Fachhochschule Nordwestschweiz
Hochschule für Technik

Google Docs Light

Workshop Web - FS22

21. Mai 2022

Studenten	P. Schmucki, J. Villing, K. Zellweger
Dozenten	D. König, S. Meichtry, J. Luthiger
Studiengang	Informatik
Hochschule	Hochschule für Technik

Inhaltsverzeichnis

1 Summary

2 Systemübersicht

2.1 Services

2.2 Sequenz

2.3 Applikationsprotokoll

2.4 Benutzerverwaltung

Es ist keine persistente Benutzerverwaltung mit Registrationsprozess implementiert. Nach erstmaligem Anmelden in der Applikation mit einem globalen Benutzer, wird ein zufälliger Author erstellt. Die Daten des Authors werden im Local Storage des Browsers gespeichert, sodass bei erneutem Öffnen der Applikation der gleiche Author wiederverwendet wird.

3 Frontend

Das Fronted der TeamDocument Applikation ist als React SPA entwickelt. Einmal angemeldet kann ein Benutzer an der kollaborativen Bearbeitung des Dokumentes teilnehmen.

Folgende Interaktionen sind möglich:

- Ändern des eigenen Namens
- Hinzufügen eines neuen Paragrafen
- Bearbeitung bestehender Paragrafen
- Sperren des Paragrafen an dem gerade gearbeitet wird (implizit)
- Verschieben von Paragrafen innerhalb des Dokuments
- Löschen eines bestehenden Paragrafen
- Wiederherstellen des zuletzt gelöschten Paragrafen (Hidden Feature)

Des Weiteren werden folgende Informationen auf dem UI dargestellt:

- Name des ursprünglichen Authors eines Paragrafen
- Name des Authors welcher aktiv einen Paragrafen bearbeitet.
- Highlight des eigenen aktuellen Paragrafen
- Liste mit allen Dokumentupdates in chronologischer Reihenfolge
- Avatare aller aktiven Benutzer

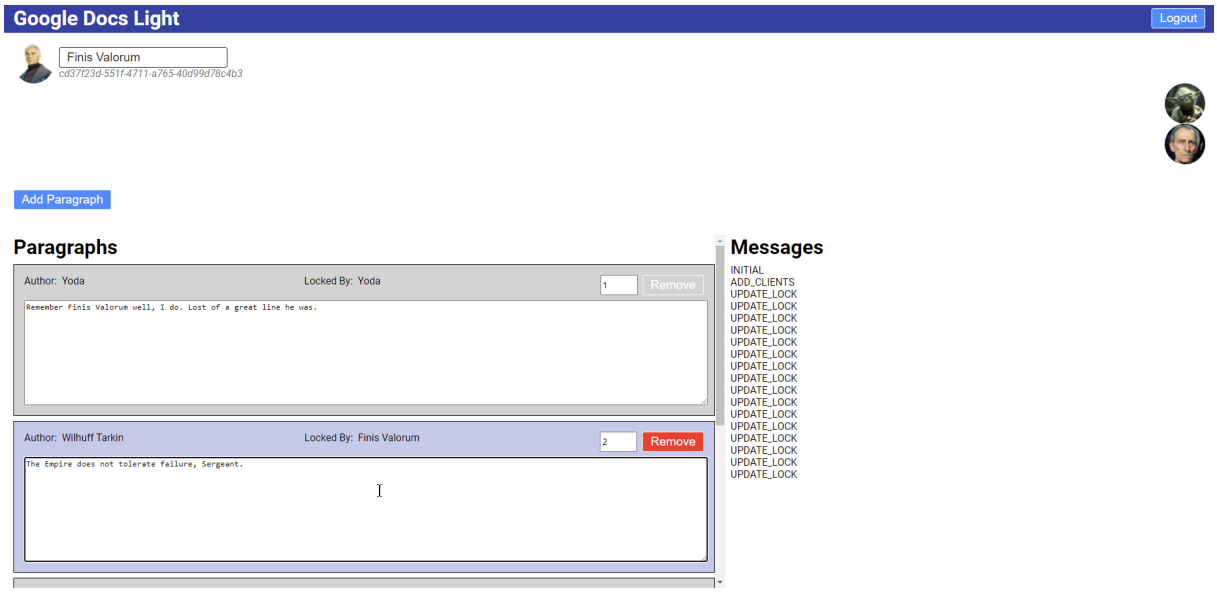


Abbildung 3.1: Team Document User Interface

3.1 Aufbau

Die UI-Elemente sind als React-Komponenten umgesetzt und hierarchisch gegliedert. Einzelne Komponenten nutzen zusätzliche Funktionalität, die in kleine Service Module ausgelagert ist.

