Fachhochschule Nordwestschweiz
Hochschule für Technik

Google Docs Light

Workshop Web - FS22

20. Mai 2022

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

Inhaltsverzeichnis

1 Technology Stack **1**

1.1 Backend Server 1

1.2 Frontend Clients 1

1.3 Datenbank System 1

2 Component Model **3**

2.1 Lösungsstrategie 3

2.2 Frontend 3

2.3 Backend 3

3 Frontend **5**

3.1 Aufbau 5

3.2 Ablaufdiagram 8

3.3 State- und Konfliktmanagment 8

3.4 Fehler Behandlung 8

1 Technology Stack

Eine zentrale Anforderung an das System ist die konsistente und verzögerungsfreie Darstellung eines Dokuments auf mehreren Klienten. Die Wahl eines geeigneten Kommunikationsprotokolls ist die Grundlage für eine erfolgreiche Lösung.

Wir verwenden HTTP-Event Streams als Grundlage für die Kommunikation zwischen dem Backend Server und den Klienten. Als konkrete Implementation dieser Technologie setzen wir Spring-WebFlux ein. Die weitere Technologieauswahl orientiert sich an diesem Grundsatz Entscheid.

1.1 Backend Server

Spring WebFlux ist integriert in das Spring Boot Ökosystem und benötigt daher eine zugrundeliegende JVM. Sprachen die auf der JVM aufbauen, haben den Vorteil, dass sie System Interoperabel sind.

Anstatt Java setzen wir jedoch auf Kotlin als Backend Sprache. Bis jetzt hat kein Mitglied des Projektteams nennenswerte Erfahrung mit Kotlin und wir möchten diese Gelegenheit nutzen, die Sprache in einem Projekt näher kennenzulernen. Wir erwarten die nachfolgenden Vorteile:

- **Null Safty.**
- Kein Boilerplate Code ohne zusatz Dependencies (Lombok).
- Native Unterstützung des Delegation Patterns.
- Flexibles non-blocking programming mit Coroutines.

1.2 Frontend Clients

Kein Teammitglied hat bis jetzt vertiefte Erfahrung im Bereich der Frontend-Entwicklung. Daher setzen wir auf das an der FHNW vermittelte Framework React, um die Clients zu implementieren. React bietet mit seinem Komponenten-Model eine einfache Abstraktionsmöglichkeit um die Anwendung sauber zu Kapseln. Die Funktionalen JSX Komponenten scheinen leichtgewichtiger im Vergleich zu den HTML-Template-Ansätzen von Angular oder VueJS.

Unser Ziel ist es in diesem Projekt die Kenntnisse in einem Projekt zu vertiefen und die Client-Software möglichst pur funktional zu halten.

1.3 Datenbank System

Um die kollaborativ erstellten Dokumente zu persistieren und zu verwalten setzen wir auf eine No-SQL Lösung. Das notwendige Datenmodel lässt sich elegant als *Document* abbilden. Durch den Einsatz einer No-SQL Lösung kann die Representation der Dokumente über alle Layer der Applikation gleichbleibend beibehalten werden, ohne die Notwendigkeit von ORM.

Konkret wird im Projekt MongoDB als Datenbanksystem verwendet. Wir haben uns für diese Variante aufgrund der bestehenden reaktiven Integration in das Springframework entschieden.

