UEK 223

Thema: Documentation

Dokumentinformationen

Dateiname: Documentation-uek-223.docx

Speicherdatum: 05.11.2021

Autoreninformationen

Autor: Dominic Ritzmann

E-Mail: ritzmann@noseryoung.com

Inhaltsverzeichnis

[1 Hauptauftrag 5](#_Toc88057109)

[1.1 Funktionale Anforderungen 5](#_Toc88057110)

[1.1.1 User, Rollen & Autoritäten 5](#_Toc88057111)

[1.1.2 Security 5](#_Toc88057112)

[1.2 Nicht funktionale Anforderungen 5](#_Toc88057113)

[1.2.1 Implementation 5](#_Toc88057114)

[1.2.2 Testing 5](#_Toc88057115)

[1.2.3 Multiuserfähigkeit 5](#_Toc88057116)

[1.2.4 Dokumentation 5](#_Toc88057117)

[2 Arbeitsaufteilung 6](#_Toc88057118)

[3 Funktionalität des Programmes 6](#_Toc88057119)

[4 Endpoints 7](#_Toc88057120)

[5 Use Cases 8](#_Toc88057121)

[5.1 Successful tests 9](#_Toc88057122)

[5.2 Failed Tests 12](#_Toc88057123)

[6 Diagramme 13](#_Toc88057124)

[6.1 Klassen Diagramm 13](#_Toc88057125)

[6.2 Domain Model 14](#_Toc88057126)

[7 Sequence Diagramme 15](#_Toc88057127)

[7.1 BlogPostController Class 15](#_Toc88057128)

[7.1.1 findAll (Get Request) 15](#_Toc88057129)

[7.1.2 findAllWithSorting (Get Request) 16](#_Toc88057130)

[7.1.3 findAllWithPagination (Get Request) 16](#_Toc88057131)

[7.1.4 Save (Post Request) 17](#_Toc88057132)

[7.1.5 Update (Put Request) 17](#_Toc88057133)

[7.1.6 Delete (Delete Request) 18](#_Toc88057134)

[7.2 UserController Class 18](#_Toc88057135)

[7.2.1 findAll (Get Request) 18](#_Toc88057136)

Abbildungsverzeichnis

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

Tabellenverzeichnis

[Tabelle 1 Versionen 3](#_Toc88042443)

[Tabelle 2 Abkürzungen 4](#_Toc88042444)

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 |
| 1.4 | 17.11.2021 | Ritzmann | Sequence Diagramme |

Tabelle 1 Versionen

# 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.

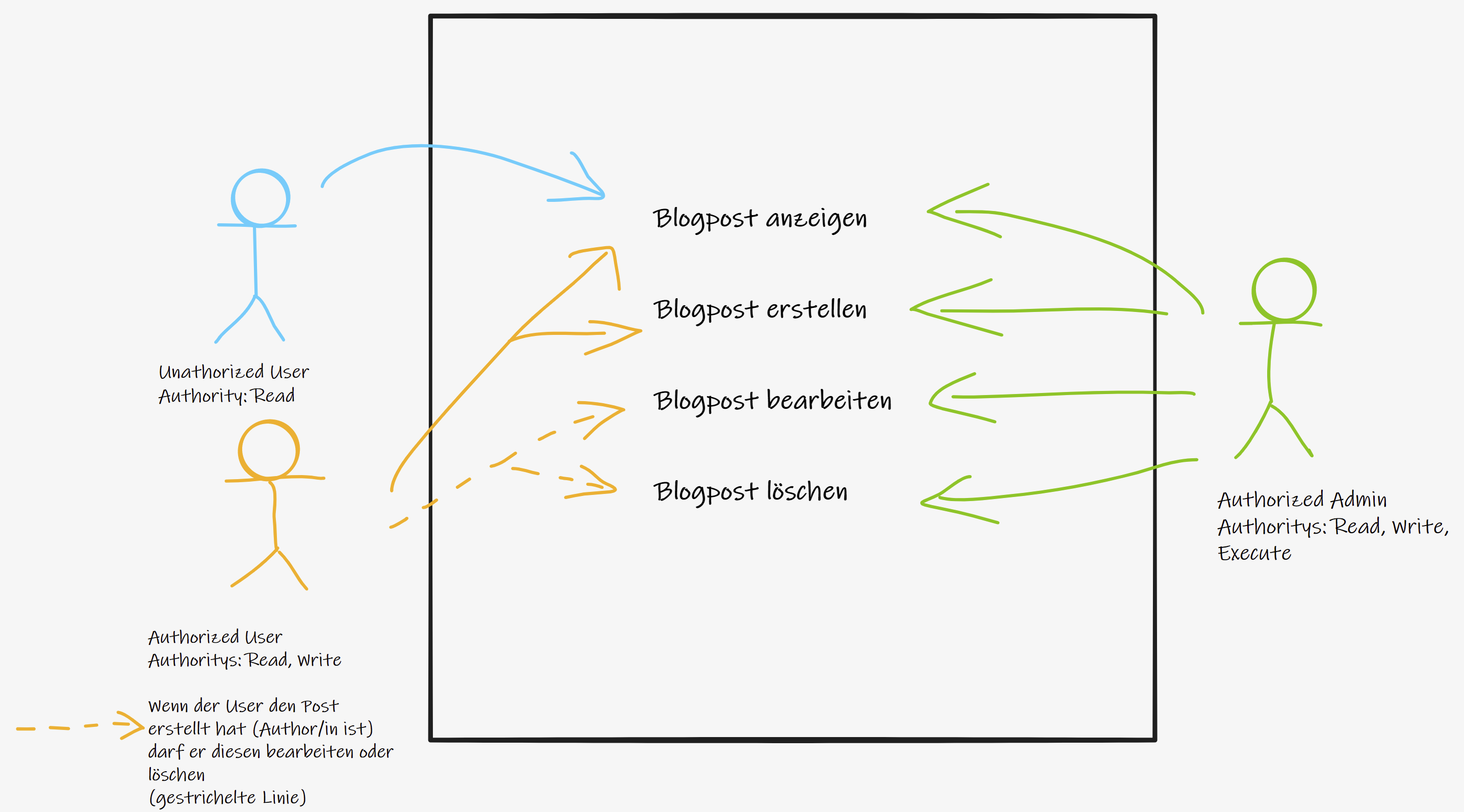
In unserem BlogPostController haben wir 3 Get Endpoints und je 1 Post, Put und Delete Endpoint. Mit dem ersten Endpoint /blogpost bekommt man alle Blogposts angezeigt. Mit dem 2ten Endpoint /blogpost/{field} kann man aufsteigend nach einem bestimmten Feld sortieren. Mit dem letzten Endpoint /blogpost/pagination/{offset}/{pageSize} 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.

