UEK 223

Thema: Documentation

Dokumentinformationen

Dateiname: Documentation-uek-223.docx

Speicherdatum: 05.11.2021

Autoreninformationen

Autor: Dominic Ritzmann

E-Mail: ritzmann@noseryoung.com

Inhaltsverzeichnis

[Abbildungsverzeichnis 2](#_Toc88039759)

[Tabellenverzeichnis 2](#_Toc88039760)

[Änderungsgeschichte 2](#_Toc88039761)

[1 Einleitung 3](#_Toc88039762)

[1.1 Sinn und Zweck 3](#_Toc88039763)

[1.2 Referenzdokumente 3](#_Toc88039764)

[1.3 Abkürzungen 3](#_Toc88039765)

[2 Hauptauftrag 4](#_Toc88039766)

[2.1 Funktionale Anforderungen 4](#_Toc88039767)

[2.2 Nicht funktionale Anforderungen 4](#_Toc88039768)

[3 Arbeitsaufteilung 5](#_Toc88039769)

[4 Funktionalität des Programmes 5](#_Toc88039770)

[5 Endpoints 6](#_Toc88039771)

[6 Use Cases 7](#_Toc88039772)

[7 Diagramme 8](#_Toc88039773)

[7.1 Klassen Diagramm 8](#_Toc88039774)

[7.2 Domain Model 9](#_Toc88039775)

[8 Sequence Diagrams 10](#_Toc88039776)

[8.1 BlogPostController Class 10](#_Toc88039777)

Abbildungsverzeichnis

**Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.**

Tabellenverzeichnis

[Tabelle 1 Versionen 2](#_Toc449701436)

[Tabelle 2 Abkürzungen 3](#_Toc449701437)

Änderungsgeschichte

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Datum** | **Autor** | **Details** |
| 1.0 | 05.11.2021 | Ritzmann | Dokument erstellt |
| 1.1 | 05.11.2021 | Ritzmann | Hauptauftrag, Funktionale und nicht funktionale Anforderungen. |
| 1.2 | 16.11.2021 | Ritzmann | Arbeitsaufteilung, Funktionalität des Programms |
| 1.3 | 17.11.2021 | Ritzmann | Endpoints, Use Cases, Diagramms |

Tabelle 1 Versionen

# Einleitung

## Sinn und Zweck

Das vorliegende Dokument beschreibt die Arbeitsaufteilung und den Ablauf des ÜK 223.

## Referenzdokumente

1. …
2. …

## Abkürzungen

|  |  |
| --- | --- |
| **Abkürzung** | **Beschreibung** |
| NYP | **N**oser **Y**oung **P**rofessionals |

Tabelle 2 Abkürzungen

# Hauptauftrag

Unser Auftrag war es, eine Multiuser Applikation zu erstellen, nach bestimmten Voraussetzungen. Wir mussten ein Blog Post Model erstellen, welches Informationen von Blogeinträgen enthält. Jeder Eintrag hat einen Author oder eine Authorin. Nur der oder die Author/in oder ein Administrator kann den Blogpost bearbeiten oder löschen. Wir mussten Endpoints erstellen, um die CRUD Operationen sicherzustellen. Die GET Methode soll Pagination und eine Sorting Funktion haben. Auch Unautorisierte Benutzer sollen Blogs lesen können.

## Funktionale Anforderungen

### User, Rollen & Autoritäten

Um die funktionalen Anforderungen zu erfüllen, müssen wir mehrere User, Rollen und Autoritäten erstellen. Die User, Rollen und Autoritäten müssen in der Datenbank abgespeichert werden können. Ausserdem sollen neue erstellt, gelöscht oder verändert werden. Wie viele Rollen und Autoritäten wir einbauen ist unsere Entscheidung.

### Security

Beim Login Versuch sollen die Daten des Users authentifiziert und autorisiert werden mithilfe von Tests. Jeder REST Endpoint ist nur durch die entsprechende Autorität erreichbar.

## Nicht funktionale Anforderungen

### Implementation

Die Daten werden in einer PostgresSQL Datenbank abgespeichert und das OR Mapping wird mit JPA gemacht.

