ueK 223

Thema: Multi-User-Applikationen objektorientiert realisieren

Dokumentinformationen

Dateiname: Dokumentation-uek223.docx
Speicherdatum: 03.03.2022

Autoreninformationen

Autoren: Andrin Klarer, Davide Marcoli

Inhaltsverzeichnis

1. Einleitung

Sinn und Zweck

Das vorliegende Dokument dient der Dokumentation des ueK's 223. Wir bekamen den Auftrag, eine Multi-User-Applikation zu realisieren.

Problemstellung

Unser Auftrag war es für einen Kunden (Berufsbildner) eine Spring Boot Applikation zu planen, dokumentieren, umzusetzen und testen. Es sollen neben Roles, Authorities und Usern ebenfalls BlogPosts umgesetzt werden. Die Aufgabenstellung sieht nicht vor Werkzeuge zur Datenverwaltung (Masken, Formulare, Berichte) zu erstellen. Unser Programm sollte ebenfalls einen Login Bereich bieten, bei dem sich die User Autorisieren müssen.

2. Planung

Anforderungsanalyse

Terminplanung

Das Projekt, inklusive Dokumentation soll innert den fünf Tagen an welchen der ueK stattfindet fertiggestellt.

Vorgehensweise

Wir arbeiten mit Github als Version Control Plattform. Andrin und Ich sind beide sehr gut mit dieser Seite und Arbeitsweise bekannt und konnten darum schnell einsteigen. Als Task-Tracking Tool brauchen wir Github Projects. Dies ist praktisch, da es direkt mit dem Repository verbunden ist und somit alles zusammenhängt. Wir haben dort mit verschiedenen Labels wie zum Beispiel «Bug» und «Documentation», damit wir immer wissen, um was es in diesem Task geht. Wir haben ein Kanban-Board mit 3 Spalten, «To Do», «In Progress» und «Done».

3. Technisches Handbuch

Naming Convention

Tabelle 1: Naming Convention

Name	Beschreibung
Git-Commits	[Category<Fix, Add, Delete, Update, Refactor>] [Description] [Optional<Changed File>]
Tabellen-Attribut	snake_case
Variablen	camelCase
Methoden	camelCase
Klassen	PascalCase

Klassen

Spring Boot arbeitet mit 3 Layern, die ich in diesem Abschnitt genauer erklären werde:

Web-Layer

Im Web-Layer sind alle Endpoints definiert. Ebenfalls sind dort Exception Handler vorhanden, die sehr wichtig für aussagekräftige Fehlermeldungen sind.

Diese Klassen brauchen die Annotation @RestController

Service-Layer

Im Service Layer ist die ganze Business-Logik stationiert. Dort wird gerechnet, validiert und verändert.