## Successful tests

|  |  |
| --- | --- |
| **Use Case:** | Neuer Blogpost kann hinzugefügt werden. |
| **Use Case ID:** | 1 |
| **Kurzberschreibung:** | In Postman wird ein neuer Post Request gemacht. Es wird über die Url <http://localhost:8080/blogpost/> zugegriffen. Unter Body werden Daten als Json mitgeschickt. |
| **Vorbedingung:** | Eingeloggter User |
| **Akteur (Primary):** | Angemeldeter User |
| **Akteur (Secondary):** | keine |
| **Hauptablauf:**   1. Der User gibt seine Daten unter Authorization mit einer Basic Auth an. 2. Der POST Request wird ausgewählt. 3. Die Url <http://localhost:8080/blogpost/> wird eingegeben 4. WHILE(SOLANGE) keine Daten eingegeben wurden, wird Schritt 5 wiederholt 5. Unter Body wird raw und JSON ausgewählt. Unten gibt man dann seine Daten an, welche in die Datenbank gespeichert werden. Die id muss nicht angegeben werden. Bei der User Id gibt man seine eigene an.   Beispiel: {          "title": "test",          "user\_id": "ceb9c131-5f7a-49b0-98b1-e7943ed35172",          "text": "test",          "category": "category"      }   1. Am Schluss drückt man auf «Send» | |
| **Nachbedingung:** | In der Datenbank wird der Eintrag unter der Tabelle blog\_posts |
| ***Alternative Flows:***  ***keine*** | |

|  |  |
| --- | --- |
| **Use Case:** | Neuer Blogpost kann hinzugefügt werden. |
| **Use Case ID:** | 2 |
| **Kurzberschreibung:** | In Postman wird ein neuer Post Request gemacht. Es wird über die Url <http://localhost:8080/blogpost/> zugegriffen. Unter Body werden Daten als Json mitgeschickt. |
| **Vorbedingung:** | Eingeloggter User |
| **Akteur (Primary):** | Admin |
| **Akteur (Secondary):** | keine |
| **Hauptablauf:**   1. Der User gibt seine Daten unter Authorization mit einer Basic Auth an. 2. Der POST Request wird ausgewählt. 3. Die Url <http://localhost:8080/blogpost/> wird eingegeben 4. WHILE(SOLANGE) keine Daten eingegeben wurden, wird Schritt 5 wiederholt 5. Unter Body wird raw und JSON ausgewählt. Unten gibt man dann seine Daten an, welche in die Datenbank gespeichert werden. Die id muss nicht angegeben werden. Bei der User Id gibt man seine eigene an.   Beispiel: {          "title": "test",          "user\_id": "ceb9c131-5f7a-49b0-98b1-e7943ed35172",          "text": "test",          "category": "category"      }   1. Am Schluss drückt man auf «Send» | |
| **Nachbedingung:** | In der Datenbank wird der Eintrag unter der Tabelle blog\_posts |
| ***Alternative Flows:***  ***keine*** | |

|  |  |
| --- | --- |
| **Use Case:** | Blogposts können aufsteigend sortiert und ausgegeben werden |
| **Use Case ID:** | 3 |
| **Kurzberschreibung:** | In Postman wird ein neuer Get Request gemacht. Es wird über die Url <http://localhost:8080/blogpost/>{sortingField} zugegriffen. Beim sortingField schreibt man beispielsweise title. Also so -> <http://localhost:8080/blogpost/>title |
| **Vorbedingung:** | keine |
| **Akteur (Primary):** | Nicht angemeldeter User |
| **Akteur (Secondary):** | keine |
| **Hauptablauf:**   1. Die Url <http://localhost:8080/blogpost/>title wird eingegeben 2. Man wählt den GET Request aus 3. Unter Authorization muss nichts angegeben werden, da jeder die Blogposts sehen kann. 4. Am Schluss drückt man auf «Send» | |
| **Nachbedingung:** | Alle Blogposts werden nach dem Filter aufsteigend gefiltert und ausgegeben |
| ***Alternative Flows:***  ***keine*** | |

|  |  |
| --- | --- |
| **Use Case:** | Blogposts können aufsteigend sortiert und ausgegeben werden |
| **Use Case ID:** | 4 |
| **Kurzberschreibung:** | In Postman wird ein neuer Get Request gemacht. Es wird über die Url <http://localhost:8080/blogpost/>{sortingField} zugegriffen. Beim sortingField schreibt man beispielsweise title. Also so -> <http://localhost:8080/blogpost/>title |
| **Vorbedingung:** | keine |
| **Akteur (Primary):** | Angemeldeter User |
| **Akteur (Secondary):** | keine |
| **Hauptablauf:**   1. Die Url <http://localhost:8080/blogpost/>title wird eingegeben 2. Man wählt den GET Request aus 3. Unter Authorization muss nichts angegeben werden, da jeder die Blogposts sehen kann. 4. Am Schluss drückt man auf «Send» | |
| **Nachbedingung:** | Alle Blogposts werden nach dem Filter aufsteigend gefiltert und ausgegeben |
| ***Alternative Flows:***  ***keine*** | |