Die Anbindung ans Backend ist mit zwei unidirektionalen Kanälen realisiert.

Der State der gesamten Applikation wird vom Redux Store bewirtschaftet.

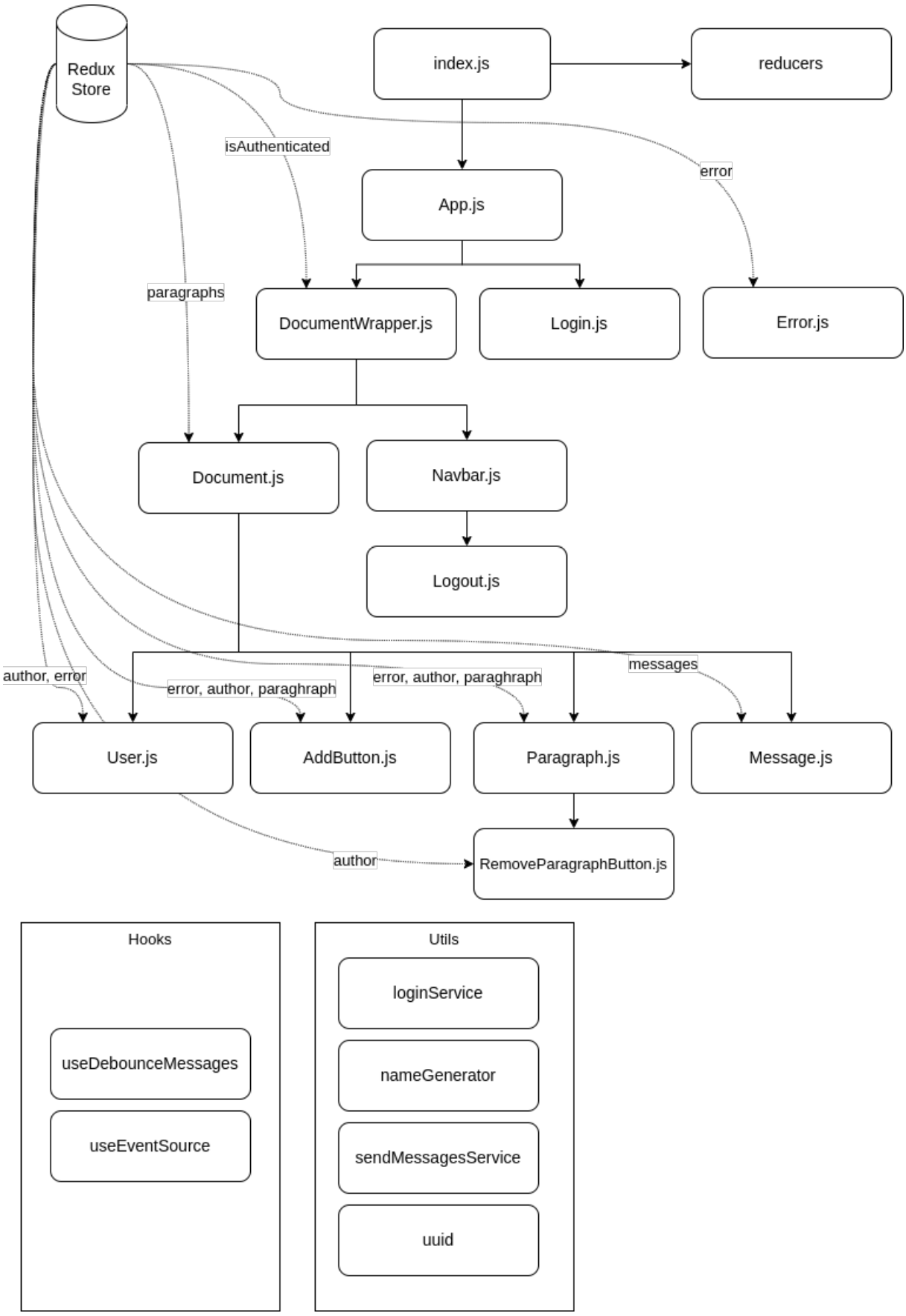


Abbildung 3.2: Komponenten Struktur

index.js

Das Index File ist der Eintrittspunkt für den Browser. Beim Laden der Applikation wird der Redux Store erstellt und initialisiert. Ebenfalls wird im Local Storage des Browsers geprüft, ob bereits ein User registriert ist. Ist dies nicht der Fall, so wird ein zufälliger Benutzer generiert.