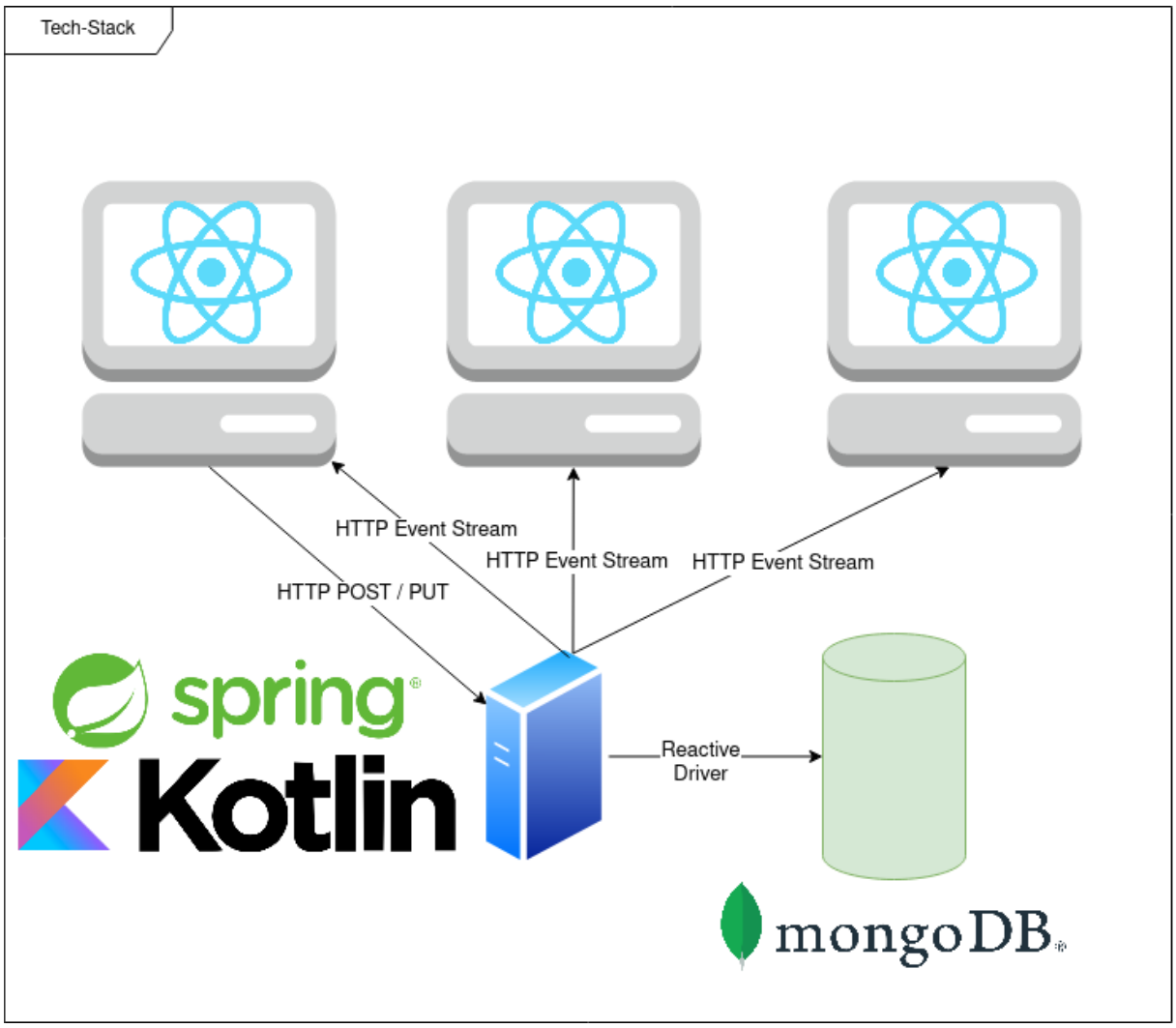


Abbildung 1.1: Technologie Stack

2 Component Model

2.1 Lösungsstrategie

Um die verteilte Dokumentenbearbeitung zu ermöglichen, verwenden wir eine Variante des Command Patterns als Lösungsansatz. Dabei sollen die eingehenden Änderungen am Dokument als einzelne Commands modelliert werden. Die eingehenden Commands werden vom Backend in der ersten Variante sequenziell innerhalb einer Que verarbeitet und die aktualisierte Version des Dokuments an alle Teilnehmer gesendet.

2.2 Frontend

2.3 Backend

Zur Organisation der Backend Sourcen orientieren wir uns an der Onion-Architecture. Die Umsetzung wird aufgrund der geringen Projektgrösse jedoch nicht in unabhängigen Modulen realisiert, sondern über die Package Struktur angedeutet. Bei der Implementation wird dennoch konsequent darauf geachtet, die einzelnen Layer so zu halten das diese als eigenständige Module extrahiert werden können.

Die erwarteten Layer sind:

- Core
  Model der Dokumente
  Business Logik zur Konsistenz garantie
  Command Engine
- Service
  Schnittstelle zwischen Persistenz und API zum Core
- Persistence
  Anbindung an das Datenbanksystem
- API
  Controller für eingehende Dokument Updates
  Controller für die Reaktive-Streams zu den aktiven Clients.

