iDocu

Intelligenter Dokumentation Manager

TEKO Schweizerische Fachschule
Thomas Engweiler
Dipl. Techniker HF Informatik
erstellt am 24. Oktober 2021

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Management Summary

1.1 Ausganglage

In vielen Firmen werden für Dokumentationen und Anleitungen verschiedene Programme verwendet. Dabei gibt es selten klare Strukturen oder Vorlagen. Das zu entwickelnde Projekt soll klare Strukturen und Vorlagen vorgeben.

1.2 Zielsetzung

Mit der App sollen Dokumentationen und Anleitungen erstellt und verwaltet werden können.

1.3 Resultat

Es wurde eine Webapplikation entwickelt, welche Cloud-Basiert funktioniert.

In dieser Webapplikation können sich Leute registrieren und anmelden. Das heisst, jeder kann seinen persönlichen Benutzer besitzen. Die angemeldeten Benutzer können zahllose Ordner erstellen und die Freigabe von jedem einzelnen Ordner einstellen.

In jedem Ordner kann ein Unterordner oder Dokument erstellt werden.

Das erstellte Dokument kann beliebig benannt werden.

Das Dokument beinhaltet «Sektionen», welche erstellt werden können. Jede Sektion ist ein separater Abschnitt des eigentlichen Dokuments, Dabei kann der Text formatiert werden und zusätzlich noch Bilder hochgeladen werden.

Die Sektionen, Dokumente, Ordner und Benutzer können verwaltet oder gelöscht werden.

Der Benutzer, welcher den Ordner besitzt, hat automatisch volle Rechte auf dem jeweiligen Ordner.

Die Verwaltung oder Löschung des Ordners benötigt Moderationsrecht auf dem Ordner. Die Verwaltung oder Löschung von Dokumenten und Sektionen benötigt Schreibrechte auf dem Ordner. Das Ansehen des Ordnerinhaltes, sowie Dokumente und Sektionen benötigt Leserechte auf dem Ordner.

Die Berechtigung der jeweiligen Ordner kann der Besitzer oder Moderator des Ordners einstellen. Auch kann der Zugriff für Gäste (nicht angemeldete Personen) gewährt werden.

Die Benutzer können ihren eigenen Benutzer verwalten oder löschen. Nur Benutzer mit Administrationsrecht können andere Benutzer verwalten oder löschen. Ausserdem kann der Administrator noch manuell die Speicherkapazität einzelner Benutzer verändern.

Jeder kostenlose Benutzer hat standardmässig 10MB Speicherkapazität. Sobald diese Speicherkapazität erreicht ist, kann kein weiterer Inhalt mehr hinzugefügt werden. Was allerdings weiterhin geht, ist das Erstellen von Ordnern oder Dokumenten. Nur das Erstellen von Sektionen wird unterbunden. Es gibt keine weiteren Einschränkungen von kostenlosen Benutzern.

Jeder Anwender muss seinen Benutzer per E-Mail verifizieren.

Die Webapplikation ist via https://www.idocu.app/ erreichbar.

2 Die Person hinter dem Projekt

Thomas Engweiler

Walterswilerstrasse 2, 5745 Safenwil
07.02.1991

2.1 Beruflicher Werdegang

Lehre als Automatiker

Gyger Elektormotoren GmbH, Oensingen
08.2008 – 07.2012

Automatiker – Fachrichtung Elektromaschinenbauer

Kernkraftwerk Gösgen-Däniken AG, 4658 Däniken
08.2012 – 12.2012

Automatiker – Fachrichtung Elektromaschinenbauer

Kaufmann AG Elektromotorenbau, 5013 Niedergösgen
01.2013 – 12.2013

Automatiker – Service Techniker

Schibli-Carnelli AG, 4512 Bellach
01.2014 – 08.2015

Automatiker – Service Techniker

TSCHUDIN AG, 2540 Grenchen
09.2015 – 09.2017

Informatiker – Junior Devoloper

4future management ag, 6002 Luzern
10.2017 – 11.2020

Informatiker – System Engineer

Rohde & Schwarz SwissQual AG, 4528 Zuchwil
03.2020 - Heute

2.2 Qualifikationsprofil

Menschen führen (Prozess 1)

Als Projektleiter von «www.pantheongame.de» führe ich ein Team von 4 Personen. Bei diesem Projekt werden monatliche Newsletter und andere unregelmässige Inhalte von Englisch ins Deutsche übersetzt und auf der Webseite zur Verfügung gestellt. Meine Funktion dabei ist es, die Aufgaben zu koordinieren und Qualitätskontrollen durchzuführen.

Entscheidungen fällen (Prozess 2)

Im meinem beruflichen Umfeld als System Engineer bin ich immer wieder gefordert Server zu definieren. (Standort, Leistung, Menge). Dabei muss ich jeweils mit den Entwicklern zusammen schauen, wie viel Leistung die Applikation benötigt und weitere Einzelheiten berücksichtigen.

Projekte planen und leiten (Prozess 3)

Das Projekt «www.pantheongame.de» habe ich eigenständig geplant. Die Webseite erhält den Inhalt von der Originalseite des Spieleherstellers, auch gingen wir eine Kooperation mit dem Spieleherstellers ein, um frühzeitig Informationen zu erhalten. Bei Projektstart habe ich zusätzlich freiwillige Helfer gesucht für diverse Funktionen welche ich als Projektleiter definiert habe. Das Projekt besteht mittlerweile aus 2 Übersetzer, 1 Korrekturleser, 2 Administratoren.

Sich sprachlich verständigen (Prozess 4)

In meiner Funktion als System Engineer muss ich Server weltweit bereitstellen (Cloud oder Dedicated). Dabei wird meistens Englisch gesprochen / geschrieben. Dabei muss ich Kundenwünsche erkennen und umsetzen. Ausserdem muss ich für jeden bereitgestellten Server einen Bericht ausfüllen und ablegen.

Wirkungsvoll präsentieren und kommunizieren (Prozess 5)

Beim Kauf der neuen Hardware für die internen Zwecke in der Firma, musste ich mehrere Präsentationen halten, um das Management von der neuen Hardware zu begeistern. Dabei musste ich mit einfachen Mitteln, das Management überzeugen können, wieso gerade die gewählte Hardware unsere Bedürfnisse abdeckt.