App.js

Die App Komponente ist das äusserste Element, welches alle anderen Elemente hält. Wir verwenden einen BrowserRouter um zwischen dem eigentlichen Dokument und der Login-Seite zu navigieren.

DocumentWrapper.js

Wrapper um das Dokument zu schützen. Solange sich ein User noch nicht ordentlich am Backend authentifiziert hat, leitet diese Komponente den Benutzer stetig auf die Login-Page weiter.

Login.js

Login Formular, welches den Login Service verwendet. Das Formular übersetzt die eingegebenen Credentials in einen Basic Auth Header und sendet damit einen GET Request ans Backend. Bei erfolgreicher Authentifizierung wird das User Principal im Local Storage abgelegt.

Error.js

Generische Fehlermeldung, welche als modales PopUp angezeigt wir, im Falle eines fehlgeschlagenen Requests.

Document.js

Document ist der Parent des eigentlichen Dokuments. Hauptsächlich ist sie dafür verantwortlich alle Paragrafen sortiert darzustellen.

Navbar.js

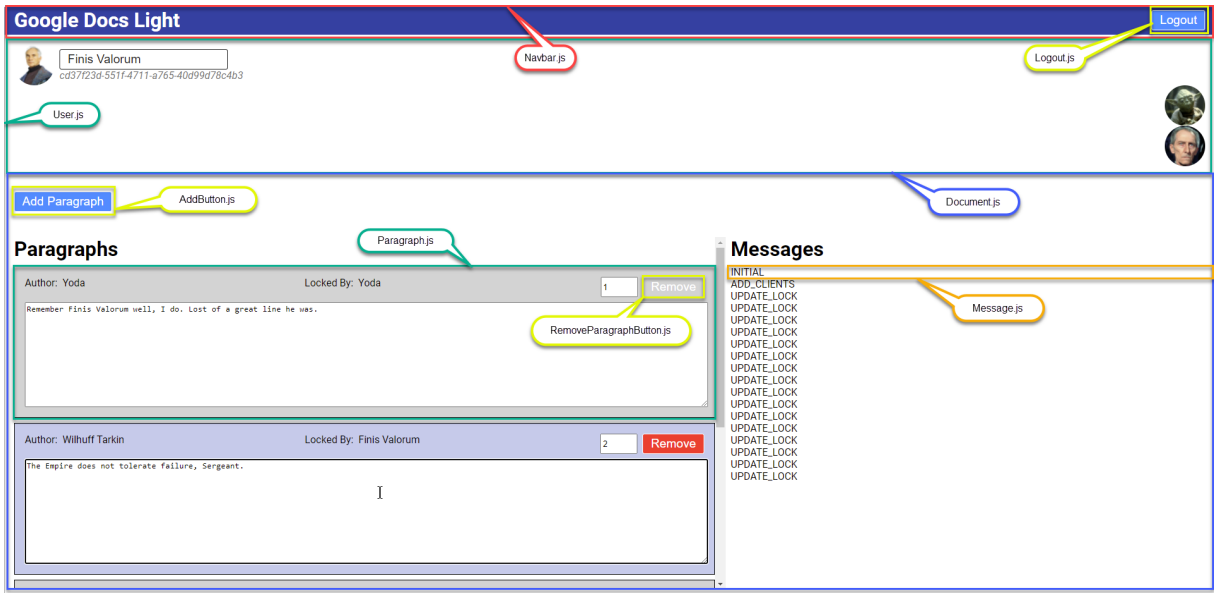
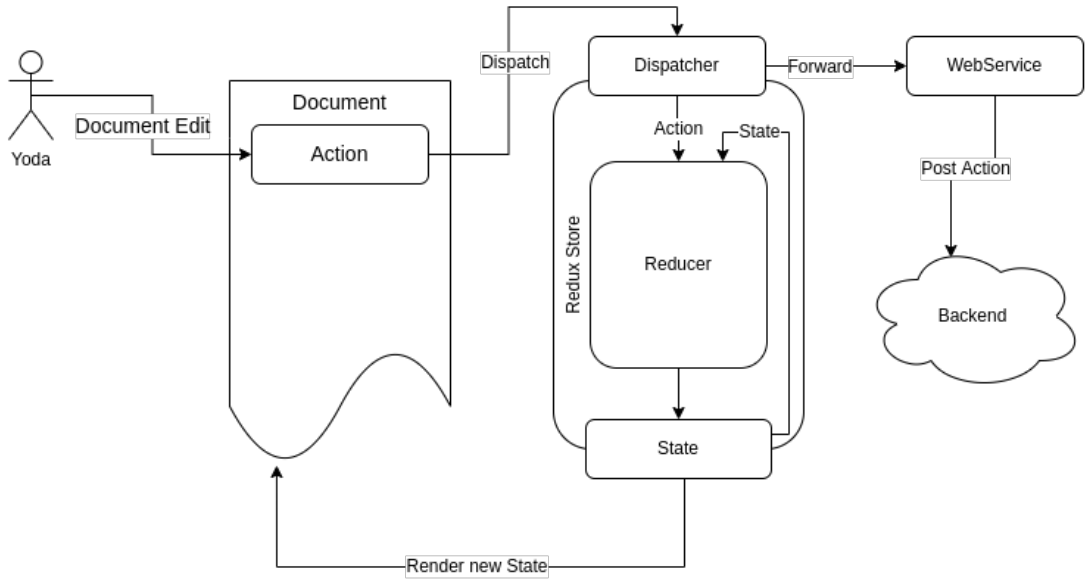