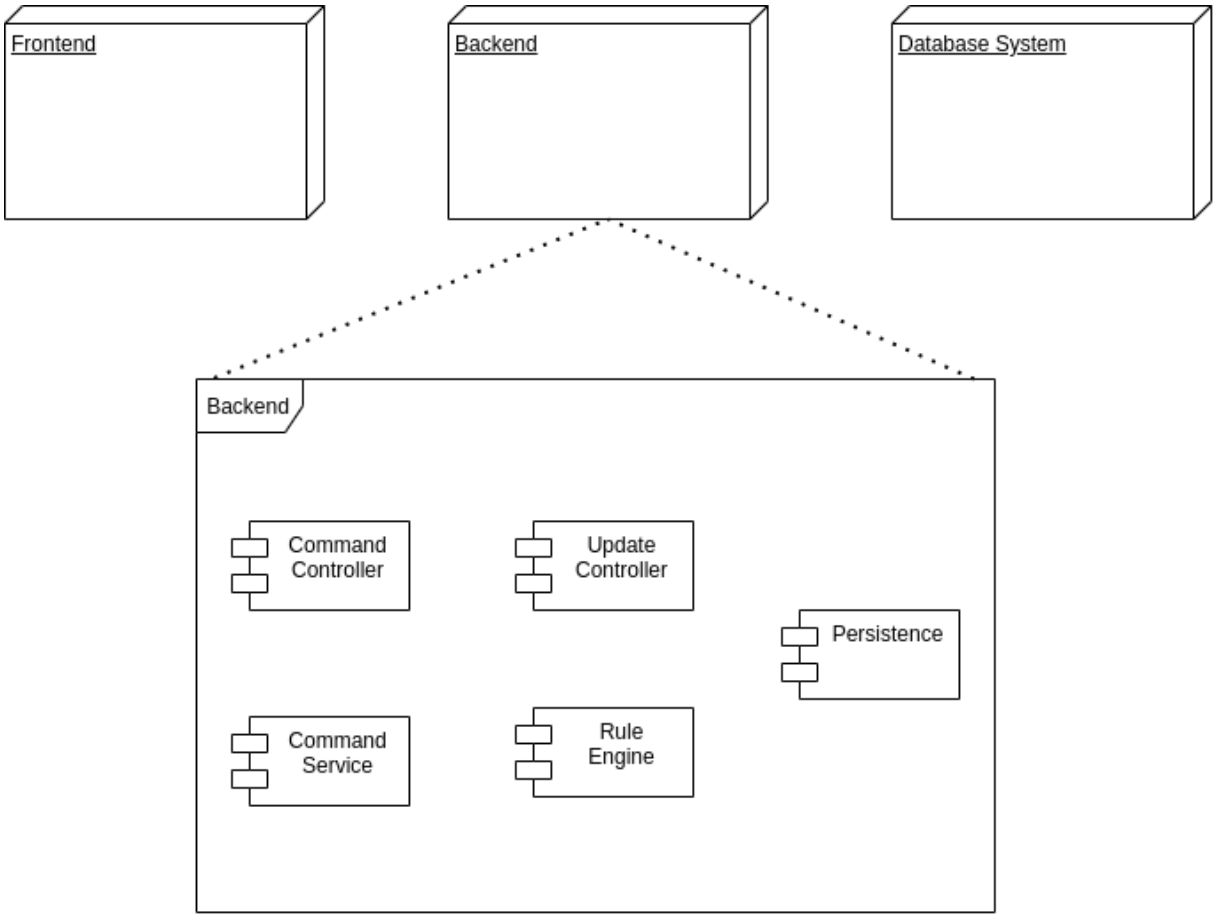


Abbildung 2.1: Simple Backend Component Model

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

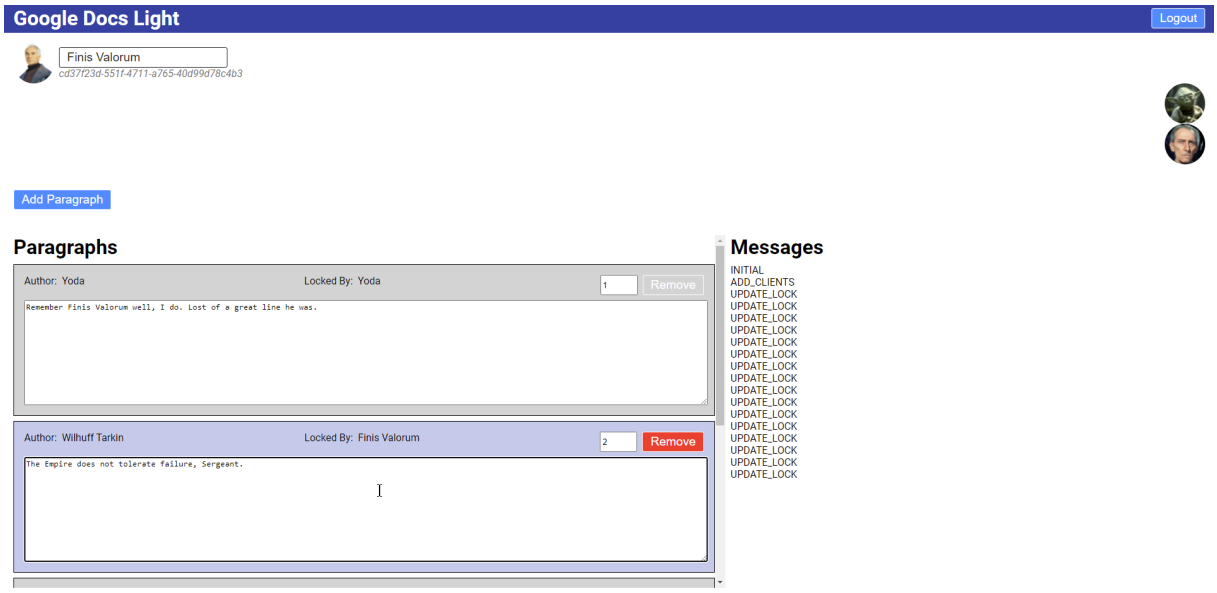


Abbildung 3.1: Team Document User Interface

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.