|  |  |
| --- | --- |
| **Use Case:** | Blogposts können aufsteigend sortiert und ausgegeben werden |
| **Use Case ID:** | 5 |
| **Kurzberschreibung:** | In Postman wird ein neuer Get Request gemacht. Es wird über die Url <http://localhost:8080/blogpost/>{sortingField} zugegriffen. Beim sortingField schreibt man beispielsweise title. Also so -> <http://localhost:8080/blogpost/>title |
| **Vorbedingung:** | keine |
| **Akteur (Primary):** | Admin |
| **Akteur (Secondary):** | keine |
| **Hauptablauf:**   1. Die Url <http://localhost:8080/blogpost/>title wird eingegeben 2. Man wählt den GET Request aus 3. Unter Authorization muss nichts angegeben werden, da jeder die Blogposts sehen kann. 4. Am Schluss drückt man auf «Send» | |
| **Nachbedingung:** | Alle Blogposts werden nach dem Filter aufsteigend gefiltert und ausgegeben |
| ***Alternative Flows:***  ***keine*** | |

## Failed Tests

|  |  |
| --- | --- |
| **Use Case:** | Neuer Blogpost kann hinzugefügt werden. |
| **Use Case ID:** | 6 |
| **Kurzberschreibung:** | In Postman wird ein neuer Post Request gemacht. Es wird über die Url <http://localhost:8080/blogpost/> zugegriffen. Unter Body werden Daten als Json mitgeschickt. |
| **Vorbedingung:** | Eingeloggter User |
| **Akteur (Primary):** | Nicht Angemeldeter User |
| **Akteur (Secondary):** | keine |
| **Hauptablauf:**   1. Der User gibt bei der Authorization «no Auth» an 2. Der POST Request wird ausgewählt. 3. Die Url <http://localhost:8080/blogpost/> wird eingegeben 4. WHILE(SOLANGE) keine Daten eingegeben wurden, wird Schritt 5 wiederholt 5. Unter Body wird raw und JSON ausgewählt. Unten gibt man dann seine Daten an, welche in die Datenbank gespeichert werden. Die id muss nicht angegeben werden. Bei der User Id gibt man seine eigene an.   Beispiel: {          "title": "test",          "user\_id": "ceb9c131-5f7a-49b0-98b1-e7943ed35172",          "text": "test",          "category": "category"      }   1. Am Schluss drückt man auf «Send» 2. Der neue Blogpost wird nicht erstellt, da der User nicht angemeldet ist, der nicht eingeloggte User hat nämlich nur die Berechtigung Blogposts anzuschauen. | |
| **Nachbedingung:** | In der Datenbank wird der Eintrag unter der Tabelle blog\_posts |
| ***Alternative Flows:***  ***keine*** | |
|  | |

# 

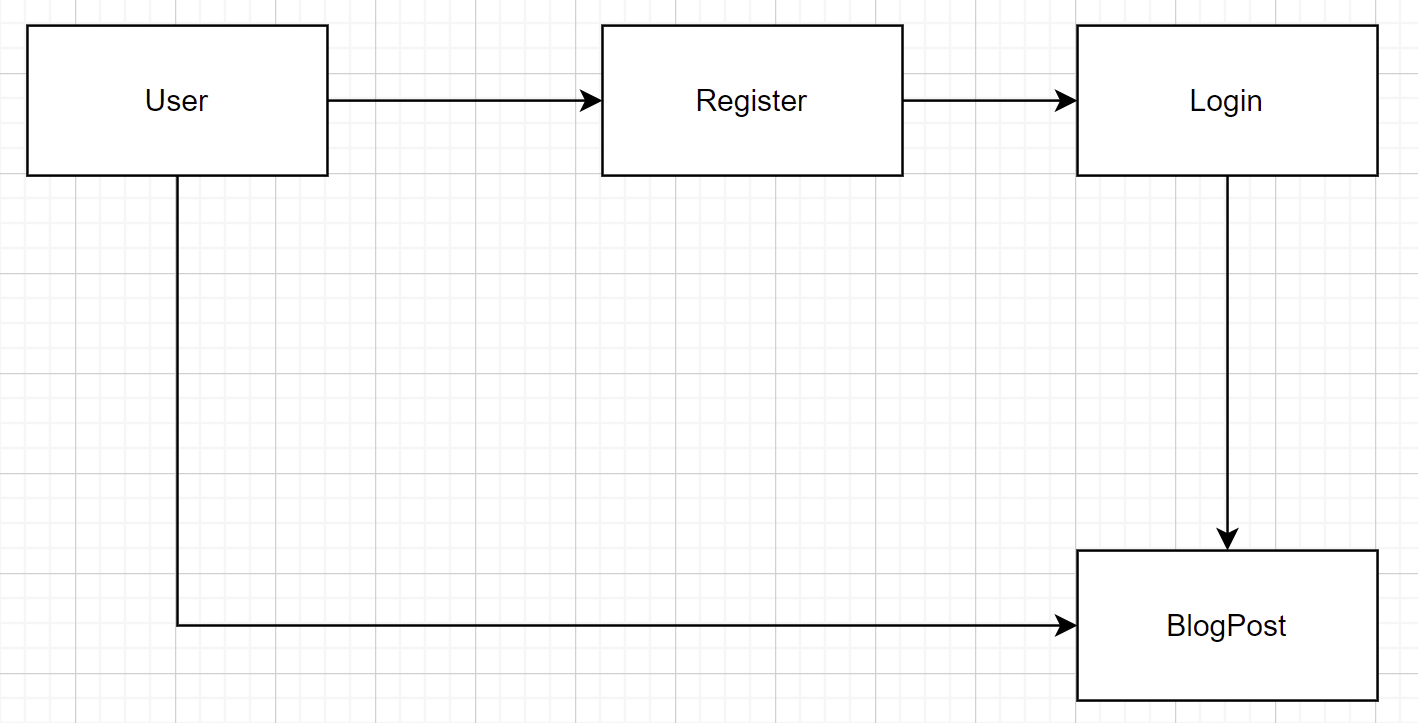
# Diagramme

## Klassen Diagramm

Ein Bild, das Text, drinnen, computer, Screenshot enthält.