Abbildung 3.3: UI-Components

3.2 Ablaufdiagram



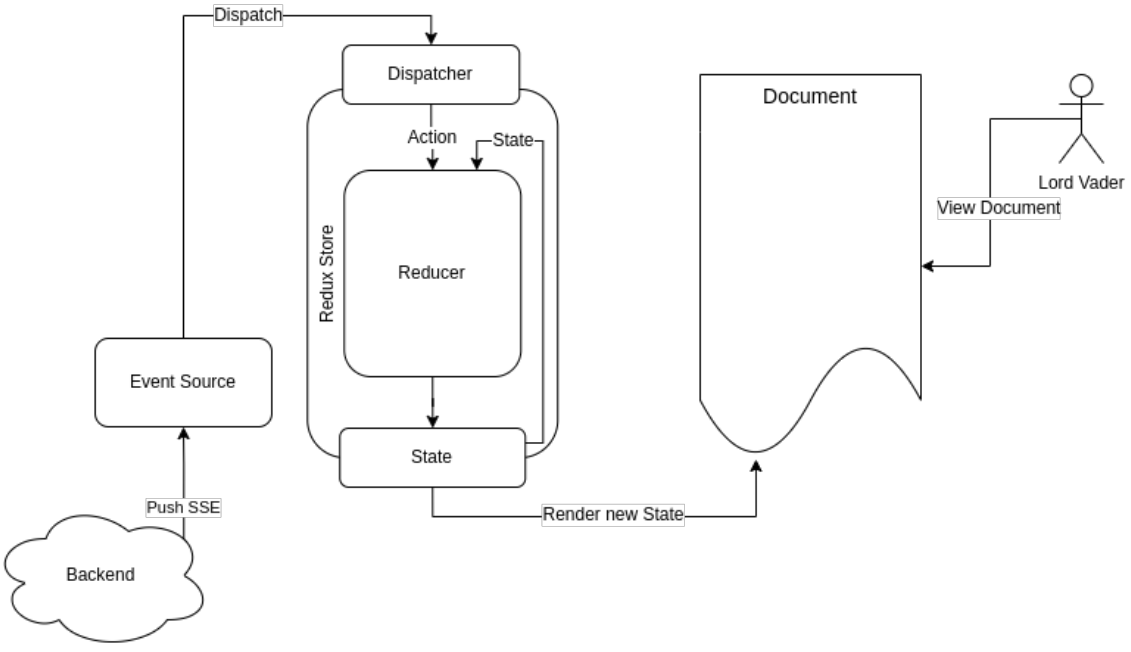


Abbildung 3.4: Datenfluss

3.3 State- und Konfliktmanagment

3.4 Fehler Behandlung

4 Backend

4.1 Aufbau

Der Aufbau der Serverapplikation orientiert sich am Konzept der Onion-Architecture. In Onion Architecture wird die Applikation in Layer aufgeteilt.