### Testing

Mindestens ein Endpoint wird ausführlich getestet. Dazu gehört die implementierte Funktionalität wird mit Component Tests in Postman getestet. Die Endpoints werden mit mehreren Usern welche unterschiedlichen Rollen haben getestet. Es wird mindestens ein Erfolgsfall und ein Error Fall gestestet. Dazu werden Use Cases nach UML Standard geschrieben.

### Multiuserfähigkeit

Die ACID (Atomicity, Consistency, Isolation, Durability) Prinzipien warden eingehalten.

### Dokumentation

Die implementierte Funktionalität im Projekt wurde ausführlich und verständlich dokumentiert. Es ist mindestens ein Readme File im Github Repository vorhanden, welches die wichtigsten Informationen beinhaltet sowie eine Setup Anleitung. Alle Endpoints, die implementiert wurden, sind beschrieben. Es wurde ein Domänenmodell erstellt, welches die ganze Applikation beschreibt. Ein Klassen Diagramm ist vorhanden, welches die gesamte implementierte Funktionalität beschreibt. Die implementierten Endpoints wurden mit einem Sequence diagram beschrieben. Für die getesteten Endpoints wurden Use Cases geschrieben.

# Arbeitsaufteilung

Die Arbeit haben wir spontan aufgeteilt, da wir die Issues von Github nutzen, welche uns einen guten Überblick verschaffte, was alles schon erledigt wurde und was nicht. Wir haben den Issues entweder nur eine Person oder beide hinzugefügt. An den meisten Aufgaben haben wir beide gearbeitet.

Ein Bild, das Text, Monitor, Screenshot, Bildschirm enthält.

Automatisch generierte Beschreibung

# Funktionalität des Programmes

Mit unserem Programm kann man mit einem nicht Angemeldetem User, Blogposts nur ansehen. Dazu kann man in der Get Methode einen Filter angeben, um nach einem bestimmten Feld zu sortieren. Dies kann man über die URL /Feld. Über die URL /pagination/offset/pageSize kann man Pagination anwenden. Mit dem /offset gibt man an, welche Seite man sehen möchte. Mit /pageSize gibt man die grösse der Seite an, also wie viele Einträge auf einer Seite angezeigt werden. Angemeldete User können Blogposts ansehen und erstellen. Dieser wird dann in der Datenbank abgespeichert. Der User, der den Blogpost erstellt hat, also Author/in oder wer Admin ist, kann den Blogpost bearbeiten oder löschen. Admins können Blogpost ansehen, erstellen, updaten und löschen.

# Endpoints

In unserem Programm gibt es viele Endpoints. In unserem UserController haben wir die Get Endpoints /api/ und /api/users. Über die /api/ Url sieht man auf der Website nur ein Hello World. Diesen Endpoint nutzten wir nur für Testzwecke. Mit der Url /api/users kann man alle User herauslesen.