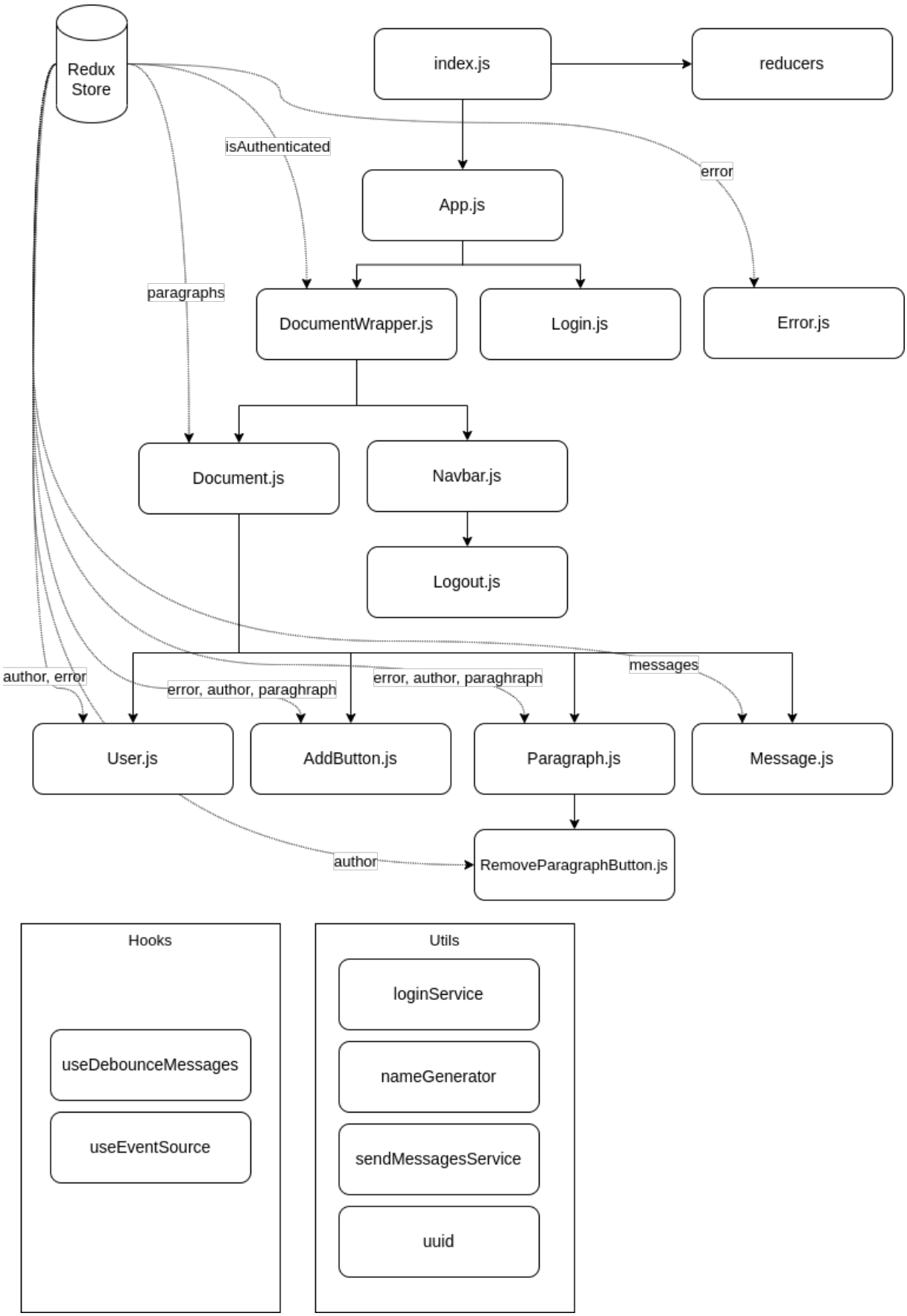


Abbildung 3.2: Komponenten Struktur

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