Unternehmensprozesse verstehen und mitgestalten (Prozess 6)

In meiner aktuellen Anstellung als System Engineer bei Rohde & Schwarz SwissQual AG bin ich zuständig für das Installieren der kundenspezifischen Software. Das Installieren der Software beinhaltet das Aufsetzen und Einstellen des Betriebssystems, dass Installieren sämtlicher Softwaredrittanbieter und das Installieren der firmeneigenen Software. Dabei überprüfe ich die Unternehmensprozesse für das Installieren der Software und passe diese bei Bedarf an.

Geschäftsziele erreichen (Prozess 7)

Als System Engineer ist es meine Aufgabe neue Hardware bereitzustellen. Meine Aufgabe dabei ist, die kosteneffizienteste Hardware zu organisieren. Dabei schaue ich die Mindestanforderung und die mögliche Expansion an.

Umfeld berücksichtigen (Prozess 8)

Als System Engineer ist es meine Aufgabe neue Hardware bereitzustellen. Dabei gibt es verschiedene Faktoren, welche ich berücksichtigen muss. Welches Tastaturlayout wünscht sich der Angestellte, wie viele Monitore benötigt der Mitarbeiter bei der Arbeit? Was sind die Mindestanforderungen für die Hardware? Welche Berechtigungen benötigt der Angestellte? Welche Software benötigt der Mitarbeiter, um effizient zu arbeiten? All diese Fragen muss ich beantworten und entsprechend bereitstellen.

Probleme analysieren und lösen (Prozess 9)

Gibt es eine Störung im internen Netzwerk oder Server bin ich zuständig für die Fehleranalyse und Fehlerbehebung. Dabei gehe ich meistens strickt nach unseren internen Protokollen (ITSM) vor.

Sich persönlich weiter entwickeln (Prozess 10)

Beim Bereitstellen eines Servers schreibe ich jeweils ein Bericht, was ich wieso erledigt habe. Dieses Vorgehen ist eine gute Selbstreflektion, welche ich immer verwende, um mich und die Prozesse zu verbessern.

Business Anforderungen analysieren und bestimmen (Prozess 12)

In Unternehmen bin ich Zuständig für das Evaluieren und Integrieren neuer Software und Hardware.

Datenschutz und Datensicherheit gewährleisten (Prozess 14)

Im Unternehmen arbeiten wir mit Daten von hoher Sicherheitsstufe. Dabei bin ich zuständig die von den Kunden erwarteten Sicherheitskonzepten umsetzten zu können und Gegebenenfalls noch anzupassen, falls diese nicht unseren Standarden entspricht.

Applikationen entwickeln, Programme erstellen und testen (Prozess 16)

Das Projekt «www.pantheongame.de» besitzt ein eigenes Design (Templates und Stylesheets). Ausserdem habe ich diverse kleine Plugins entwickelt, um den Inhalt der Originalwebseite besser zu implementieren. Zusätzlich ist die Webseite mit dem Nachrichten Dienst «Discord» verknüpft. Die Webseite basiert auf dem CMS von WoltLab Suites.

System- und Netzwerkarchitektur bestimmen (Prozess 17)

Als das Homeoffice begonnen hat, war meine Aufgabe unsere Internen Netzwerke auch via VPN erreichbar zu machen und trotzdem die Sicherheit zu gewährleisten. Alle unsere Netzwerke sind logisch voneinander getrennt. Nur einzelne Arbeitsplätze besitzen Zugriff auf mehrere Netzwerke. Mit mehreren VPN Regeln konnte ich dies schliesslich umsetzen.

Konzepte und Dienste entwickeln (Prozess 18)

In meiner Funktion als System Engineer musste ich die Alarmierung bei Systemstörung von kundenspezifischen Systemen entwickeln.

Konzepte und Dienste umsetzen (Prozess 19)

Die Alarmierung bei Systemstörung von kundenspezifischen Systemen, welche ich entwickelt habe, musste ich auch umsetzten. Die Umsetzung habe ich mit der Monitoring Software «Zabbix» umgesetzt. Da unsere Kundensysteme weltweit stehen, war die Implementierung der einzelnen Systeme sehr komplex.

3 Projektidee

Das Tool «Intelligenter Dokumentation Manager» kurz «iDocu», soll sich auf Dokumentationen und Anleitungen jeglicher Form fokussieren.

Mit vielen Programmen, werden bereits Dokumentationen oder Anleitungen geschrieben, doch keines bietet eine saubere Struktur und intelligente Verwaltung an. Aus dem Gedanken entstand das Projekt «iDocu».

Im Rahmen der Diplomarbeit werden die Grundlagen des Programmes entwickelt und möglichst umgesetzt.

4 Projektauftrag

Projekttitel	Idocu (Intelligent Documentation Manager)

Projektleiter	Thomas Engweiler

Projektdaten			
Start	13 September 2021	Ende	25 Oktober 2021

Projektbeschreibung	
Ausgangslage / Projektbegründung	In vielen Firmen werden für Dokumentationen und Anleitungen verschiedene Programme verwendet. Dabei gibt es selten klare Strukturen oder Vorlagen. Das zu entwickelnde Projekt soll klare Strukturen und Vorlagen vorgeben.
Sinn und Zweck / Nutzen	Die App soll nach Fertigstellung diversen Firmen zur Verfügung gestellt werden, um die internen Dokumentationen zu erstellen und verwalten.
Projektrichtziel	Mit der App sollen Dokumentationen und Anleitungen erstellt und verwaltet werden können.

Projektplanung	
Projektphasen / Meilensteine	• Projektplanung • Backend Entwicklung • Frontend Entwicklung • Deployment • Abschluss

