Teamdokument Gruppe 6

Julian Beck, Timon Keller, Joshua Schärer

# Inhaltsverzeichnis

[Organisation 3](#_Toc104240484)

[Informationen 3](#_Toc104240485)

[Arbeitsaufteilung 3](#_Toc104240486)

[Anforderungen 3](#_Toc104240487)

[Funktionale Anforderungen 3](#_Toc104240488)

[Eigene Anforderungen 4](#_Toc104240489)

[Technologien 4](#_Toc104240490)

[Frontend 4](#_Toc104240491)

[React 4](#_Toc104240492)

[**SignalR** 4](#_Toc104240493)

[Backend 4](#_Toc104240494)

[**C#** 4](#_Toc104240495)

[Datenbank 5](#_Toc104240496)

[MongoDB 5](#_Toc104240497)

[Architektur 5](#_Toc104240498)

[Frontend 5](#_Toc104240499)

[Backend 6](#_Toc104240500)

[Persistenz 7](#_Toc104240501)

[Datenbankschema 7](#_Toc104240502)

[Design 7](#_Toc104240503)

[Kommunikation 8](#_Toc104240504)

[Sequenzdiagramme 8](#_Toc104240505)

[Anforderung R01 8](#_Toc104240506)

[Anforderung R02 9](#_Toc104240507)

[Anforderung R03 9](#_Toc104240508)

[Anforderung R04 10](#_Toc104240509)

[Anforderung R05 10](#_Toc104240510)

[Deployment 11](#_Toc104240511)

[Docker 11](#_Toc104240512)

# Organisation

Dieses Kapitel geht kurz auf einzelne organisatorische Aspekte ein.

## Informationen

Source-Code:  
<https://github.com/FHNW-WODSS-FS22/team-document-solariumigel>

URL der lauffähigen Version unserer Lösung:  
<https://woweb-client.azurewebsites.net>

## Arbeitsaufteilung

Die folgende Tabelle 1 schafft eine Übersicht welches Teammitglied was gemacht hat.

|  |  |  |
| --- | --- | --- |
| **Thema** | **Aspekt** | **Mitglied** |
| Frontend | Logik | Julian Beck |
| Timon Keller |
| Joshua Schärer |
| Design | Timon Keller |
| Joshua Schärer |
| Backend | Logik | Julian Beck |
| Joshua Schärer |
| Datenbank | Julian Beck |
| Testing | Frontend | Timon Keller |
| Backend | Julian Beck |
| Live-Schaltung | Azure / Docker | Julian Beck |
| Joshua Schärer |
| Dokumentation |  | Julian Beck |
| Timon Keller |
| Joshua Schärer |

Tabelle 1: Aufteilung der Arbeit im Team.

# Anforderungen

In diesem Kapitel werden kurz alle gestellten Anforderungen an die zu erarbeitende Applikation aufgelistet.

## Funktionale Anforderungen

Die Tabelle 2 listet alle Anforderungen aus der Aufgabenstellung auf.

|  |  |  |
| --- | --- | --- |
| **Nr** | **Name** | **Beschreibung** |
| R01 | Kollaboratives Arbeiten | Mehrere Benutzer können gleichzeitig, also innerhalb einer Arbeitssitzung, gemeinsam Arbeiten. |
| R02 | Arbeitssitzung beitreten und verlassen | Ein Benutzer kann jederzeit einer Arbeitssitzung beitreten oder diese verlassen. |
| R03 | Anzeige aller aktiven Benutzer | Jeder Benutzer sieht, welche anderen Benutzer aktuell an der Arbeitssitzung teilnehmen. |
| R04 | Verhindern von Konflikten | Ein Benutzer kann nicht unabsichtlich die Arbeit eines anderen Benutzers aufhalten, stören, verfälschen oder löschen. |
| R05 | Konsistentes Dokument | Ein Dokument wird innert nützlicher Frist in einen konsistenten, für alle Benutzer gemeinsamen Zustand überführt. |
| R06 | Persistentes Dokument | Ein Dokument kann zu einem späteren Zeitpunkt wieder geöffnet werden, ohne dass der erarbeitete Zustand verloren geht. |

Tabelle 2: Tabelle mit allen funktionalen Anforderungen aus der Aufgabenstellung.

## Eigene Anforderungen

Zusätzlich zu den gegebenen Anforderungen, haben wir zusätzlich weitere eigene Anforderungen spezifiziert.

|  |  |  |
| --- | --- | --- |
| **Nr** | **Name** | **Beschreibung** |
| R07 | Mehrere Dokumente verwalten | Es können mehrere Dokumente erstellt, bearbeitet und gelöscht werden. |

Tabelle 3: Tabelle mit eigenen Anforderungen.