Automatisch generierte Beschreibung

## Domain Model

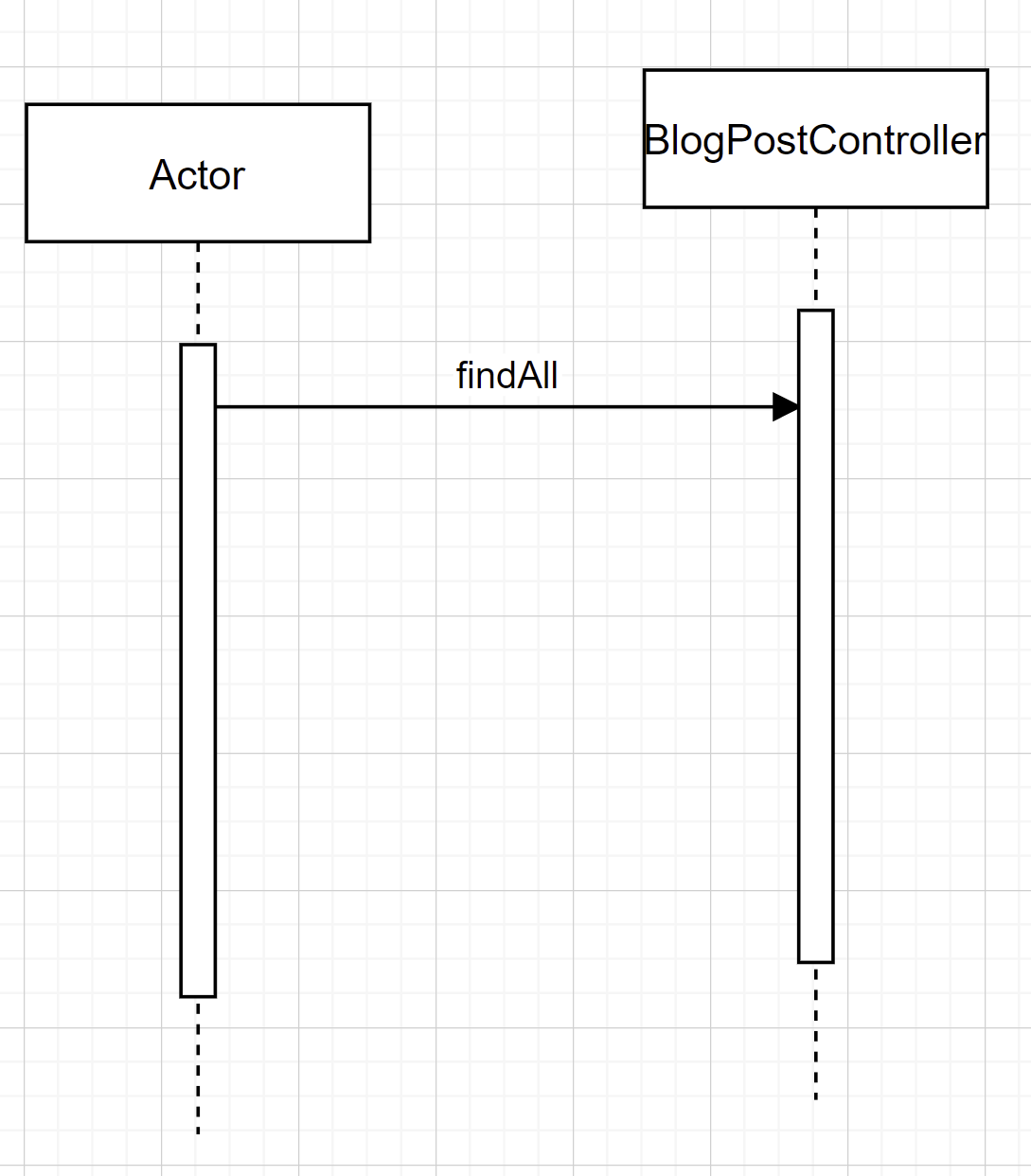


Ein User, der nicht angemeldet ist, kann direkt auf die Blogposts zugreifen. Allerdings kann er sie nur anschauen. Er kann weder einen neuen Post erstellen, einen bestehenden bearbeiten oder löschen. Wenn der User sich registriert und anmeldet, kann er auf die Blogposts zugreifen. Er kann Sie anschauen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