Tabelle 1: Projektauftrag

Endergebnisse	Erfolgskriterien
1) Funktionalität	1)
a) Kunden können sich auf der Webseite registrieren und anmelden.	a) Ein Benutzer mit E-Mail-Adresse und Passwort ist angelegt.
b) Jeder Kunde erhält mindestens einen Ordner 1) Der Kunde kann mindestens einen öffentlichen Ordner erstellen 2) Der Kunde kann mindestens einen privaten Ordner erstellen	b) Ein Ordner ist erstellt. 1) Der Ordner ist öffentlich zugänglich 2) Der Ordner ist öffentlich nicht zugänglich
c) Der Speicherplatz des Kunden kann eingegrenzt werden	c) Der Speicherplatz ist auf 10 MB eingegrenzt
d) Jeder Kunde kann seine eigenen Ordner verwalten, inklusive Berechtigung	d) Eine Berechtigung für einen Ordner ist erstellt.
e) Ob ein Kunde einen privaten Ordner für andere Freigeben kann, soll steuerbar sein.	e) Die Freigabeberechtigung für einen User wurde angepasst.
f) Bilder müssen hochgeladen und eingefügt werden können	f) 3 Bilder wurden erfolgreich hochgeladen und dem Dokument zu geordnet.
g) Text muss formatierbar sein	g) Der Text wurde fett und unterstrichen formatiert.
h) Einem Ordner können Dokumente hinzugefügt werden	h) Einem Ordner wurden 3 Dokumente hinzugefügt
i) Kunde können gelöscht werden	i) Ein Kunde wurde gelöscht
j) Ordner können gelöscht werden	j) Ein Ordner wurde gelöscht
k) Dokumente können gelöscht werden	k) Ein Dokument wurde gelöscht
2) Benutzeroberfläche	
a) Die Benutzeroberfläche soll sich am Windows Explorer orientieren	a) Die Struktur der Ordnerübersicht unterscheidet sich zu maximal 30%
b) Die App soll mehrsprachig entwickelt werden	b) Die App wird mit Deutscher und Englischer Sprache angeboten
3) Technische Aspekte	3)
a) Das Frontend soll mit React.js entwickelt werden	a) Das Frontend ist mit React.js entwickelt
b) Das Backend soll mit Node.js entwickelt werden	b) Das Backend ist mit Node.js entwickelt
c) Die Datenverwaltung soll mit PostgreSQL umgesetzt werden	c) Die Datenverwaltung ist mit PostgreSQL umgesetzt.

Tabelle 2: Endergebnisse / Erfolgskriterien

5 Projektplanung

5.1 Kommunikationsplan

Wie wird mit dem Auftraggeber oder Diplomlehrer wann kommuniziert?

Statusberichte

Inhalt / Ziele	Festhalten der Fortschritte, Tendenzen, Problematiken
Teilnehmer / Verteiler	Diplomlehrer, Projektteam
Termin	Wöchentlich jeweils am Freitag
Art	E-Mail, PowerPoint-Präsentation

Tabelle 3: Statusbericht Definition

5.2 Projektstrukturplan / Phasenplan

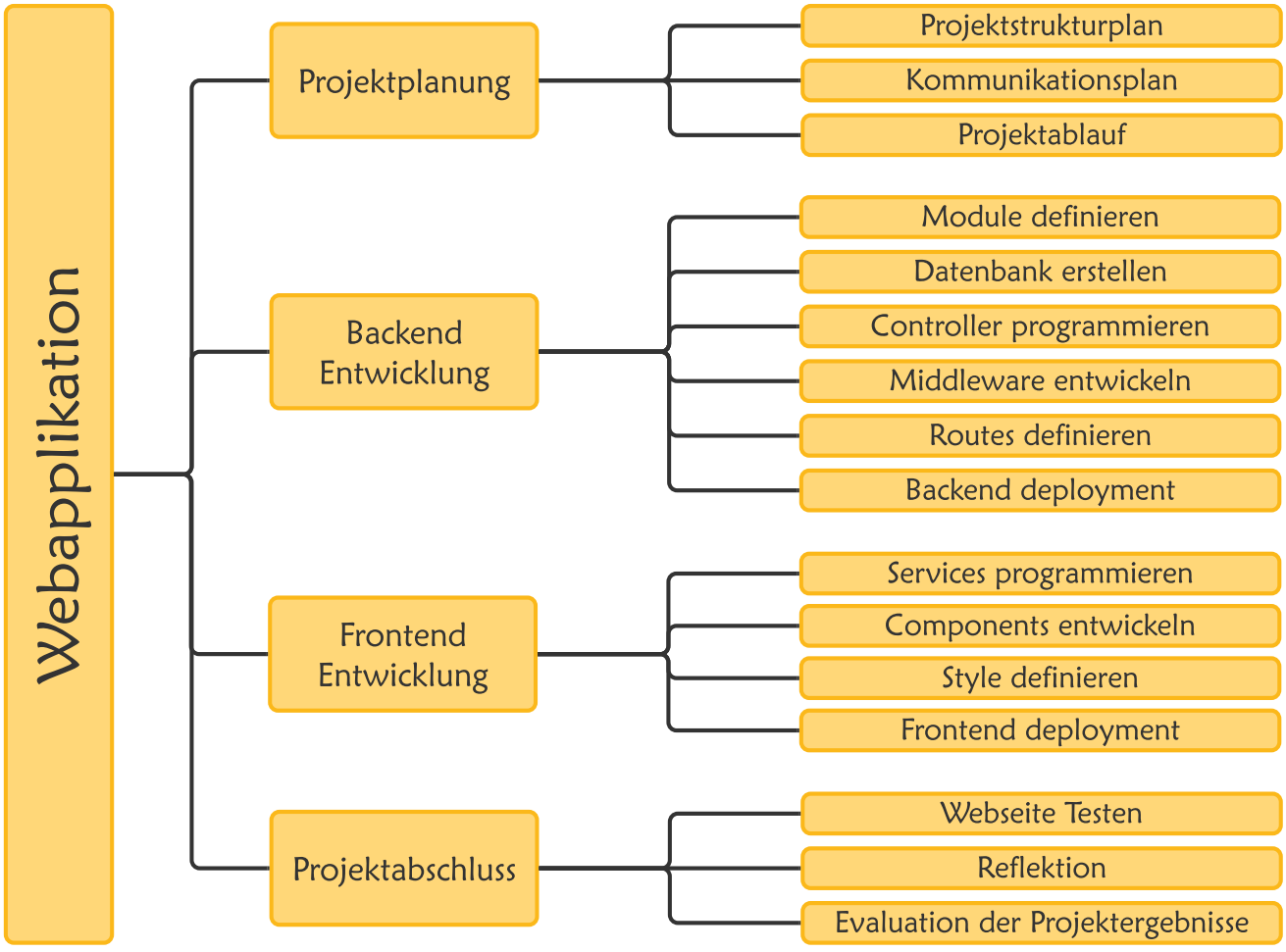
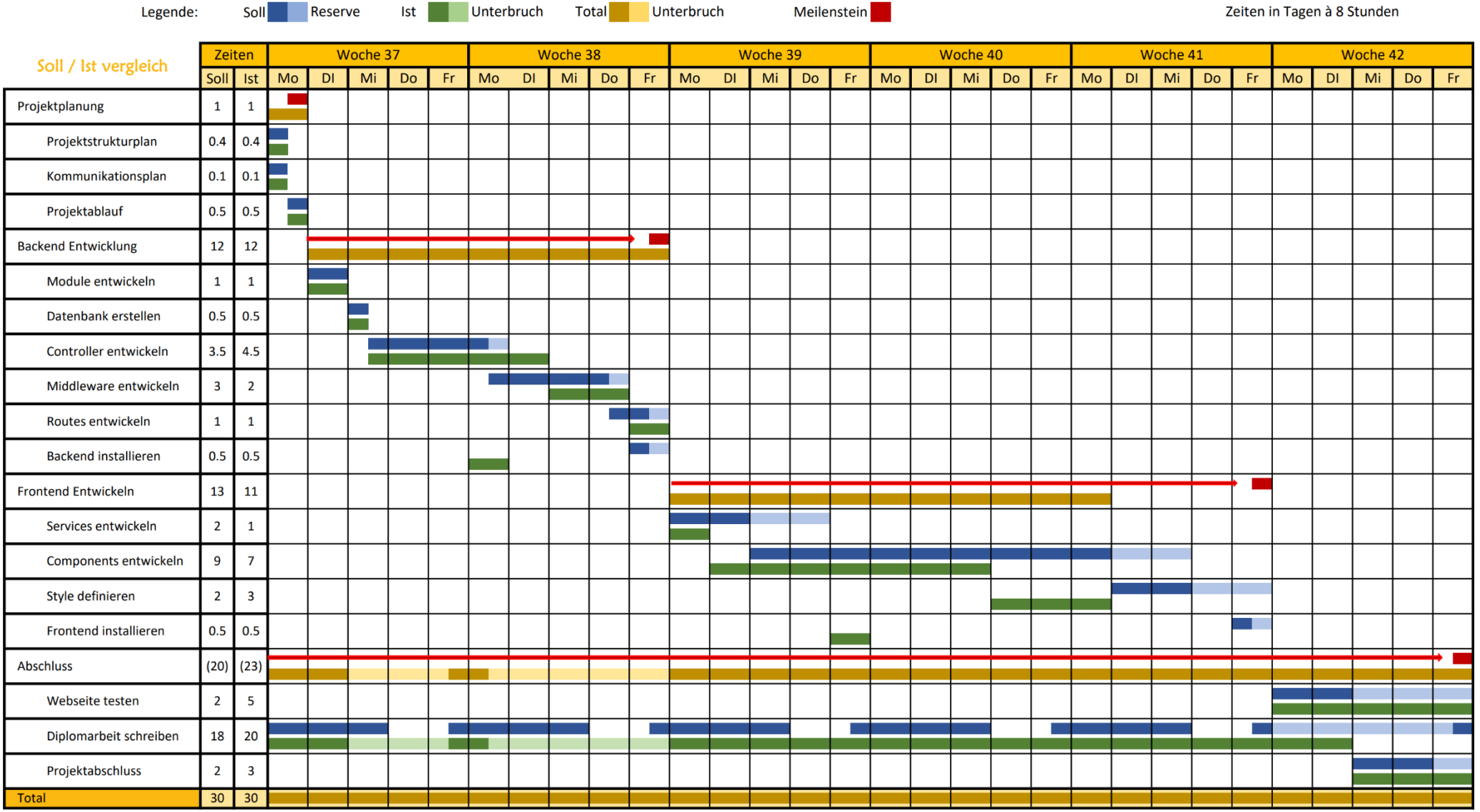