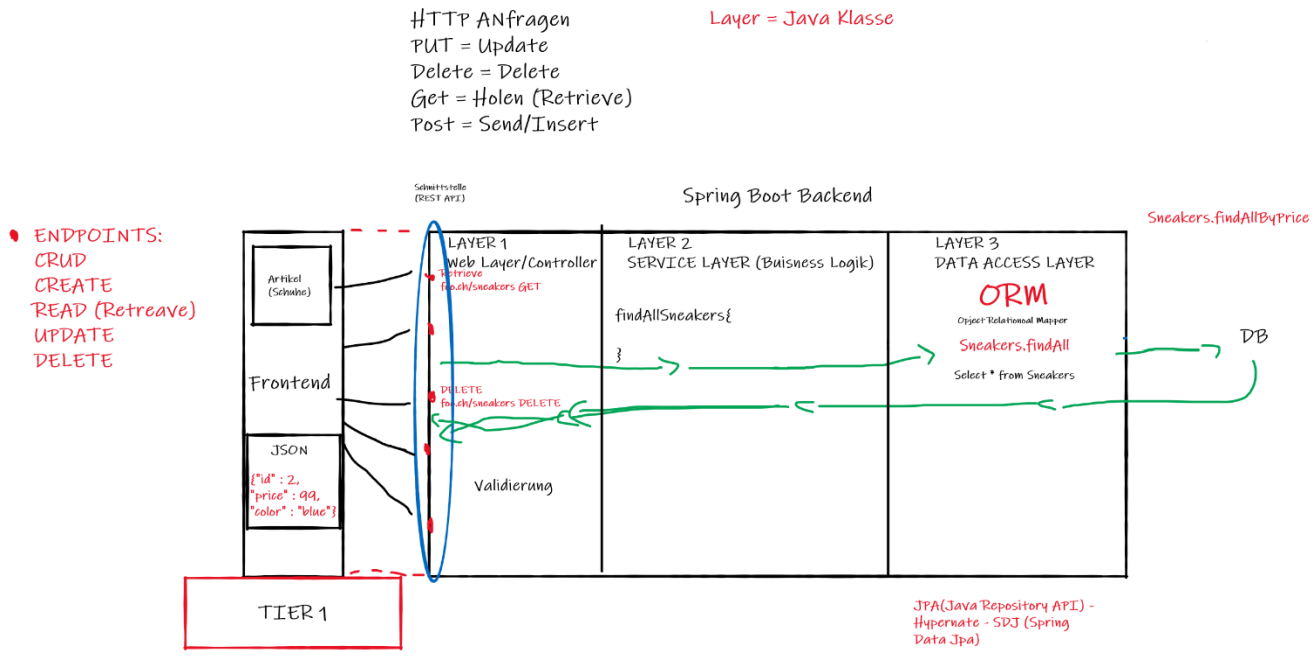
Diese Klassen brauchen die Annotation @Service

Data-Access-Layer

Im Data-Access werden die Daten vom Programm zu der Datenbank und umgekehrt geschickt. Wir extenden das Interface mit JpaRepository, damit wir viele vorgeschriebene Funktionen haben.

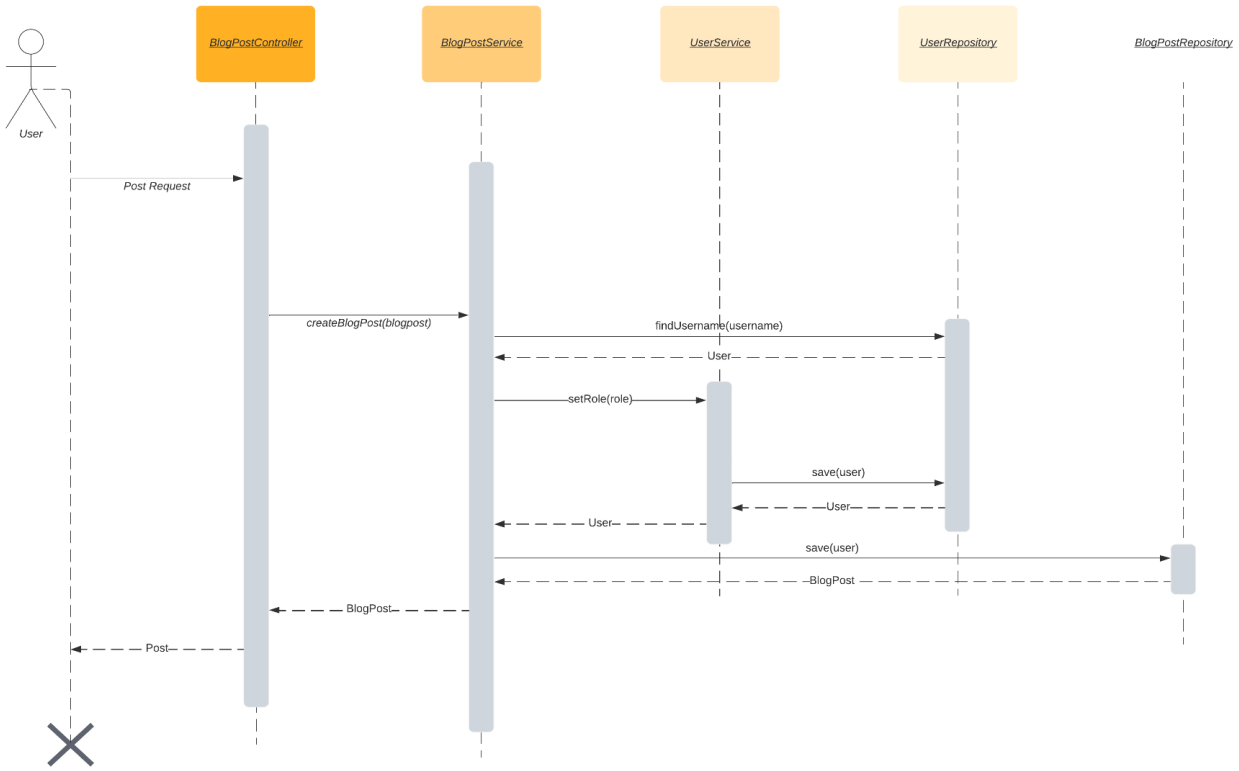
Diese Klassen brauchen die Annotation @Repository