# Sequence Diagramme

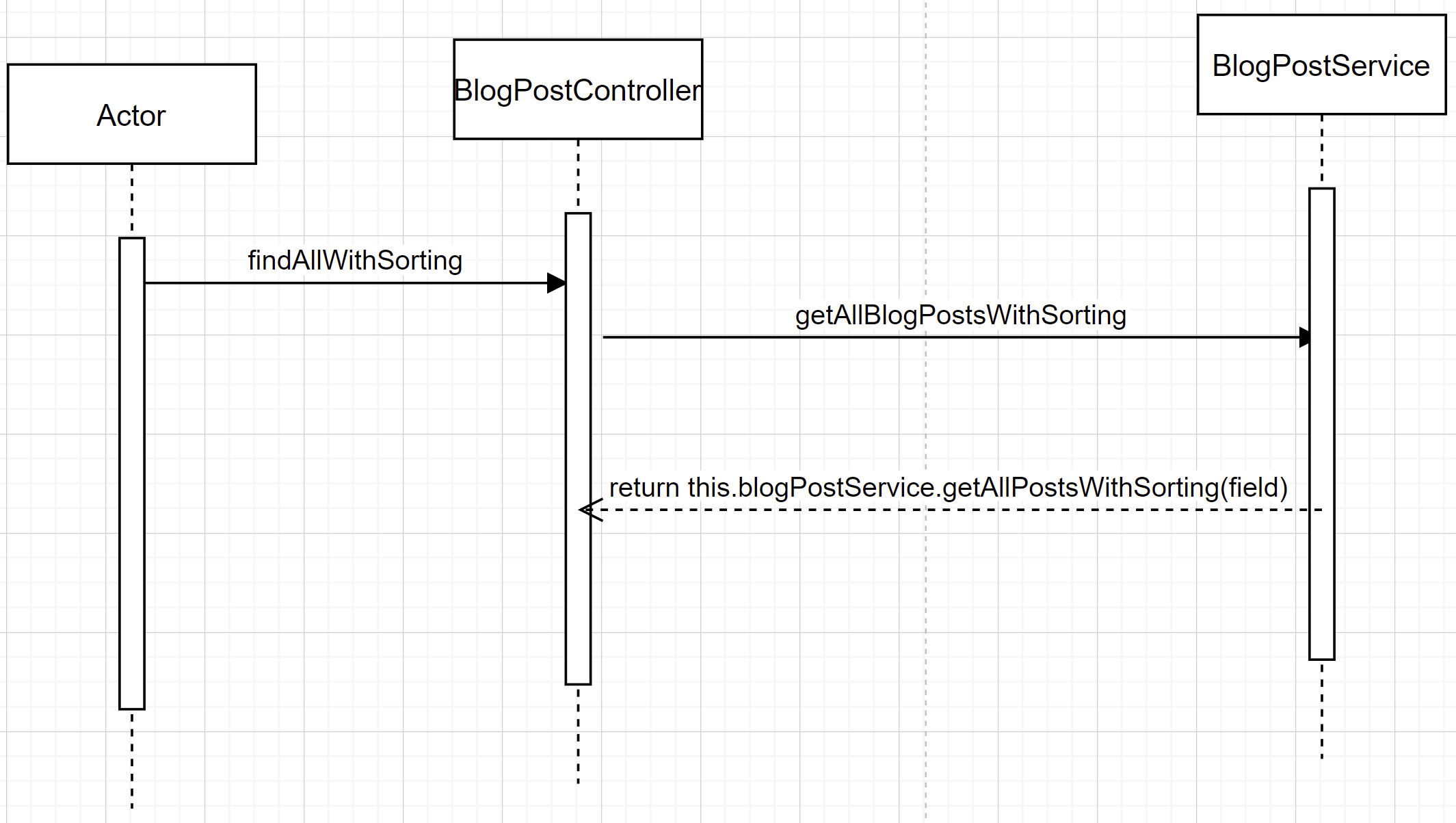
## BlogPostController Class

### findAll (Get Request)



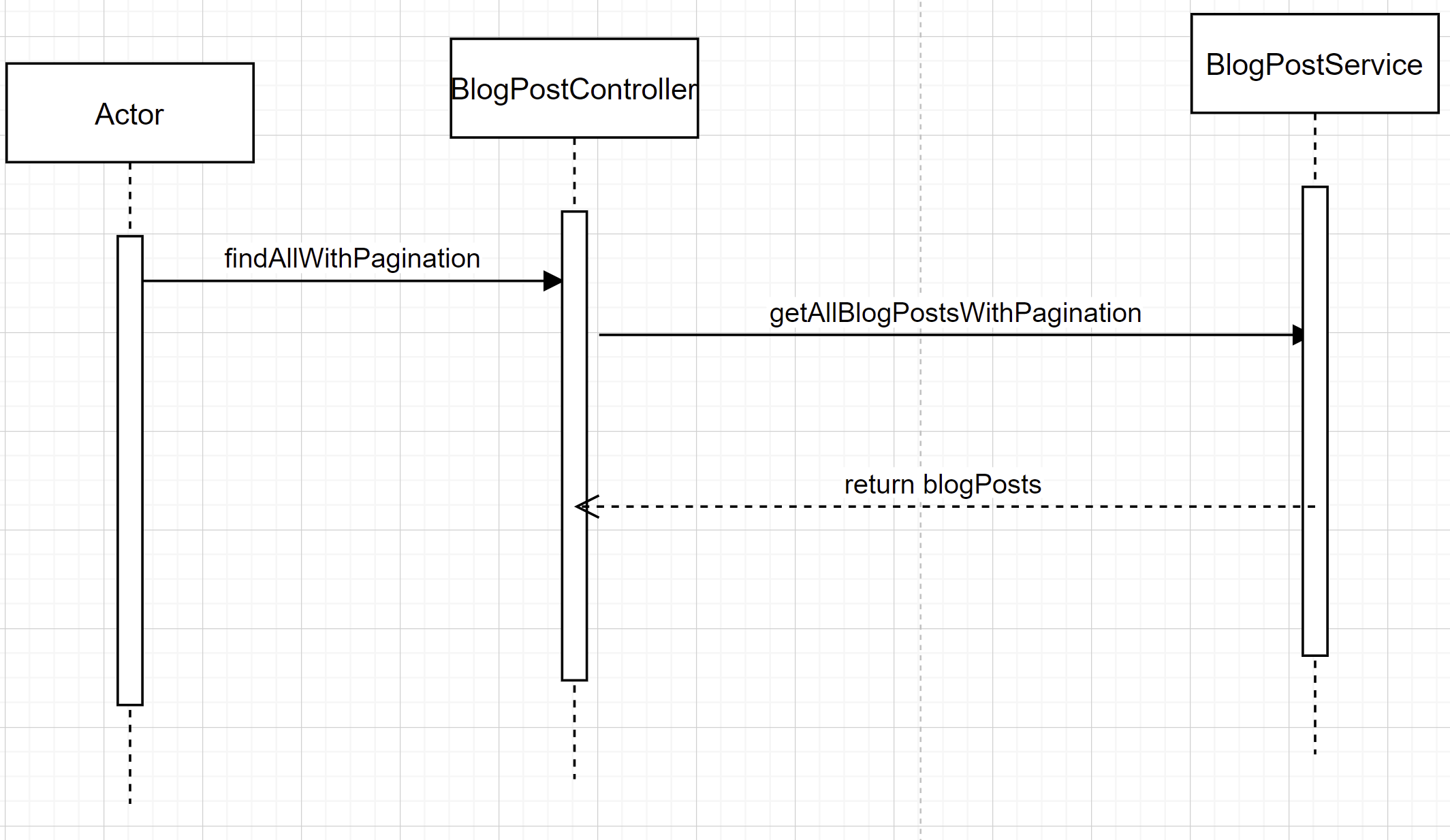
Mit der findAll Methode können wir alle Blogposts aus der Datenbank herauslesen. Dazu braucht es keine zusätzlichen Berechtigungen.