Abbildung 1: Projektstruktur

5.3 Soll / Ist Vergleich



Soll / Ist vergleich	Zeiten		Woche 37					Woche 38					Woche 39					Woche 40					Woche 41					Woche 42				
	Soll	Ist	Mo	Dl	Mi	Do	Fr	Mo	Dl	Mi	Do	Fr	Mo	Dl	Mi	Do	Fr	Mo	Dl	Mi	Do	Fr	Mo	Dl	Mi	Do	Fr	Mo	Dl	Mi	Do	Fr
Projektplanung	1	1																													
Projektstrukturplan	0.4	0.4																													
Kommunikationsplan	0.1	0.1																													
Projektablauf	0.5	0.5																													
Backend Entwicklung	12	12																													
Module entwickeln	1	1																													
Datenbank erstellen	0.5	0.5																													
Controller entwickeln	3.5	4.5																													
Middleware entwickeln	3	2																													
Routes entwickeln	1	1																													
Backend installieren	0.5	0.5																													
Frontend Entwickeln	13	11																													
Services entwickeln	2	1																													
Components entwickeln	9	7																													
Style definieren	2	3																													
Frontend installieren	0.5	0.5																													
Abschluss	(20)	(23)																													
Webseite testen	2	5																													
Diplomarbeit schreiben	18	20																													
Projektabschluss	2	3																													
Total	30	30																													

Tabelle 4: Soll / Ist Vergleich

30 Tage à 8 Stunden entspricht 240 Stunden.

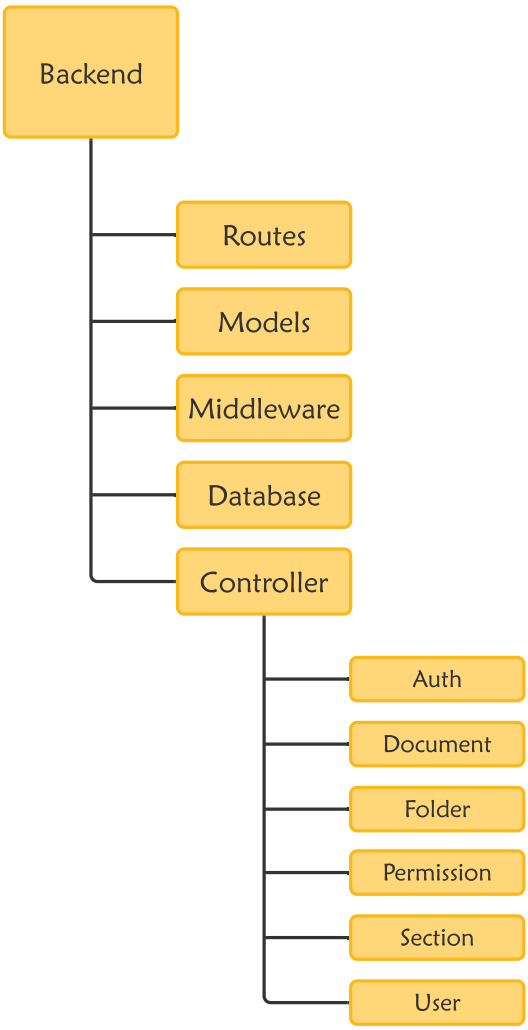
6 Projektrealisierung

Bei der Projektrealisierung wird Schritt für Schritt erklärt wie das Programm entwickelt und umgesetzt wurde.