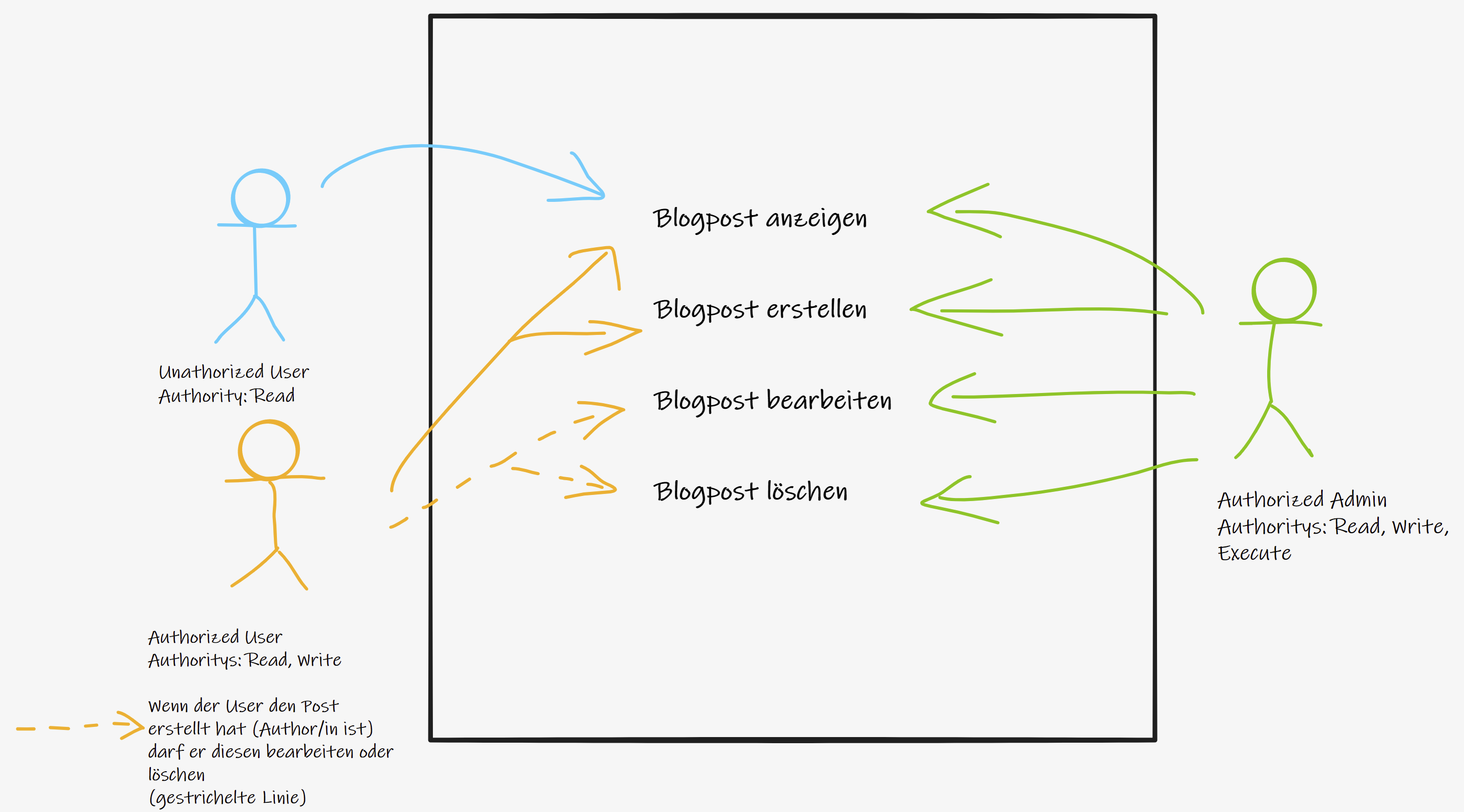
In unserem BlogPostController haben wir 3 Get Endpoints und je 1 Post, Put und Delete Endpoint. Mit dem ersten Endpoint /blogpost bekommt man alle Blogposts angezeigt. Mit dem 2ten Endpoint /blogpost/{field} kann man aufsteigend nach einem bestimmten Feld sortieren. Mit dem letzten Endpoint /blogpost/pagination/{offset}/{pageSize} kann man Pagination anwenden. Offset steht für die Seitenzahl und Pagesize für wie viele Einträge auf einer Seite vorhanden sind. Für diese Endpoints braucht man keine bestimmten Autorisierungen.

Um auf den Post Endpoint zuzugreifen, nutzt man die Url /blogpost/. Um diesen Endpoint nutzen zu können, muss man mindestens angemeldet sein.

Um auf den Update Endpoint zuzugreifen, nutzen wir die Url /blogpost/search/{id}. Ganz hinten geben wir die Id des Posts an. So können wir einen Post updaten. Um diesen Endpoint nutzen zu können, muss man Author/in des Posts oder Admin sein.

Der letzte Endpoint ist der delete Endpoint. Die Url für diesen Endpoint lautet /blogpost/delete/{id}. Hinten geben wir wieder als letztes die Id des Posts, an den wir löschen möchten. Um diesen Endpoint nutzen zu können, muss man Author/in des Posts oder Admin sein.

