad | Beschreibung | Request Body | Response Body |
|---|---|---|---|---|
| GET | /users | Alle Benutzer abrufen | - | JSON-Liste aller User |
| GET | /users/{id} | Benutzer nach ID abrufen | - | JSON eines Users |
| POST | /users | Neuen Benutzer erstellen | { "username": "...", "email": "..." } | JSON des erstellten Users |
| PUT | /users/{id} | Benutzer aktualisieren | { "username": "...", "email": "..." } | JSON des aktualisierten Users |
| DELETE | /users/{id} | Benutzer löschen | - | Statusmeldung (z.B. 200 OK) |

---

### Posts

| Methode | Pfad | Beschreibung | Request Body | Response Body |
|---|---|---|---|---|
| GET | /posts | Alle Posts abrufen | - | JSON-Liste aller Posts |
| GET | /posts/{id} | Post nach ID abrufen | - | JSON eines Posts |
| POST | /posts | Neuen Post erstellen | { "content": "...", "userId": ... } | JSON des erstellten Posts |
| PUT | /posts/{id} | Post aktualisieren | { "content": "...", "userId": ... } | JSON des aktualisierten Posts |
| DELETE | /posts/{id} | Post löschen | - | Statusmeldung (z.B. 200 OK) |

---

### Comments

| Methode | Pfad | Beschreibung | Request Body | Response Body |
|---|---|---|---|---|
| **GET** | **/comments** | **Alle Kommentare abrufen** | **-** | **JSON-Liste aller Comments** |
| **GET** | **/comments/{id}** | **Kommentar nach ID abrufen** | **-** | **JSON eines Comments** |
| **POST** | **/comments** | **Neuen Kommentar erstellen** | **{ "content": "...", "userId": ..., "postId": ... }** | **JSON des erstellten Comments** |
| **PUT** | **/comments/{id}** | **Kommentar aktualisieren** | **{ "content": "...", "userId": ..., "postId": ... }** | **JSON des aktualisierten Comments** |
| **DELETE** | **/comments/{id}** | **Kommentar löschen** | **-** | **Statusmeldung (z.B. 200 OK)** |

# A- Auswerten

## Hilfestellung

Für einzelne technische Fragen und Formulierungen habe ich **ChatGPT** genutzt. Zusätzlich erhielt ich Unterstützung von **Sven Schirmer**, der mir bei spezifischen Problemen weitergeholfen hat.

## Auswertung

Am meisten Mühe hatten mir die **Testplanung** und die **Tests in Insomnia**, da diese deutlich mehr Zeit brauchte als erwartet. Beim Testplan hatte ich mühe mit dem Formatieren und das definieren von klaren Testschritten. Die Dokumentation ging am besten.