In unseren Augen ist die Anforderung R07, wie in der Tabelle 3 beschrieben, von solch einem verteilten System nicht wegzudenken. Daher haben wir gemeinsam beschlossen, dass wir gerne den zusätzlichen Aufwand auf uns nehmen, um diese eine Anforderung auch entsprechend umzusetzen.

# Technologien

Dieses Kapitel beschreibt unser gewählten Technologie-Stack. Besonders wird auf die Gründe, weshalb wir uns genau für diese Technologien entschieden haben, eingegangen.

## Frontend

ReactWir haben uns für React entschieden, da React heutzutage sehr oft und gern benutzt wird. Zudem kennen wir React bereits aus anderen Modulen im Studium. SignalR lässt sich auch gut mitbenutzten, was unsere Entscheidung noch gefestigt hat.

### **SignalR**

SignalR verwendet den neuen WebSocket-Transport. Sollte dieser jedoch nicht vorhanden sein, greift SignalR auf ältere Versionen zurück. Man hat durch SignalR viele Funktionen, welche man mit reinen Websocket noch zusätzlich implementieren müsste.

## Backend

### **C#**

Da das Geschäft von einem Teammitglied daran ist, das Backend auf C# umzustellen, haben wir uns dazu entschieden dies bereits einmal in kleinerer Version auszuprobieren.

## Datenbank

## MongoDB

Uns hat MongoDB im Team interessiert, da wir es noch nie ausprobiert haben. Zudem können potenzielle Änderungen an den Datenstrukturen einfacher getätigt werden. Zwar ist laut einigen Berichten, “Für die klassischen Anwendungen in der Webentwicklung mit überschaubaren Daten und Datenstrukturen ist MySQL die bessere Lösung”, wir wollten dennoch mal ausprobieren ob und wie gut MongoDB funktioniert.

# Architektur

In diesem Kapitel wird genauer auf die Architektur eingegangen. Speziell werden Entwurfsentscheidungen, als auch interessante Aspekte beschrieben.

## Frontend

Die folgende Abbildung 1 schafft einen ersten Überblick über den Aufbau des Frontend. Das Frontend ist mit ReactJS implementiert.

Diagram

Description automatically generated

Abbildung 1: Überblick über die Struktur des Frontend.

Zu einem haben wir unsere Seiten, durch welche der Benutzer sich navigieren kann. Hierfür werden class components des React frameworks verwendet.  
Weitere existieren auch Components. Components sind functional components und werden innerhalb der Seiten geladen. Das verhilft zu einer besseren Trennung des Codes und verschafft eine gute Struktur.   
Schliesslich gibt es noch einzelne Hilfsklassen. Die Hilfsklassen dienen ebenfalls, um die Logik abzugrenzen und eine einfachere Lesbarkeit zu gewährleisten. Gute Beispiele sind hierfür die API-Klasse und der ConnectionBuilder. Während die API-Klasse alle Logik enthält, um von dem Frontend auf die REST-Schnittstelle auf dem Backend zuzugreifen, erleichtert der ConnectionBuilder den Aufbau einer Verbindung zu dem SignalR-Hub auf dem Backend. Diese Trennung erleichtert das mehrfache Aufrufen der Schnittstellen in verschiedenen views. Die folgende Abbildung 2 zeigt das Vorgehen kurz dar.

Diagram

Description automatically generated Diagram

Description automatically generated

Abbildung 2: Verbindung zum Backend über die API-Klasse. Abbildung 3: Verbindung zum Backend über den ConnectionBuilder.

Als Anmerkung kann das exakte Vorgehen über die API-Klasse in dem Sequenzdiagramm zu der Anforderung R07, wie später in dem Dokument beschrieben, eingesehen werden.