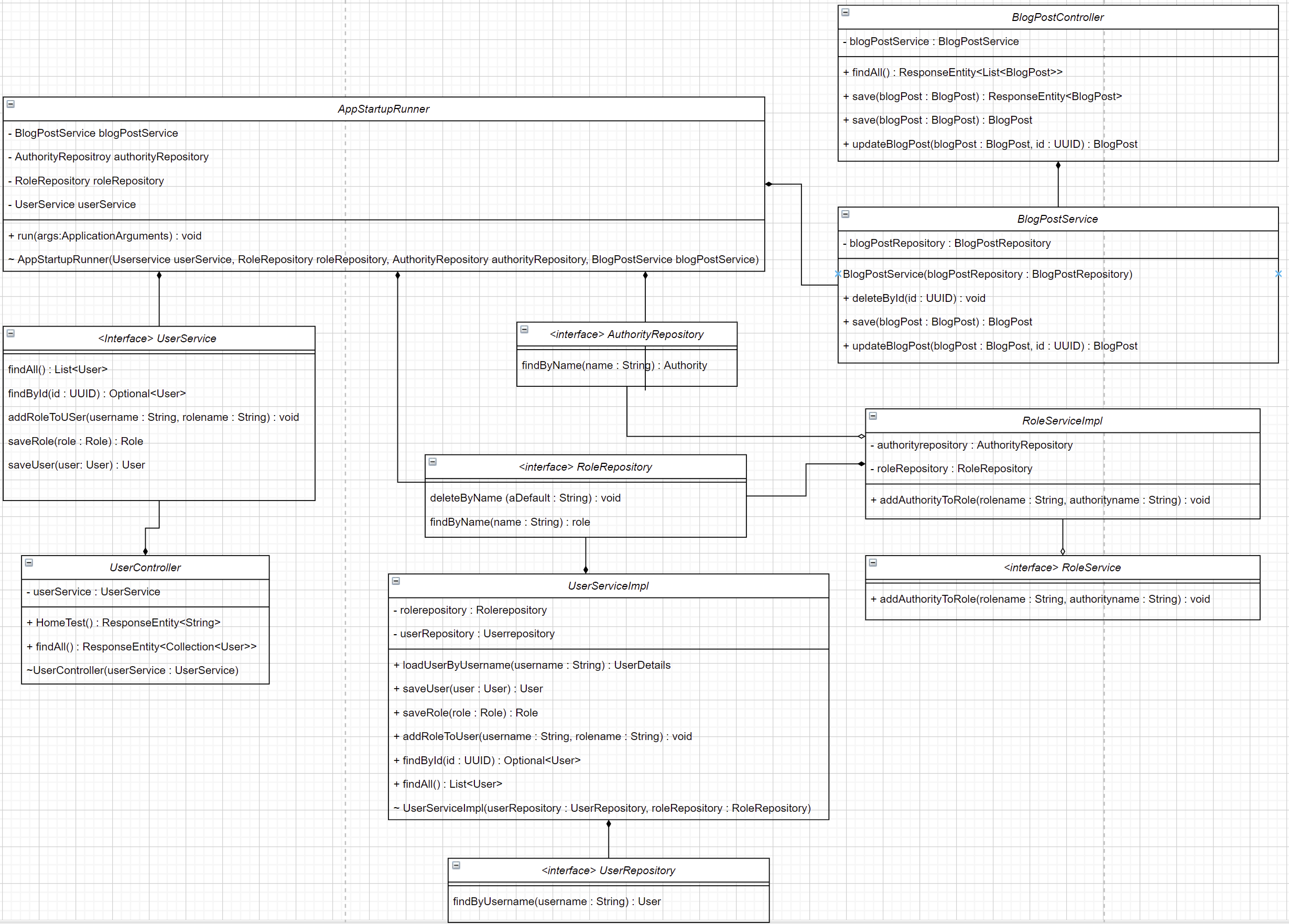
# Use Cases



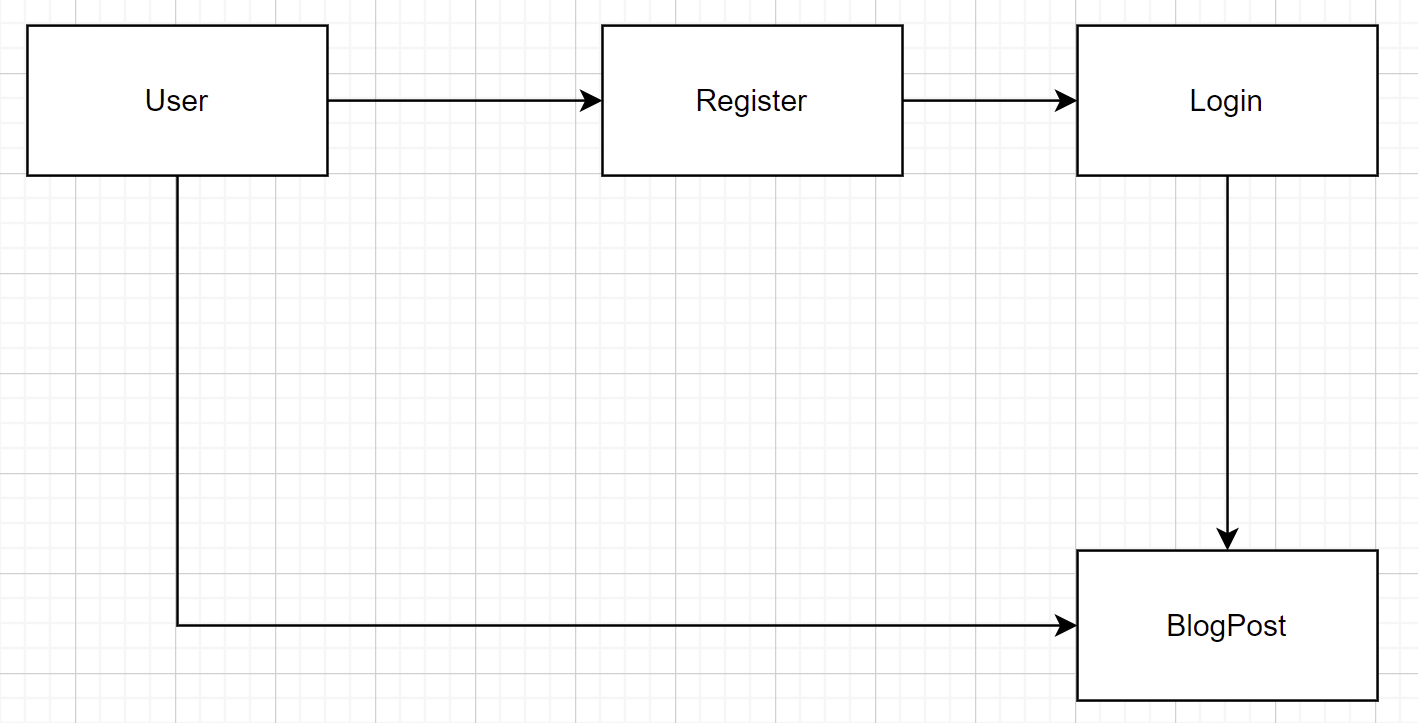
In diesem Diagramm werden nochmals die Rechte der jeweiligen User Visuell dargestellt. Die gestrichelte Linie beim angemeldeten User bedeutet, wenn dieser User der/die Author/in des Blogposts ist.