### findAllWithSorting (Get Request)



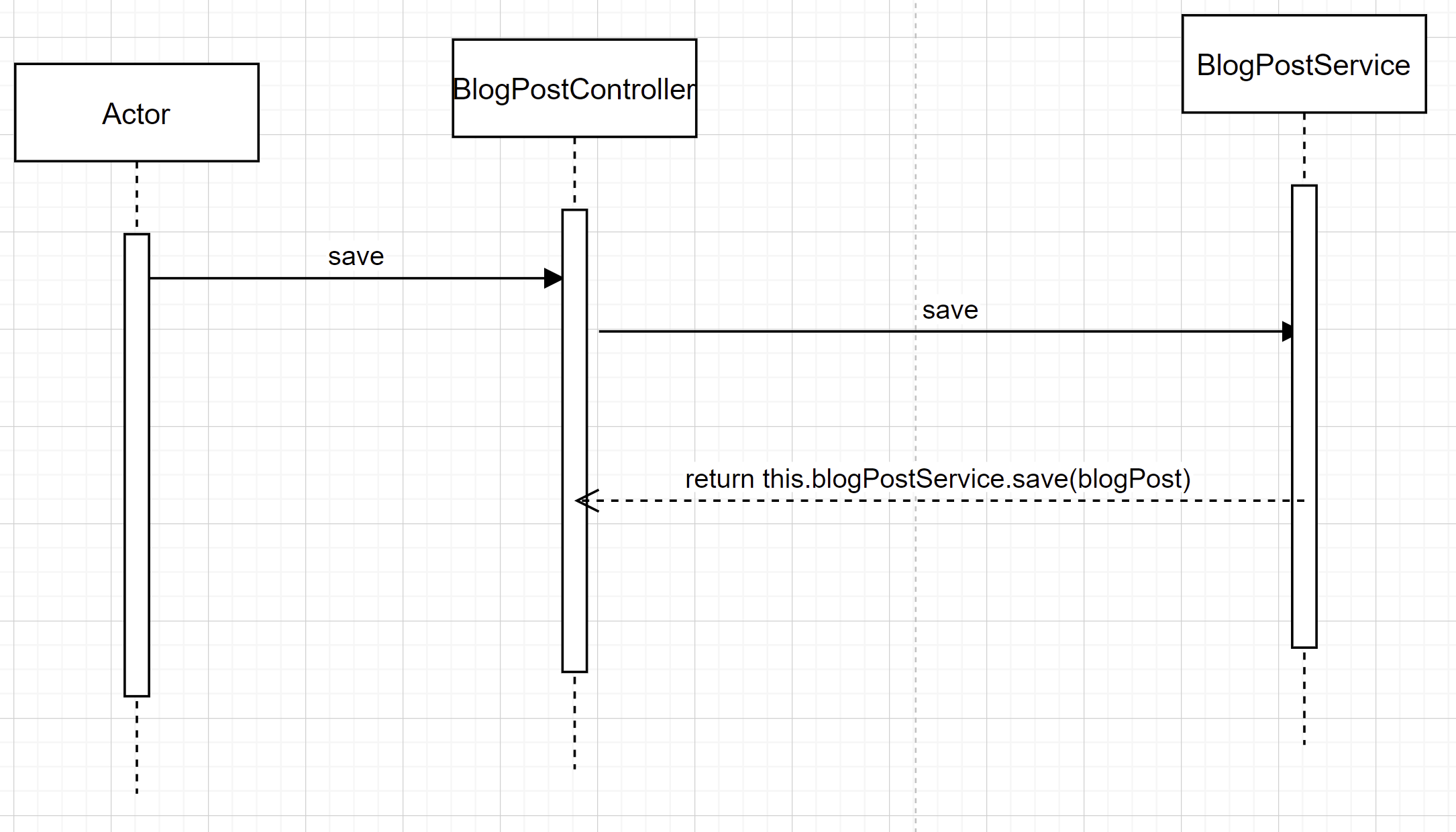
Mit der findAllWithSorting Methode können wir alle Blogposts aus der Datenbank herauslesen und nach einem Feld aufsteigend sortieren. Dazu braucht es keine zusätzlichen Berechtigungen.