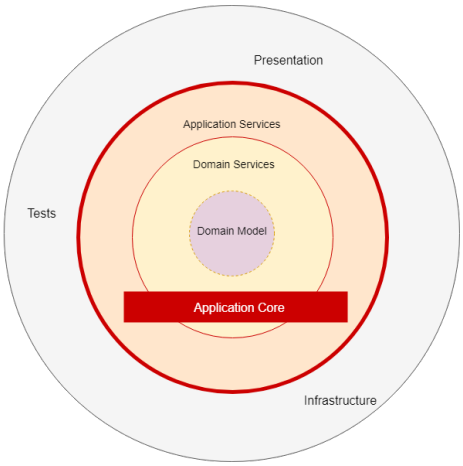


Abbildung 4.1: Onion Architecture

Um zu garantieren, dass keine ungewollten Abhängigkeiten zwischen Layern bestehen, können die Layer in eigene Module verpackt und Abhängigkeiten über Interfaces abstrahiert werden. Dies erhöht jedoch die interne Komplexität der Applikation. Die Umsetzung wird aufgrund der geringen Projektgrösse deshalb nicht in unabhängigen Modulen realisiert, sondern über die Package Struktur gelöst. Bei der Implementation wird dabei konsequent darauf geachtet, die einzelnen Layer so zu halten das diese als eigenständige Module extrahiert werden können. Für die Verwaltung der Komponenten der Serverapplikation wird folgende Packagestruktur definiert:

```
ch.fhnw.woweb.teamdocumentserver
├── config
├── domain
├── persistence
├── service
└── web
```

Abbildung 4.2: Package Struktur Cloud Service

Der Domain Layer wird durch das Package domain abgebildet. Dieses beinhaltet die Domänenobjekte und darf keine Abhängigkeiten auf andere Module oder Frameworks beinhalten. Umgekehrt dürfen alle anderen Layer Abhängigkeiten auf den Domain Layer haben. Die Fachlogik der Applikation wird im Domain Service Layer implementiert. Dieser wird durch das Package service abgebildet. Das Package Service beinhaltet alle Komponenten welche die Domänenobjekte verwalten oder den internen Zustand der Applikation führen. Der Layer Application Services bildet die Brücke zwischen externer Infrastruktur und Domain Services. Er ist in den Packages persistence und web beinhaltenagebildet. Dabei definiert das Package persistence Services welche für Interaktion mit der Datenbank verwendet werden. Das Package web definiert die HTTP-Endpunkte, welche für die Kommunikation mit dem Frontend des Systems verwendet werden. Letztlich beinhaltet das Package config die technische Konfiguration der Applikation.

4.2 API

Die Backendapplikation bietet eine HTTP-Schnittstelle, welche von Frontendapplikationen verwendet werden kann. Die Schnittstelle ermöglich es, sich im System anzumelden, Dokumente zu laden und Änderungen an Dokumenten zu laden und speichern. Um diese Funktionalität zu ermöglichen, bietet die Schnittstelle die drei Bereiche ”Authentication”, ”Document” und ”Message”.

4.2.1 API Authentication

Macht Authentifizierung

Endpunkt:	/api/v1/authentication
Methode	GET
Headers:	Authentication: Basic
Response Code:	200, 401 oder 500
Response Body:	application/json

4.2.2 API Document

Liefert den Initialen Status und Updates

Endpunkt:	api/v1/document
Methode	GET
Headers:	Authentication: Basic
	X-ClientId: text
Response Code:	200, 401 oder 500
Response Body:	text/event-stream

4.2.3 API Message

Verarbeitet Updates

Endpunkt:	/api/v1/message
Methode	POST
Headers:	Authentication: Basic
	Content-Type: application/json
Body:	DocumentCommand
Response Code:	200, 401 oder 500

Stellt den zuletzt gelöschten Paragraphen wieder her.

Endpunkt:	/api/v1/message/restore
Methode	POST
Headers:	Authentication: Basic
Response Code:	200, 401 oder 500

4.3 Komponenten

4.3.1 Domäne

Im Zentrum der Domäne stehen die beiden Klassen DocumentCommand und Document.