# Diagramme

## Klassen Diagramm



## Domain Model

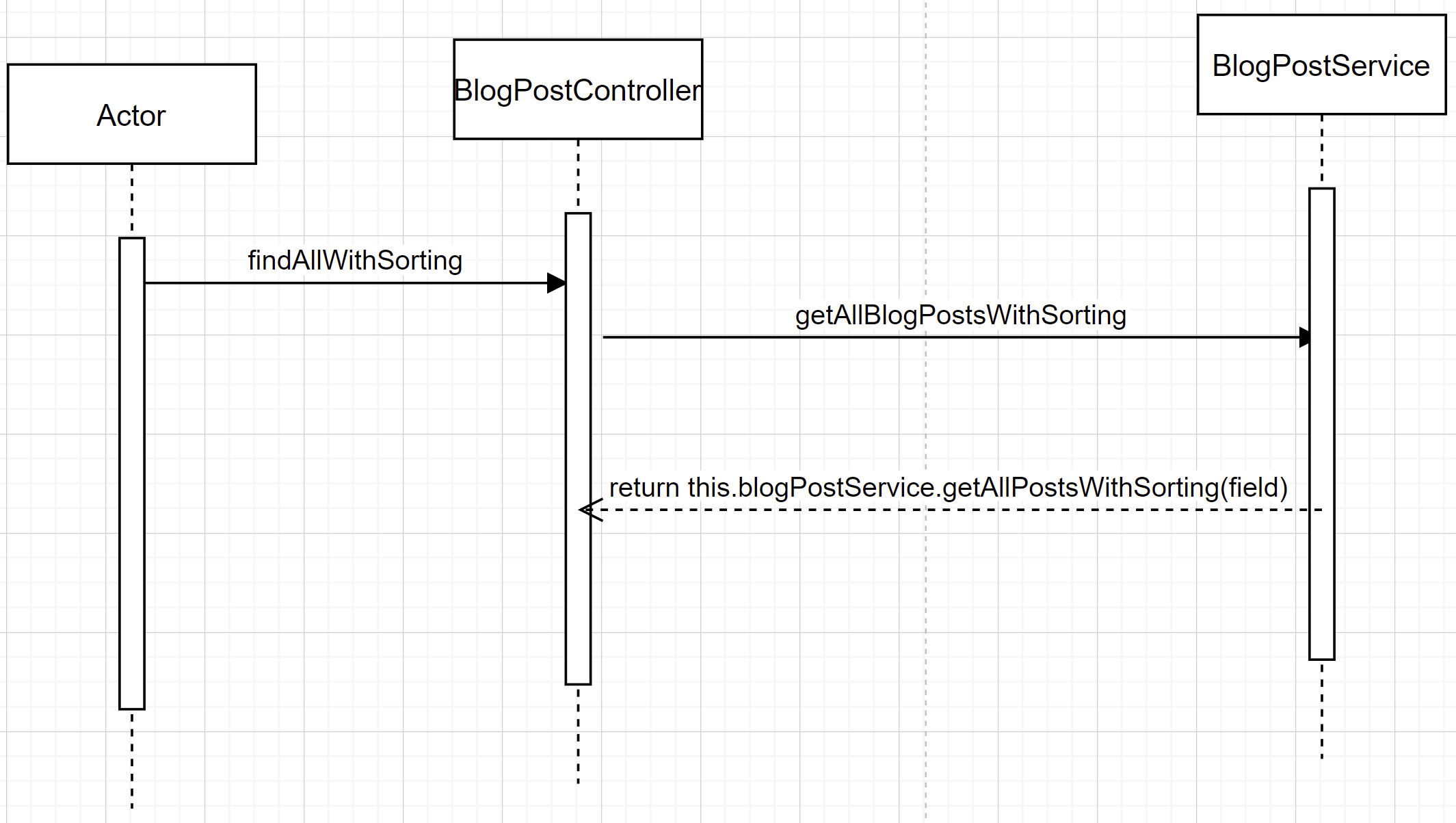


Ein User, der nicht angemeldet ist, kann direkt auf die Blogposts zugreifen. Allerdings kann er sie nur anschauen. Er kann weder einen neuen Post erstellen, einen bestehenden bearbeiten oder löschen. Wenn der User sich registriert und anmeldet, kann er auf die Blogposts zugreifen. Er kann Sie anschauen und einen neuen Post erstellen. Wenn der User der/die Author/in des Blogposts ist, kann der User ihn auch bearbeiten oder löschen.