6.1 Backend Entwicklung

Das Backend wird eine Node.JS Express Rest API, welche mit Javascript programmiert wird.

6.1.1 Struktur und Gliederung



Die Struktur vom Backend richtet sich stark am "best practice".

Im Hauptordner "root" befindet sich lediglich index.js und build.js, welche beide Startpunkte sind. Index.js führt die eigentliche Applikation aus. Die build.js migriert die Datenbank, diese Datei dient lediglich für die Initialisierung der Datenbank auf einem neuen Server.

Im Ordner "Router" werden die Dokumente für das Routing der API abgelegt.

Im Ordner "Models" werden die Dokumente für das Definieren von Datensätzen abgelegt.

Alle Module, welche dazwischen sequenziell ausgeführt werden, gehören in den Ordner "Middleware".

Im Ordner "Database" findet sich das Database Objekt, welches die Initialisierung und Authentifizierung der Datenbank übernimmt.

Der "Controller" Ordner enthält mehrere Unterordner, um die Struktur besser zu gewährleisten, da die Controller auf mehrere Dokumente aufgeteilt sind.

Dabei hat jeder Controller sein eigenen Ordner, das eigentliche Objekt wird als index.js abgespeichert und die Module zusätzlich geladen.

Abbildung 2: Gliederung des Backend

6.1.2 Module Definieren

Um die Module entwickeln zu können verwenden wir das externe Modul "Sequelize". Sequelize unterstützt uns mit der Kommunikation mit der Datenbank und das Definieren von Modulen. Mit einfachen Schritten, kann so eine Datenbank erstellt und genutzt werden.

Jedes Modul wird in ein separates Dokument abgespeichert.

In der Index.js Datei im Ordner Models werden alle Models zu einem Objekt hinzugefügt und die Relationen anhand des Entitäten Relationen Modell gemacht. Das erstellte Objekt wird "db" bezeichnet und in allen Controller und Middlewares verwendet.

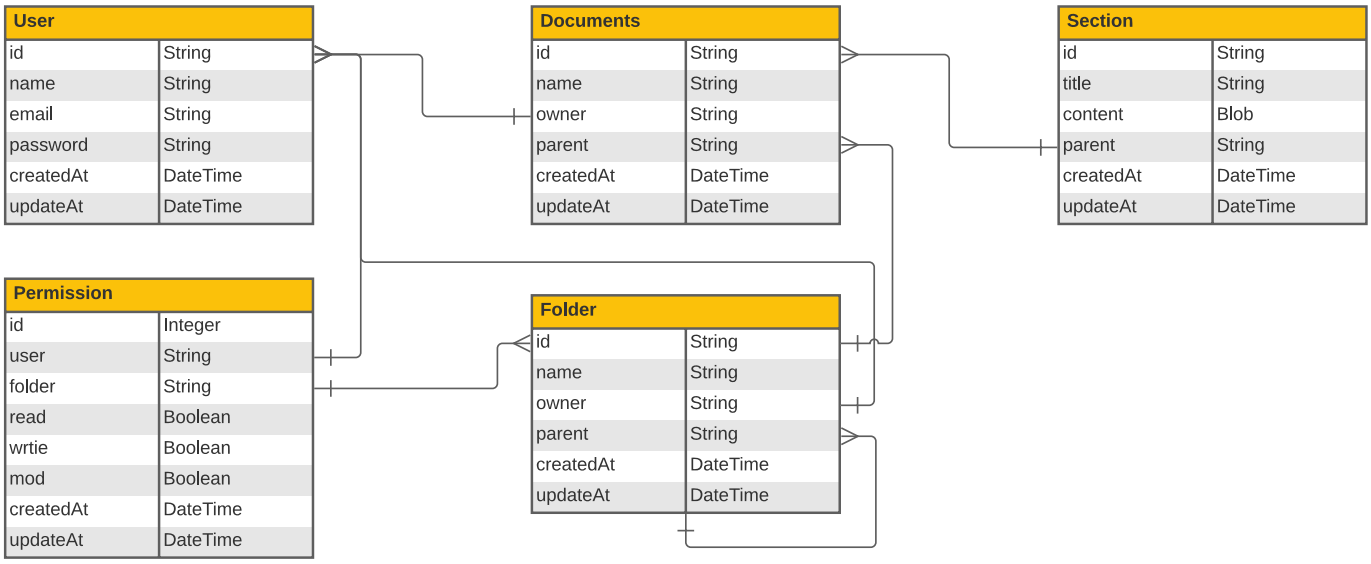


Abbildung 3: Entitäten Relationen Modell

UUID

Bei Modellen, welche die Kunden Identifikation Nummern sehen, sollen sogenannten UUID verwendet werden. Diese verschleiern die eigentliche Anzahl der bereits vorhandenen Datensätze.

Parent

Modelle welche Vater-Objekte haben, soll das Vater-Objekt jeweils als "parent" dargestellt werden, um keine Verwechslung mit der eigentlichen ID zu ermöglichen.

Owner

Modelle welchen Usern zugewiesen werden, soll der User jeweils als "owner" dargestellt werden. Dies soll klar kennzeichnen was das für eine UserID ist.

Model: Documents

Das "Documents" Model dient zum Abspeichern der Dokument Informationen. Jedes Dokument muss einem Ordner zugewiesen werden. Der Ordner wird dem Schlüssel "parent" zugewiesen. Der Wert "owner" speichert, wer das Dokument erstellt hat. Es hat keinen Einfluss auf die Berechtigung.

Modul: Folder

Das "Folder" Model dient zum Abspeichern von Ordner Informationen. Das Model kann eine Referenz auf einen anderen Ordner besitzen, weshalb das Model ein Schlüssel "parent" besitzt. Dabei gibt es keine Einschränkungen wie oft Kind ein Vater-Objekt besitzt. Der Wert "owner" speichert, wer den Ordner erstellt hat. Dabei hat der "owner" des Ordners immer vollen Zugriff darauf.

Permission

Das "Permission" Model wird für das Verwalten von den Berechtigungen zwischen User und Ordner genutzt. Jeder Ordner, der erstellt wird, hat standardmässig keine Berechtigungstabelle.

6.1.3 Datenbank erstellen

Für das Erstellen der Datenbank wird eine laufende Instanz des definierten Datenbanksystems verlangt.