Navbar.js

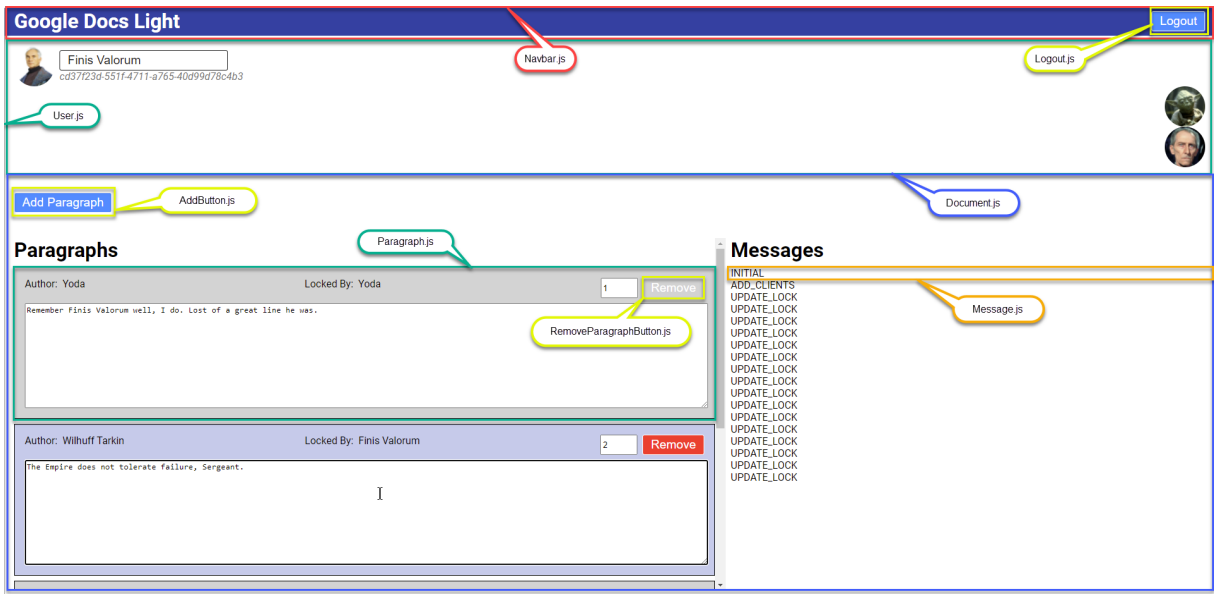
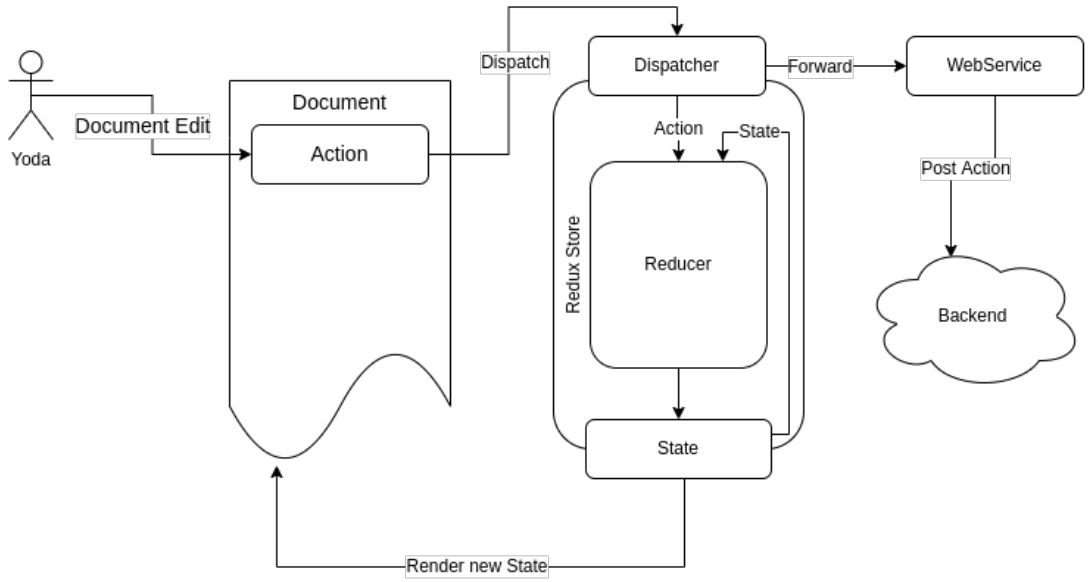