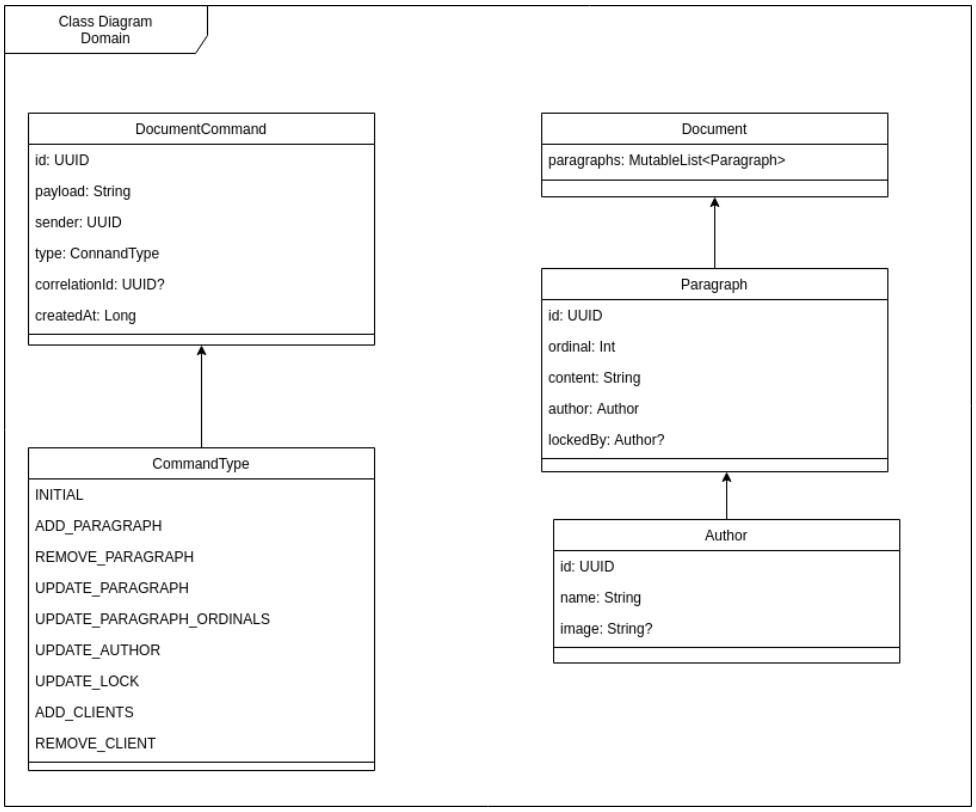


Abbildung 4.3: Klassendiagramm Domain

Document

Eine Instanz der Klasse Document repräsentiert den aktuellen Zustand eines Dokuments. Dieser Zustand wird in der Serverapplikation geführt und verwaltet. Ein Document besteht im Wesentlichen aus einer Liste von Paragraphs. Diese Liste kann mutiert, aber nicht ersetzt werden.

Paragraph

Eine Instanz der Klasse Document repräsentiert einen Abschnitt in einem Dokument. Jeder Paragraph wird durch eine UUID identifiziert. Ein Paragraph definiert weiter ein Attribut Content welches den Textinhalt des Abschnitts beinhaltet und ein Attribut Ordinal, welches die Position des Abschnitts im Dokument darstellt. Weiter ist jeder Paragraph einem Author zugewiesen. Letztlich hat ein Paragraph ein Optionales Attribut lockedBy. Dieses kann entweder leer (NULL) sein oder einen Author beinhalten. Dieses Feld kann von Consumern der API verwendet werden um das bearbeiten eines Paragraphen zu erlauben oder verbieten.

Author

Eine Instanz der Klasse Author repräsentiert einen Benutzer, der an einem Dokument mitarbeitet. Jeder Author wird durch eine UUID identifiziert und muss einen Namen definieren. Weiter besitzt ein Author ein optionales Attribut image. Darin kann die URL zu einem benutzerbild abgespeichert werden

DocumentCommand

Eine Instanz der Klasse DocumentCommand stellt eine Änderung am Zustand einer Document-Instanz dar. DocumentCommands werden als einzige Entität, Persistenzschicht abgelegt werden. Damit DocumentCommands eindeutig identifiziert werden können, beinhaltet jede Instanz ein Attribut vom Type UUID. Diese id wird auch als Identifikator in der MongoDB verwendet.

Die Änderungen welche ein DocumentCommand darstellt, werden über die Attribute payload und type definiert. Die payload hat den Typ String und beinhaltet JSON-Serialisierten des neuen Zustands, er Änderung die vorgenommen werden soll. Das Feld type beinhaltet einen Wert aus der Enum CommandType. Dieser Wert kann in den Domänenservices verwendet werden um die Payload korrekt zu Deserialisieren und die nötigen Änderungen am Document vorzunehmen.

Das Optionale Feld correlationId kann entweder NULL oder eine UUID beinhalten. Eine allfällige UUID zeigt immer auf die Id eines anderen DocumentCommands, welcher mit dem aktuellen Command zusammenhängt. Dadurch wird es möglich die Identifkation der Payload eines Commands zu verwenden, ohne die Payload jedesmal deserialisieren zu müsen.

CommandType

Bei CommandType handelt es sich um eine Enum. Diese Enum beinhaltet alle Arten von DocumentCommands, welche im System bekannt sind. CommandTypes werden als ihr String Wert auf DocumentCommands persistiert.