Beim Initialisieren der Datenbank wird automatisch überprüft, ob sich die definierten Module von der Datenbank unterscheiden. Mit dem Befehl Sync wird die Datenbank synchronisiert. Wobei bereits erstellte Elemente nicht angepasst werden, ausser dies wird per Parameter übergeben.

```
db.sync({force: true}).then(() => {
    models.role.create({
        id: 1,
        name: "user"
    }).then(() => {
        Builder.count();
    });

    models.role.create({
        id: 2,
        name: "admin"
    }).then(() => {
        Builder.count();
    });

    models.role.create({
        id: 3,
        name: "tester"
    }).then(() => {
        Builder.count();
    });


});
```

Abbildung 4: Auszug aus build.js

In meinem Skript «build.js» welche für das Erstellen der Datenbank genutzt wird, wird eine Neuerstellung der gesamten Datenbank erzwungen. Ist die eigentliche Synchronisation abgeschlossen werden die Benutzer-Rollen erstellt.

Da alle asynchron zueinander erstellt werden und das Skript erst weiter machen soll, wenn alle abgeschlossen sind, gibt es noch das «Builder» Objekt. Das Builder Objekt zählt die erstellten Gruppen.

Sobald alle Gruppen erstellt sind, wird noch der Administratoren Account erstellt.

Sobald alles abgeschlossen ist, wird «database builded» ausgegeben und das Skript kann beendet werden.

6.1.4 Controller Erstellen

Jede Funktion von den Controllern wird in ein gleichnamiges Dokument gespeichert. Dies dient zur besseren Übersicht. Der eigentliche Controller wird dann als «index.js» im dazugehörigen Ordner gespeichert.

Die Berechtigungslogik wird mit der Middleware «hasPermission» abgewickelt, weshalb eine Überprüfung der Berechtigung in dem Controller nicht notwendig ist.

```
const remove = (req, res) => {

    if (req.userid !== req.params.userid) {
        authJwt.isAdmin(req, res, () => {
            userDelete(req,res);
        });
    } else {
        userDelete(req,res);
    };
};
```

Abbildung 5: Auszug aus controller user remove

Nur bei speziellen Berechtigungen steuert der Controller die Berechtigungen selbst. Eine Ausnahme davon ist, dass «user» Objekt. Der Benutzer kann nur sein eigenes «user» Objekt anpassen. Ausgenommen der Benutzer gehört der Administratoren Gruppe «admin» An. Administratoren können alle Benutzer anpassen.

In der Abbildung 5 wird diese Ausnahme mit «authJwt.isAdmin» überprüft.

```
const get = (req, res) => {
    const query = {};
    query.where = {id:req.params.folderid};

    db.folder.findOne(query).then((folder) => {
        return res.success(folder);
    }).catch((e) => {
        return res.error(e.message,e);
    });
};

export default get;
```

Abbildung 6: Auszug aus controller folder get

Das externe Modul sequelize bietet einfache vorgefertigte Abfragen. In der Abbildung 6 wird eine Select Abfrage durchgeführt, welche ein Ergebnis zurückgibt. Ausserdem ist die Abfrage mit «where» auf der primären ID ergänzt. Ist die Suche erfolgreich wird ein Ordner ausgegeben, ist die Suche nicht erfolgreich wird ein Fehler ausgegeben. Eine erfolgreiche Suche, kann auch kein Resultat erhalten. Fehlerhaft wäre sie nur, falls einer der angegebenen Parameter ungültig wäre.

```
const getList = (req, res) => {
    const query = {};
    query.where = {
        [db.Op.and]: [],
    };

    if (req.query.folderid) {
        query.where[db.Op.and].push({parent: req.query.folderid});
    };

    if (req.query.parent) {
        query.where[db.Op.and].push({parent: {[db.Op.is]: null}});
    }

    if (req.userid !== 0 && req.query.own) {
        query.where[db.Op.and].push({owner: req.userid});
    };

    if (req.query.name) {
        query.where[db.Op.and].push({name: {[db.Op.iLike]: `%${req.query.name}%`}});
    };

    if (req.query.limit) {
        query.limit = req.query.limit;
    };

    if (req.query.offset) {
        query.offset = req.query.offset;
    };

    if (req.query.orderName && req.query.orderASC) {
        query.order = [
            [req.query.orderName, (req.query.orderASC) ? "ASC" : "DESC"]
        ];
    };

    query.include = {
        model: db.permission,
        where: {
            [db.Op.and]: [
                {read: true},
                {userid: (req.userid !== 0) ? req.userid : null}
            ],
        },
        attributes: ["read"],
        required: (req.userid !== 0) ? false : true,
    };

    db.folder.findAll(query).then((folders) => {
        const folderList = [];
        folders.forEach((folder) => {
            if (folder.owner === req.userid || folder.permissions.find((perm) => perm.read == true)) {
                folderList.push(folder);
            }
        })

        return res.success(folderList);
    }).catch((e) => {
        return res.error(e.message,e);
    });
};

export default getList;
```

Abbildung 7: Auszug aus folder getList

In der Abbildung 7 ist ein komplexes Query welche mehrere Parameter enthalten kann. Je nach dem Ob die Suche eingegrenzt werden muss oder nicht. Ausserdem wird mit `query.include` ein inner Join Query aufgebaut. In diesem include wird die Permission zusätzlich geladen.

Nach der Abfrage wird überprüft ober der Benutzer Leseberechtigung für den jeweiligen Ordner hat. Leseberechtigt ist er entweder als Besitzer «owner» des Ordners oder mit der entsprechenden «read» Berechtigung.

6.1.5 Middleware Entwickeln

Alle Funktionen, welche für die Überprüfung der Routes dient, werden im Middleware Ordner abgelegt.

Die Middlewares werden als Parameter der Router Funktionen übergeben und vor dem Controller ausgeführt.

ValidateUUID

Das Backend verwendet uuidv4.

Damit die Abfragen nicht scheitern, werden die eingegebenen Parameter abgefangen und überprüft.