### findAllWithPagination (Get Request)



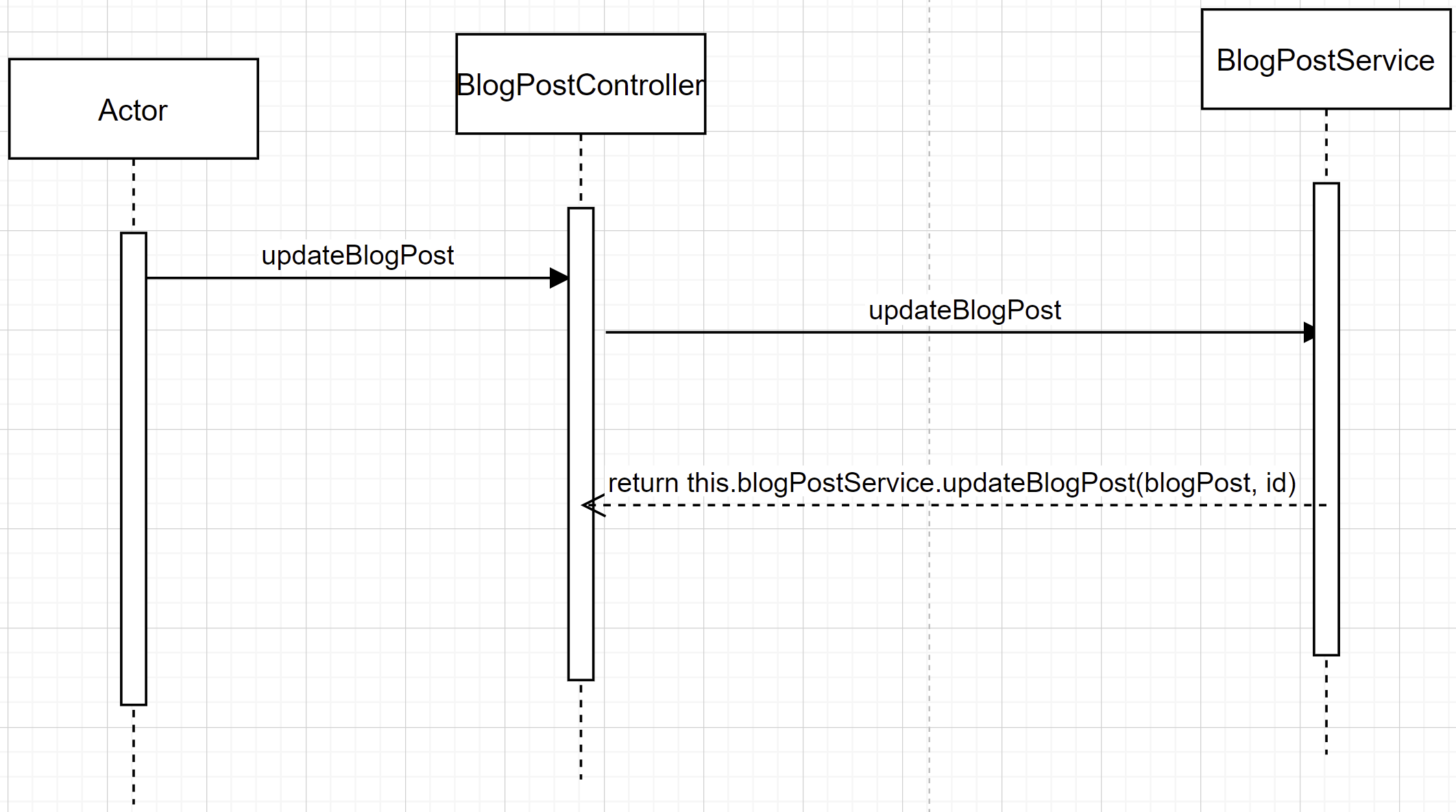
Mit der findAlWithPaginationl Methode können wir alle Blogposts aus der Datenbank herauslesen und Anzahl Beiträge pro Seite und wie viele Seiten es gibt bestimmen. Dazu braucht es keine zusätzlichen Berechtigungen.

### Save (Post Request)



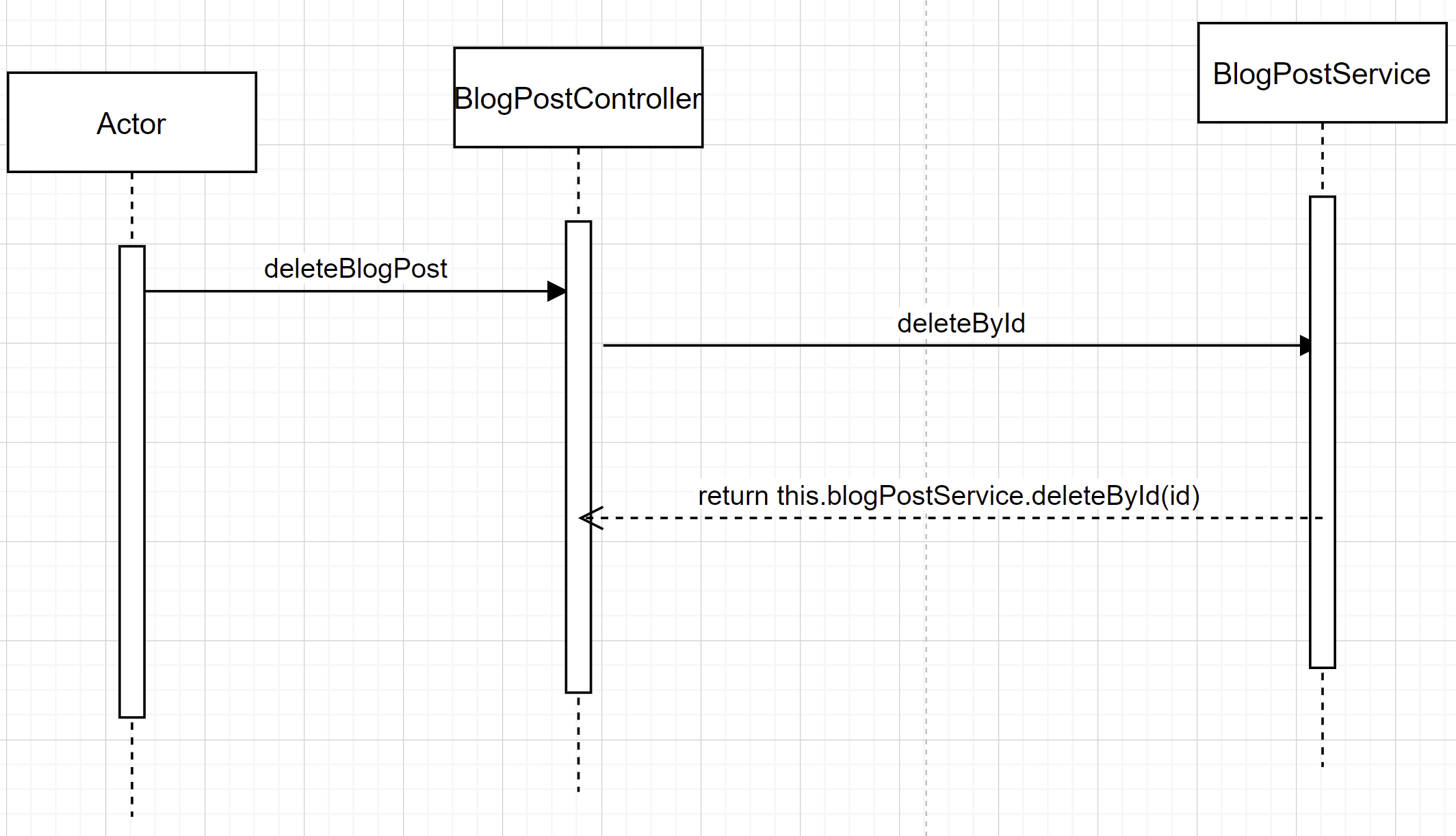
Mit der save Methode können wir einen neuen Blogpost erstellen. Um diese Methode nutzen zu können, muss man angemeldet sein.