Visualisierung der 3 Layers (inkl. Frontend):

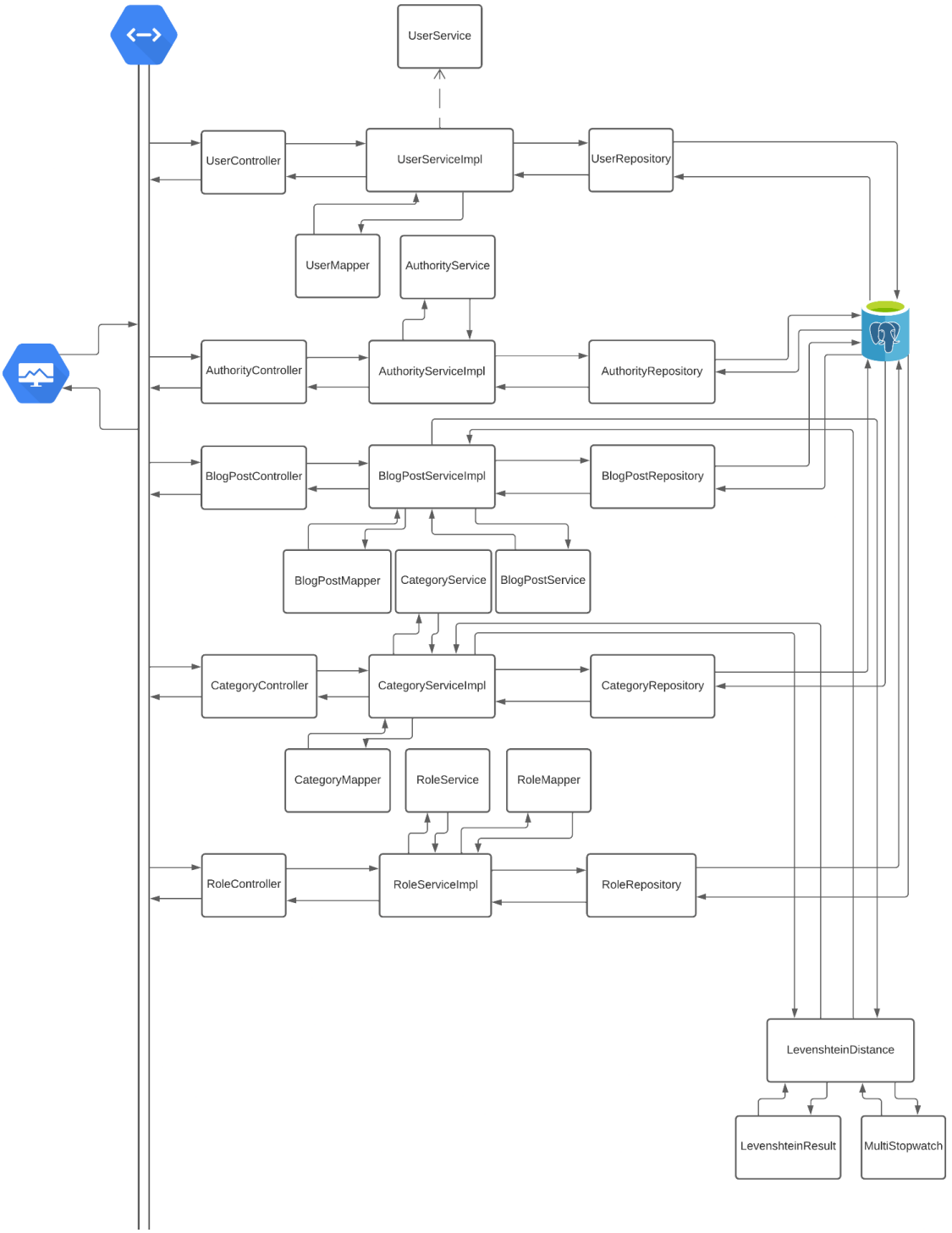


Sequence Diagram

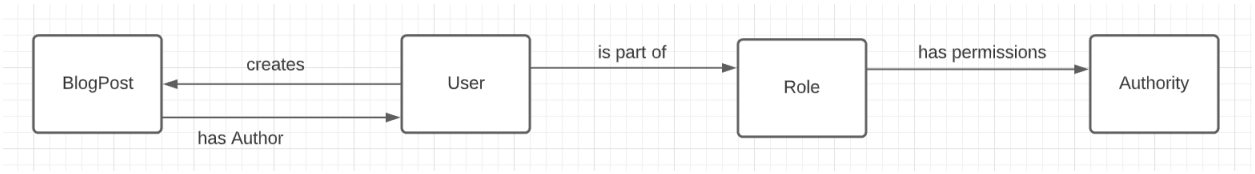
Add BlogPost:



Domain Diagram



Korrekte Version:



Use Cases

Update BlogPost

Use Case	Update BlogPost
Use Case ID	1
Description	User submits an updated blogpost which should override an existing blogpost.
Precondition	The user has a valid account.
Actors (Primary)	User
Actors (Secondary)	none
Procedure: 1. The user navigates to [domain]/api/blog-post. 2. The user sends a put request with a blog post (eg. via form), and a UUID at the end of the URL to the backend. 3. The system checks if the user has the authority to do so 4. The system verifies that a blogpost with the provided UUID already exists. 5. IF the user is author of the blogpost, allow to update 5.1 The system copies unchangeable data from the old to the new post. 5.2 The system attempts to update the post. 6. The user can now see his updated blog post	
Postcondition:	none
Alternative flows: invalid blogpost, no access	

Use Case	Create new user(-account): invalid blogpost
Use Case ID	2.1
Description	The system verifies if the blogpost with the provided id exists.
Precondition	none
Actors (Primary)	User
Actors (Secondary)	none
Procedure: 1. Flow starts at step 4 in main procedure 2. The system informs the user that the ‘blogpost-to-update’ couldn't be found	
Postcondition:	none

Use Case	Create new user(-account): no access
Use Case ID	2.2
Description	The system verifies that the user is author of the ‘blogpost-to-update’
Precondition	The given ‘blogpost-to-update’ exists
Actors (Primary)	User
Actors (Secondary)	none
Procedure: 3. Flow starts at step 4 in main procedure 4. The system informs the user he is not allowed to update this blogpost	
Postcondition:	none

Create new User

Use Case	Create new user(-account)
Use Case ID	2
Description	An user without an account attempts to sign up.
Precondition	
Actors (Primary)	User
Actors (Secondary)	none
Procedure: 1. The user navigates to [domain]/api/user. 2. The user sends a post request with the data for a new user (eg. via form) to the backend. 2.1 The system checks the input of the user. 2.2 The system encrypts the password 2.3 The system assigns the role USER by default to the new user account 2.4 The system attempts to save the blog post in the database 3. The user can now see his new user.	
Postcondition:	The user can now log in in his own account
Alternative flows: invalid email, duplicate email, duplicate username	