Dabei werden alle Parameter von der URL und von gesendeten Objekt Überprüft welche folgenden Namen besitzen: «userid», «folderid», «documentid», «sectionid» und «owner»

authJWT

Diese Middleware überprüft das verwendete JWT-Token und lädt entsprechend das User-Objekt.

```
router.use('/documents',[authJwt.verifyToken],document);
router.use('/folders',[authJwt.verifyToken], folder);
router.use('/users',[authJwt.verifyToken], user);
router.use('/permissions',[authJwt.verifyToken], permission);
```

Abbildung 8: Auszug aus routes index

Jedes Mal, wenn eine Anfrage an die in Abbildung 8 abgebildeten Routen gesendet wird, wird das JWT-Token verifiziert.

```
const verifyToken = (req, res, next) => {
    let token = req.headers["x-access-token"];

    if (!token) {
        req.userid = 0; //Gast
        next();
        return;
    }

    jwt.verify(token, config.secret, (err, decoded) => {
        if (err) {
            req.userid = 0; //Gast
        } else {
            req.userid = decoded.id;
        }

        next();
        return;
    });
};
```

Abbildung 9: Auszug aus middleware authJwt

Das JWT Token wird als Header Paramater «x-access-token» erwartet. Wird kein gültiges Token gefunden wird der User auf 0 gesetzt. Somit weiss das fortlaufende Programm, dass es sich um einen Gast handelt. Wurde ein gültiges Token gefunden, wird die Benutzer-ID als User gesetzt. Die Benutzer-ID ist wie alle anderen ID eine UUIDv4.

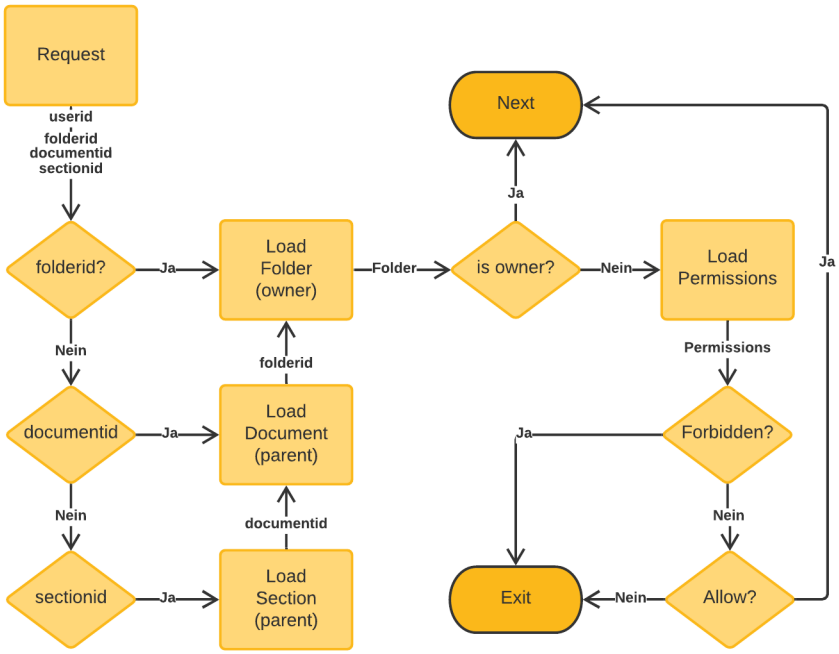
verifySignUp

Diese Middleware wird nur für die Registration verwendet, sie überprüft die Einzigartigkeit eines Benutzers.

hasPermission

Damit die Berechtigung nicht jeweils im Controller überprüft werden muss, wird dies per Middleware gelöst. Die Funktionen sind so aufgebaut, dass anhand der Erwarteten Parameter die Berechtigung geladen wird und entsprechend geprüft.

Abbildung 10: Berechtigungsberechnung



Response Extension

Diese Middleware erweitert das Response Objekt von Express Modul «Routes» um weitere Funktionen. Diese Funktionen dienen dazu, dass alle Ausgaben nach dem gleichen Aufbau ausgegeben werden.

Logger

Dieses Modul dient zum Loggen von den Zugriffen auf der Webseite.

6.1.6 Routes definieren

Hier werden die Routen aufgelistet mit den jeweiligen Methoden und erwarteten Parametern

Registration

POST /api/v1/signup/

Header

```
{}
```

Body

```
{
	“username”: String,
	“email”: String,
	“password”: String,
}
```

Wird zur Registration im System verwendet.

Anmeldung

POST /api/v1/signin/

Header

```
{}
```

Body

```
{
	“username”: String,
	“password”: String,
}
```

Wird zum Anmelden im System verwendet.

Sitzungserneuerung

GET /api/v1/refresh/

Header

```
{
	“x-access-token”: JWT
}
```

Body

```
{}
```

Wird zur Erneuerung der Sitzung verwendet.

Verifizierung

GET /api/v1/refresh/

Header

```
{
	“x-access-token”: JWT
}
```

Body

```
{}
```

Wird zur E-Mail-Adressen Verifizierung verwendet.

Benutzer

GET /api/v1/users/

Header

```
{
	“x-access-token”: JWT
}
```

Body

```
{}
```

Gibt die Liste aller Benutzer aus. [benötigt Administrationsrechte]

POST /api/v1/users/

Header

```
{
	“x-access-token”: JWT
}
```

Body

```
{
	“username”: String,
	“email”: UUIDv4,
	“password”: UUIDv4,
	“active”: Boolean,
	“quota”: Int,
	“lang”: String,
}
```

Erstellt einen neuen Ordner. [benötigt Administrationsrechte]

POST /api/v1/users/:userid

Header

```
{
	“x-access-token”: JWT
}
```

Body

```
{
	“name”: String,
	“parent”: UUIDv4,
	“owner”: UUIDv4
}
```

Anhand des Parameters “:userid” werden die übergebenen Werte in das Benutzer Objekt gespeichert.

GET /api/v1/users/:userid

Header

```
{
	“x-access-token”: JWT
}
```

Body

```
{}
```

Anhand des Parameters «:userid» wird das entsprechende Benutzer Objekt zurückgeliefert.

DELETE /api/v1/users/:userid

Header

```
{
	“x-access-token”: JWT
}
```

Body

```
{}
```

Löscht den mit dem Parameter «:userid» definierten Benutzer.

Ordner

GET /api/v1/folders/

Header

```
{
	“x-access-token”: JWT
}
```

Body

```
{}
```

Gibt die Liste von Ordner aus.

POST /api/v1/folders/

Header

```
{
	“x-access-token”: JWT
}
```

Body

```
{
	“name”: String,
	“parent”: UUIDv4,
	“owner”: UUIDv4

}
```

Erstellt einen neuen Ordner.

POST /api/v1/folders/:folderid

Header

```
{
	“x-access-token”: JWT
}
```

Body

```
{
	“name”: String,
	“parent”: UUIDv4,
	“owner”: UUIDv4
}
```

Anhand des Parameters “:folderid” werden die übergebenen Werte in das Ordner Objekt gespeichert.