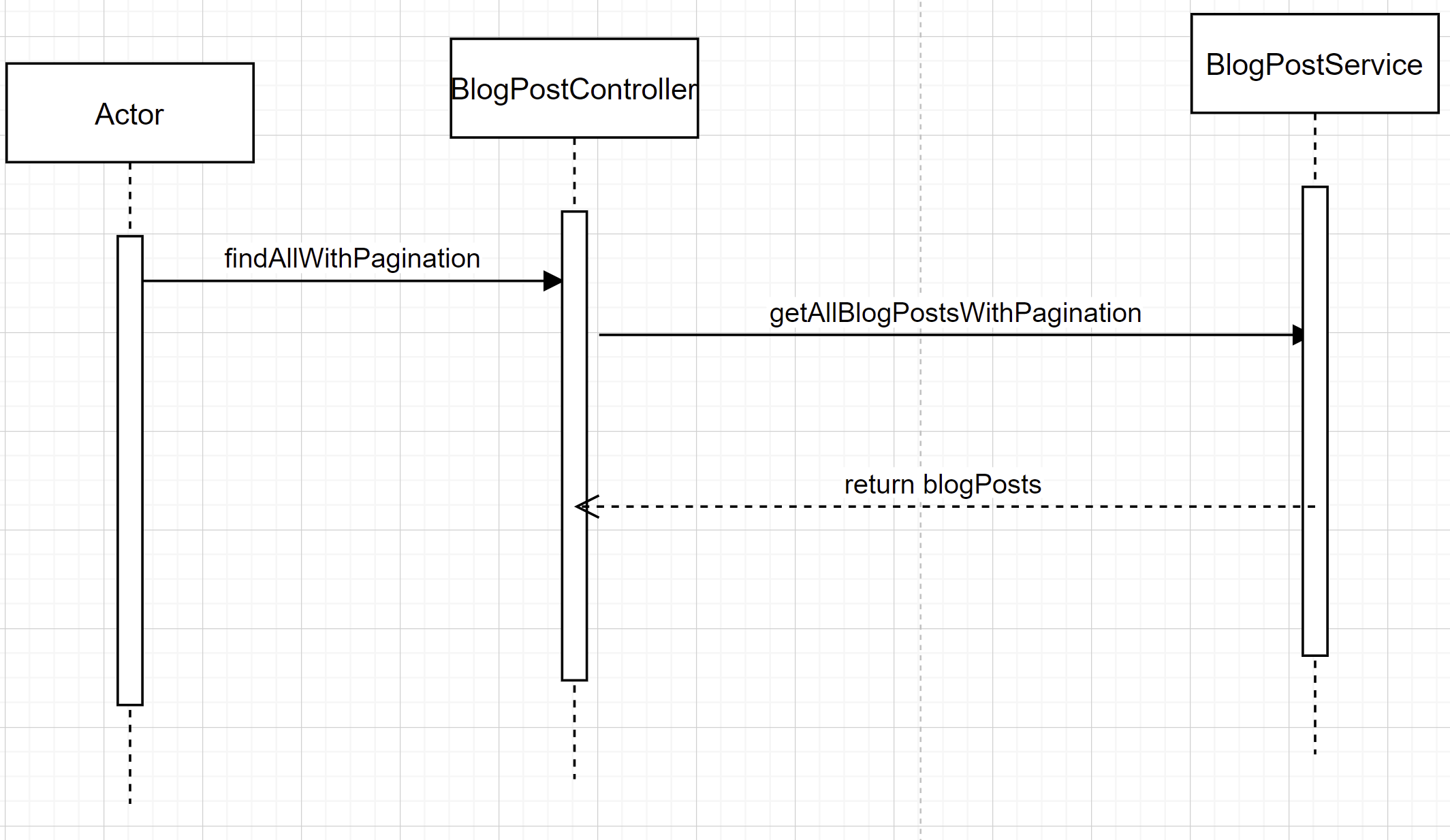
# Sequence Diagrams

## BlogPostController Class

### findAllWithSorting



### findAllWithPagination



### update