## Backend

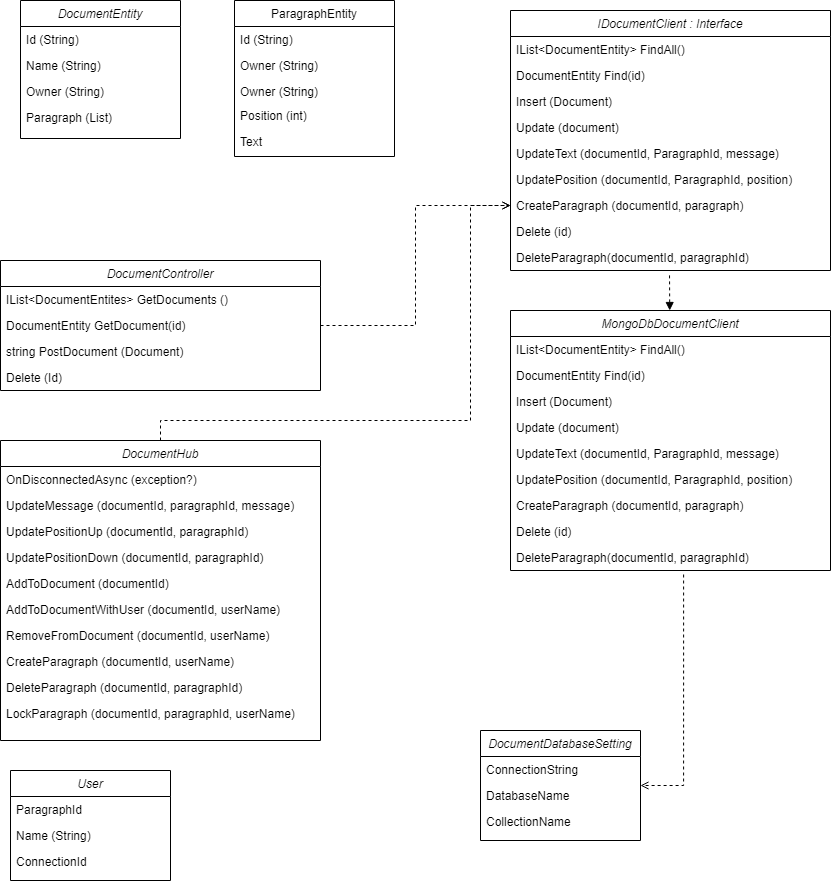
Klassendiagramm

Abbildung 4: Klassendiagram des Backends

Die Hauptkomponente des Backend ist der DocumentHub, dort ist die Logik für das Verhalten implementiert. Der Hub handelt die Websocket Verbindungen zu den verschiedenen Clients.

## Persistenz

Für die Persistenz wird ein Interface verwendet, so können verschiedene Persistenz Systeme verwendet werden. Für unsere Implementation haben wir MongoDB verwendet.

## Datenbankschema

Das Datenbankschema besteht auf zwei Elemente, dem DocumentEntity und dem ParagraphEntity. Ein Dokument kann 0 oder mehrere Paragrafen haben.

Diagram

Description automatically generated

Abbildung : Datenbank Schema

# Design

Wir haben es uns als Aufgabe gestellt, dass wir nebst all den Funktionellen Anforderung auch eine Intuitives Design implementieren möchten. Wir haben uns dazu entschieden die Dokumente und Paragrafen als Cards darzustellen und sie so visuell auch besser zu unterteilen. Zudem haben wir uns für ein Dark-Theme entschieden, da wir das als Team allgemein bevorzugen. Das Hinzufügen von neuen Dokumenten und das Ändern des Usernamens findet via Popup statt. So schaffen wir mehr Überblick auf den Hauptseiten. Allgemein war unser Ziel das UI möglichst simple und mit viel Whitespace zu gestalten, sodass man möglichst schnell die Funktionen findet, die man sucht.

# Kommunikation

Dieses Kapitel soll mit der Hilfe von Sequenzdiagrammen die Kommunikation unter den einzelnen Komponenten übersichtlich aufzeigen.

## Sequenzdiagramme

Hier einige Sequenzdiagramme zu den funktionalen Anforderungen, wie beschrieben in der Tabelle 2.

### Anforderung R01

Timeline

Description automatically generated with medium confidence

Abbildung 6: Sequenzdiagramm zu der Anforderung R01.

### Anforderung R02

Diagram

Description automatically generated

Abbildung 7: Sequenzdiagram zu der Anforderung R02.

### Anforderung R03

Timeline

Description automatically generated with low confidence

Abbildung 8: Sequenzdiagram zu der Anforderung R03.

### Anforderung R04

Diagram

Description automatically generated

Abbildung 9: Sequenzdiagram zu der Anforderung R04.

### Anforderung R05

Diagram

Description automatically generated

Abbildung 10: Sequenzdiagram zu der Anforderung R07.

# Deployment

Dieses Kapitel zeigt kurz, wie wir unsere finale Applikation in die produktive Umgebung eingebunden haben.

## Docker

Für das Deplyoment benützen wir Docker Containers + Azure Cloud.

Wir haben 3 Docker Images, welche wir für unsere Lösung brauchen.

* Client
* Server
* Datenbank

Die Images sind in einem Docker-Compose konfiguriert und können somit überall schnell und vollständig installiert werden. Durch Docker sind wir auch sehr unabhängig welches OS auf dem Host läuft. Dies kann auf einem Windows/Mac/Linux PC laufen oder wie unsere produktive Umgebung auf Azure.

Wir haben uns für Docker entschieden, da es in den Firmen heutzutage die meist genutzte und am weitesten verbreitete Containerlösung ist.