### Update (Put Request)



Mit der Update Methode kann man einen bestehenden Blogpost bearbeiten. Um einen Blogpost zu bearbeiten, muss man entweder der/die Author/in oder Admin sein.

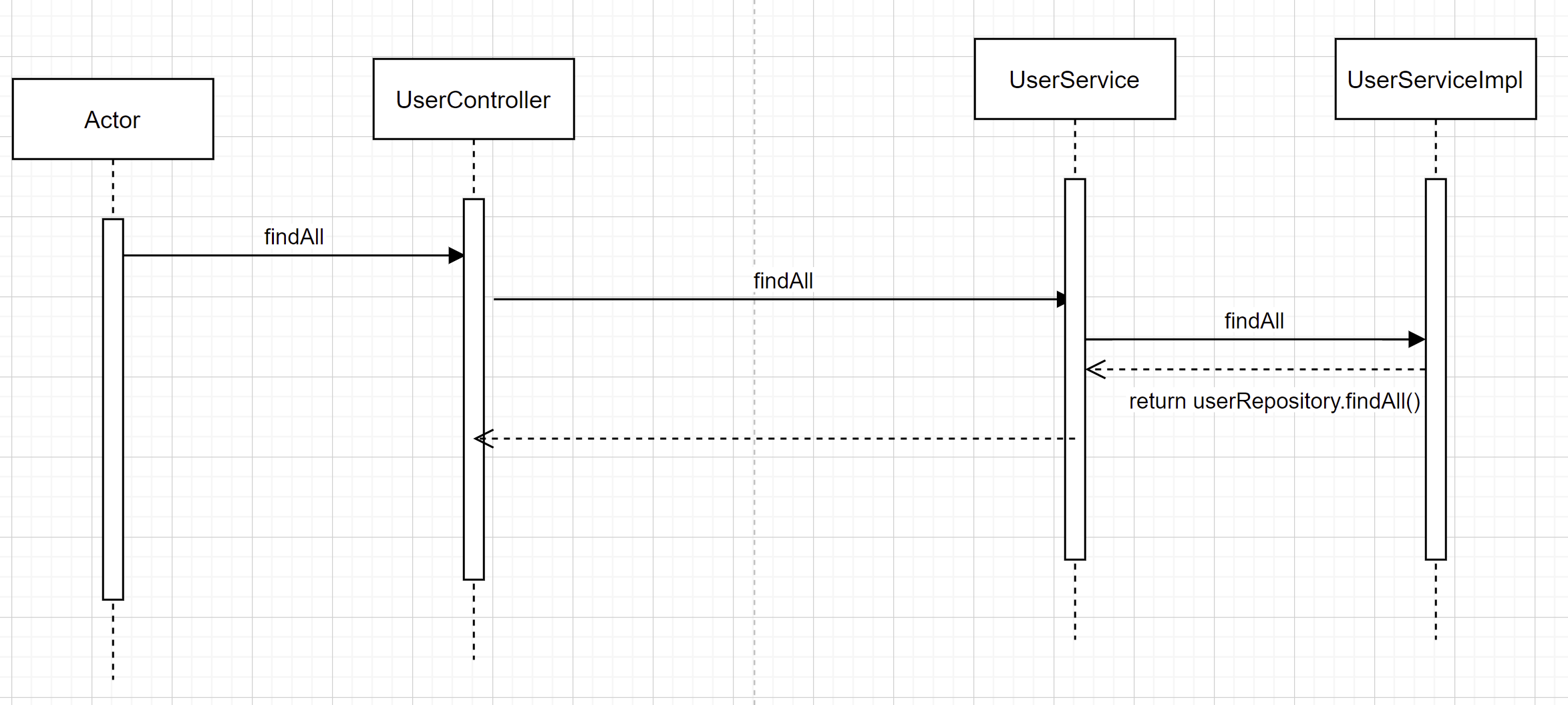
### Delete (Delete Request)



Mit der delete Methode können wir einen bestehenden Blogpost löschen. Um einen Blogpost zu löschen, muss man entweder der/die Author/in oder Admin sein.

## UserController Class

### findAll (Get Request)



Mit der findAll Methode können wir alle User, die registriert sind, aus der Datenbank lesen. Dies ist nur mit Administrator Rechten möglich.