INITIAL
ADD_PARAGRAPH
REMOVE_PARAGRAPH
UPDATE_PARAGRAPH
UPDATE_PARAGRAPH_ORDINALS
UPDATE_AUTHOR
UPDATE_LOCK
ADD_CLIENTS
REMOVE_CLIENT

4.3.2 Services

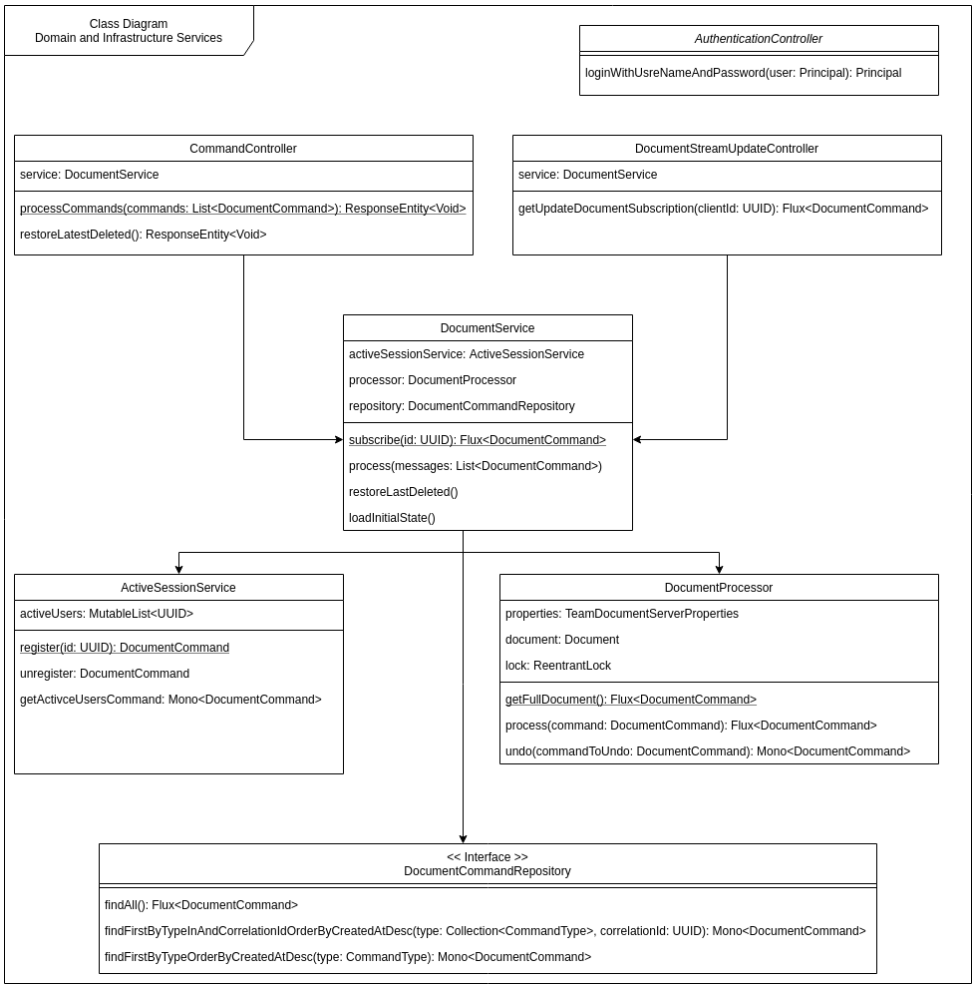


Abbildung 4.4: Klassendiagramm Services

AuthenticationController

Die Klasse AutenticationController implementiert einen RestController. Dieser stelt einen einzelnen GET Endpunkt zur Verfügung, über welchen sich Benutzer mittels Basic Authentication anmelden können.

CommandController

Die Klasse CommandController implementiert einen RestController. Dieser Controller bietet zwei POST Endpunkte zur Verfügung. Über den ersten Endpoint kann eine Liste von DocumentCommands an den Server gesendet werden. Der Endpunkt übergibt diese Liste von Commands an den DocumentService. Diese wenden die Änderungen am Zustand des Dokuments an und leiten die Änderungen an andere Teilnehmer weiter. Über den zweiten Endpunkt kann ein gelöschter Paragraph wiederhergestellt werden. Die entsprechende Fachlogik wird an den DocumentSerivce delegiert.

DocumentStreamUpdateController

Die Klasse DocumentStreamUpdateController implementiert einen RestController. Dieser Controller ermöglicht es, den aktuellen Zustand eines Dokumentes zu laden und Änderungen am Dokument zu abonnieren. Der Endpunkt, welcher dazu zur Verfügung steht erwartet, dass der Custom Header ”X-ClientId” gesetht ist. Dieser muss einen Stringwert beinhalten, welchen den Author der die Daten anfrägt identifiziert. Das Laden des Dokuments und das Erstellen der Abonnierung wird an den DocumentSer-

vice delegiert. Als Rückgabetyp wird ”Flux<DocumentCommand> verwendet. Dadurch ist es möglich den Status des Documents und alle folgenden Änderungen in einem Stream zurückzugeben.