Abbildung 3.3: UI-Components

3.2 Ablaufdiagram



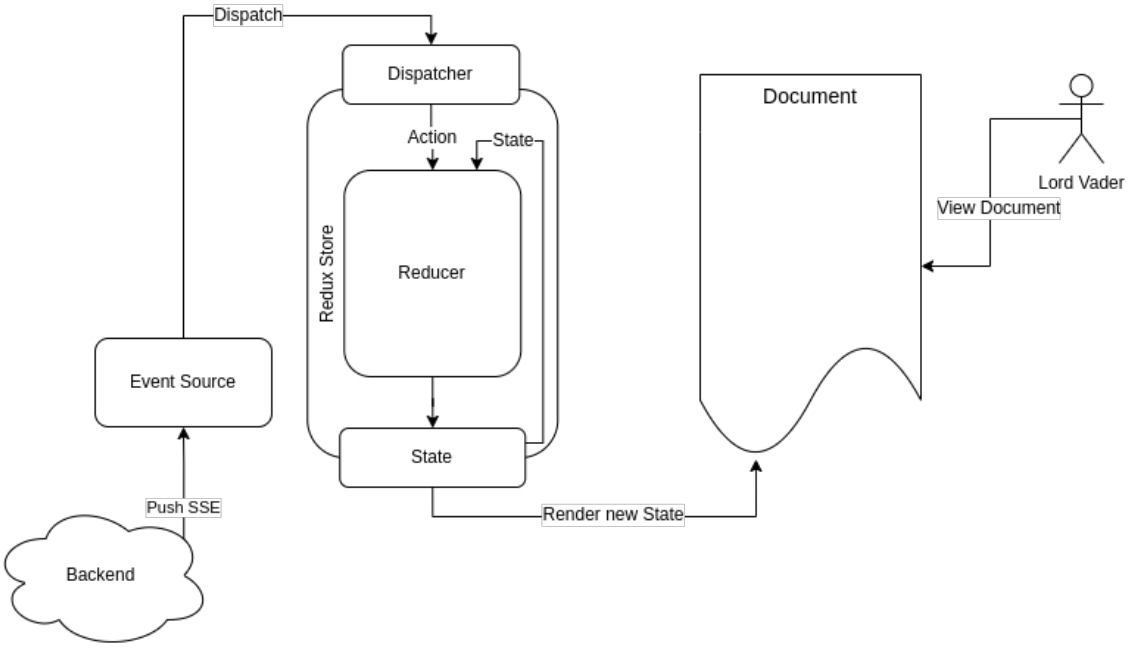


Abbildung 3.4: Datenfluss

3.3 State- und Konfliktmanagment

3.4 Fehler Behandlung