Use Case	Create new user(-account): invalid email
Use Case ID	2.1
Description	The system checks the email and gives feedback
Precondition	The user enters an invalid email
Actors (Primary)	User
Actors (Secondary)	none
Procedure: 5. Flow starts at step 2.1 in main procedure 6. The system informs the user that his email is invalid	
Postcondition:	none

Use Case	Create new user(-account): duplicate email
Use Case ID	2.2
Description	The system informs the user about the email that's already used.
Precondition	The user enters an email that's already in use by another account.
Actors (Primary)	User
Actors (Secondary)	none
Procedure: 1. Flow starts at step 2.1 in main procedure 2. The system informs the user that his email is already used	
Postcondition:	none

Use Case	Create new user(-account): invalid email
Use Case ID	2.3
Description	The system checks the input and gives feedback
Precondition	The user gives an invalid email

Actors (Primary)	User
Actors (Secondary)	none
Procedure: 1. Flow starts at step 2.1 in main procedure 2. The system informs the user that his username is already in use.	
Postcondition:	none

Endpoints

Wie haben in unserer Spring Applikation in allen Controllern Endpoints verbaut. Die Rest Endpoints reagieren auf die HTTP-Status GET, POST, PUT und DELETE.

Die Swagger-Dokumentation der Endpoints findet man unter der URL http://localhost:8080/swagger-ui/index.html/

Known Issues

1. Deleting category doesn't work, because of foreign key constraints. We have discussed this and agreed on leaving it away since it's an edge case and we would dive deeper into it and learn how to fix it during an advanced course.

   The same happens when deleting roles or authorities.

Testing

Postman Tests

Postman Kollektion ist im Repository. Es kann von dort heruntergeladen und in Postman hinein geladen werden.

Postman Flow

Unit Tests

Sie können mit dem Befehl `gradle test` (`./gradlew test` in Powershell).

Reflektion

Unserer Wahrnehmung nach, machten wir unser Team gemeinsam zu einem erfolgreichen Team. Wir haben uns von Anfang an gut organisiert, um unsere Arbeit im späteren Verlauf viel einfacher zu gestalten. Die Github-Issues haben uns geholfen, da wir so am Morgen jedes Tages schauen konnten, was für diesen Tag ansteht. Das Zuweisen von Aufgaben hat so auch perfekt geklappt und wir wussten immer, wo der andere gerade am Arbeiten ist. Wir haben den Main-Branch als Base benutzt und von dort aus Feature- und Bugfix-Branches erstellt. Auf diesen konnte die jeweilige Person tun, was sie wollte, sogar nicht fertigen Code pushen. Dies war uns sehr wichtig, da wir so auch kleinere Experimente durchführen konnten, ohne dass der Haupt-Branch nicht mehr funktioniert. Falls wir mit dem Tasks oder Bugfix fertig waren, für den dieser Task bestimmt war, haben wir uns kurz abgesprochen und dann diesen Branch zurück gemergt. Wir hatten manchmal Merge Konflikte, die wir aber fast immer schnell und ohne Probleme lösen konnten. Einzig die Swagger Dokumentation musste Andrin leider über 3 Mal schreiben, da es beim Mergen zu Überschreibungen kam. Dies war zwar nervig, führte aber nicht zu grossen Verzögerungen. Wir hatten Glück, dass wir beide das Thema gut verstanden und so wenig Fragen zum Thema stellen mussten. Wir hatten während diesen 5 Tagen auch Zeit, uns um andere Mitlernende zu kümmern, falls diese Fragen hatten. Zusammen mit Lazar haben wir ebenfalls während den Inputs das Wichtigste aufgeschrieben, damit wir uns gut auf das Fachgespräch vorbereiten können. Wir nutzten dafür das fantastische Tool Notion. Hier sind unsere Notizen. Wir werden auch in den noch folgenden ÜK's diesen Tool nutzen, da kollaboratives Arbeiten so extrem einfach und effektiv ist.