DocumentService

Die Klasse DocumentService ist dafür Verantwortlich erhaltene Anfragen für Dokumente und Änderungen an Dokumenten zu verarbeiten. Sie delegiert die entsprechende Fachlogik Klassen ActiveSessionService, DocumentProcessor und DocumentCommandrepository.

Die Methode process erlaubt es, eine Liste von DocumentCommands zu verarbeiten. Dabei wird die Verarbeitung an den DocumentProcessor delegiert. Dieser wendet Änderungen an und löst Konflikte auf. Die übergebenen Commands und allfällige Commands zur Konfliktlösung werden vom DocumentProcessor zurückgegeben. Der DocumentService übergibt diese an das DocumentCommandRepository zur persistierung. Anschliesend werden die Persistierten Commands über den Sink veröffentlicht.

Die Methode subscribe erlaubt es, einen Stream des aktuellen Zustands des Dokuments und aller künftigen Änderungen an einem Dokument anzufragen. Diese Informationen werden aus drei Quellen zusammengetragen. Zuerst wird der aktuelle Stand des Dokuments beim DocumentProcessor angefragt. Anschliessend wird eine Liste aller aktuellen Authoren des Dokuments aus dem ActiveSessionService geladen. Letztlich wird eine Subscription auf den Sink erstellt. Diese drei Quellen werden zusammengefasst aus der Methode zurückgegeben. Um sicherzustellen, dass die Liste der aktiven Nutzer aktuell ist, wird zudem beim Öffnen der Subscription eine Registrierung beim ActiveSessionService durchgeführt. Diese Registrierung wird beim Schliessen der Subscription wieder entfernt.

DocumentProcessor

Die Klasse DocumentProcessor führt den Zustand des Dokuments, welches mit der Applikation verwaltet wird. Er ist dafür Verantwortlich, Änderungen an diesem Dokument vorzunehmen. Dazu besitzt Sie ein Attribut document vom gleichnamigen Typ. Über die Methode process kann ein einzelner DocumentCommand angewendet werden. Der Processor verarbeitet denm Command anhand des gesetzten CommandTypes. Dabei muss er allfällige Konflikte erkennen und auflösen. Nach der Verarbeitung des Commands werden alle Änderungen und Konfliktlösungen als DocumentCommands zurückgegeben.

ActiveSessionService

Die Klasse ActiveSessionService fürt den Zustand der aktiven Nutzer einer Session. Dazu führt die Klasse eine Liste der Identifikatoren aller aktiven Nutzer einer Session. Der Service bietet Methoden um die aktiven Benutzer auszulesen, einen neuen Benutzer zu registrieren und einen Benutzer zu entfernen.

DocumentCommandRepository

Das Interface DocumentCommandrepository erweitert das Interface ReactiveCrudRepository<DocumentCommand, UUID> von Spring. Es kann damit verwendet werden um Create, Read, Update und Delete Optionen für DocumentCommands in der angebundenen MongoDB auszuführen.

4.3.3 Konfiguration

Spring

Alle Klassen im Package SServiceßind mit der Spring-Boot-Annotation "@Service" versehen. Sie können damit automatisch von Spring-Boot instanziert werden und stehen Sie in Spring Beans zur Verfügung und können über Constructor-Injection verwendet werden.

Alle Klassen im Package "webßind mit der Spring-Boot-Annotation "@RestController" versehen. Sie können damit automatisch von Spring-Boot instanziert werden.

application.yml

Die Datei application.yml beinhaltet die konfigurierbaren Werte der Serverapplikation. Dies beinhaltet die Konfiguration der angebundenen MongoDB, Referenzen zu Umgebungsvariablen mit User Credentials und Logging Konfiguration.

TeamDocumentServerProperties

Die Konfigurationsklasse TeamDocumentServerProperties ist mit der Annotation "@ConfigurationProperties(prefix = teamdocument")" versehen. Sie kann wo nötig in den Serviceklassen verwendet werden, um auf Werte aus dem application.yaml zuzugreifen.

WebConfig

Die Klasse WebConfig beinhaltet die Konfiguration für SpringSecurity.

4.4 Sequenz

4.5 State- und Konfliktmanagment

5 Testing

5.1 Frontend

5.2 Backend

5.3 End to End Test

6 Ausblick

7 Fazit