GET /api/v1/folders/:folderid

Header

```
{
	“x-access-token”: JWT
}
```

Body

```
{}
```

Anhand des Parameters «:folderid» wird das entsprechende Ordner Objekt zurückgeliefert.

DELETE /api/v1/folders/:folderid

Header

```
{
	“x-access-token”: JWT
}
```

Body

```
{}
```

Löscht den mit dem Parameter «:folderid» definierten Ordner.

Dokumente

GET /api/v1/documents/

Header

```
{
	“x-access-token”: JWT
}
```

Body

```
{}
```

Gibt alle Dokumente aus, welche anhand der Rechte des Benutzers ausgegeben werden können.

GET /api/v1/folders/:folderid/documents

Header

```
{
	“x-access-token”: JWT
}
```

Body

```
{}
```

Gibt alle Dokumente des mit dem Parameter «:folderid» definierten Ordner aus.

GET /api/v1/documents/:documentid

Header

```
{
	“x-access-token”: JWT
}
```

Body

```
{}
```

Anhand des Parameters «:documentid» wird das entsprechende Dokumenten Objekt zurückgeliefert.

POST /api/v1/folders/:folderid/documents/

Header

```
{
	“x-access-token”: JWT
}
```

Body

```
{
	“name”: String,
	“owner”: UUIDv4
}
```

Erstellt ein Dokument anhand der übergebenen Werte auf dem mit dem Parameter «:folderid» definierten Ordner.

POST /api/v1/documents/:documentid

Header

```
{
	“x-access-token”: JWT
}
```

Body

```
{
	“name”: String,
	“parent”: UUIDv4,
	“owner”: UUIDv4
}
```

Anhand des Parameters “:documentid” werden die übergebenen Werte in das Dokumenten Objekt gespeichert.

DELETE /api/v1/documents/:documentid

Header

```
{
	“x-access-token”: JWT
}
```

Body

```
{}
```

Löscht das mit dem Parameter «:documentid» definierten Dokument.

Sektionen

GET /api/v1/documents/:documentid/sections

Header

```
{
	“x-access-token”: JWT
}
```

Body

```
{}
```

Gibt alle Sektionen des mit dem Parameter «:documentid» definierten Dokuments aus.

GET /api/v1/documents/:documentid/sections/:sectionid

Header

```
{
	“x-access-token”: JWT
}
```

Body

```
{}
```

Mit den definierten Parametern «:documentid, :sectionid» wird das entsprechende Sektionen Objekt ausgegeben.

POST /api/v1/documents/:documentid/sections/

Header

```
{
	“x-access-token”: JWT
}
```

Body

```
{
	“title”: String,
	“content”: String,
	“owner”: UUIDv4
}
```

Erstellt eine Sektion anhand der übergebenen Werte auf dem mit dem Parameter «:documentid» definierten Dokument.

POST /api/v1/documents/:documentid/sections/:sectionid

Header

```
{
	“x-access-token”: JWT
}
```

Body

```
{
	“title”: String,
	“content”: String,
	“owner”: UUIDv4
}
```

Anhand der Parameters “:documentid, :sectionid” werden die übergebenen Werte in das Sektionen Objekt gespeichert.

DELETE /api/v1/documents/:documentid/sections/:sectionid

Header

```
{
	“x-access-token”: JWT
}
```

Body

```
{}
```

Löscht die mit den Parametern «:documentid, :sectionid» definierte Sektion.

Berechtigung

GET /api/v1/folders/:folderid/permissions

Header

```
{
	“x-access-token”: JWT
}
```

Body

```
{}
```

Gibt alle Berechtigungen des mit dem Parameter «:folderid» definierten Ordner aus.

GET /api/v1/folders/:folderid/permissions/:permissionid

Header

```
{
	“x-access-token”: JWT
}
```

Body

```
{}
```

Mit den definierten Parametern «:folderid, :permissionid» wird das entsprechende Berechtigungsobjekt ausgegeben.

POST /api/v1/folders/:folderid/permissions/

Header

```
{
	“x-access-token”: JWT
}
```

Body

```
{
	“read”: Boolean,
	“write”: Boolean,
	“mod”: Boolean,
	“userid”: UUIDv4?
}
```

Erstellt eine Berechtigung anhand der übergebenen Werte auf dem mit dem Parameter «:folderid» definierten Ordner.

POST /api/v1/folders/:folderid/permissions/:permissionid

Header

```
{
	“x-access-token”: JWT
}
```

Body

```
{
	“read”: Boolean,
	“write”: Boolean,
	“mod”: Boolean,
	“userid”: UUIDv4?
}
```

Anhand der Parameters “:folderid, :permissionid” werden die übergebenen Werte in das Berechtigungsobjekt gespeichert.

DELETE /api/v1/folders/:folderid/permissions/:permissionid

Header

```
{
	“x-access-token”: JWT
}
```

Body

```
{}
```

Löscht die mit den Parametern «:folderid, :permissionid» definierte Berechtigung.

SSL-Zertifikats-Bestätigung

GET /.well-known/acme-challenge/:certificateid

Header

```
{}
```

Body

```
{}
```

Gibt die Zertifikats-Verifizierungs-Zeichenkette aus. Diese Route wird für Let’s Encrypt verwendet.

6.1.7 Backend Deployment

Das Backend wird auf einem Cloud Server deployed.

Für das Deployment wird eine Virtuelle Machine verwendet, welche bei Hetzner gehostet wird.

Für das Backend muss Node.js und PostgreSQL installiert werden.

Die Datenbank «idocu» mit den definierten Zugangsdaten muss erstellt werden.

Anschliessend kann der Backend Code auf den Server geladen werden. Ist dieser hochgeladen müssen die einzelnen Node.js Komponenten installiert werden. `npm install` der Befehl installiert alle definierten Module.

Ist die Installation abgeschlossen kann die Datenbank initialisiert werden: `npm run-script build`. Der Build initialisiert vollautomatisch die komplette Datenbank mit allen Relationen. Anschliessend kann der Server mit `npm start` gestartet werden.

Automatisierung

Damit der Server kontinuierlich läuft kann via system deamon ein Dienst erstellt und gestartet werden.

SSL-Zertifikat

Die SSL-Zertifizierung ist vorbereit. Es muss lediglich per Letsencrypt ein Zertifikat manuell angefragt werden und das erstellte Zertifikat in den definierten Ordner geschoben werden. Danach funktioniert der Server mit SSL-Zertifikat.

6.2 Frontend Entwicklung

6.2.1 Struktur und Gliederung

Ordner Struktur

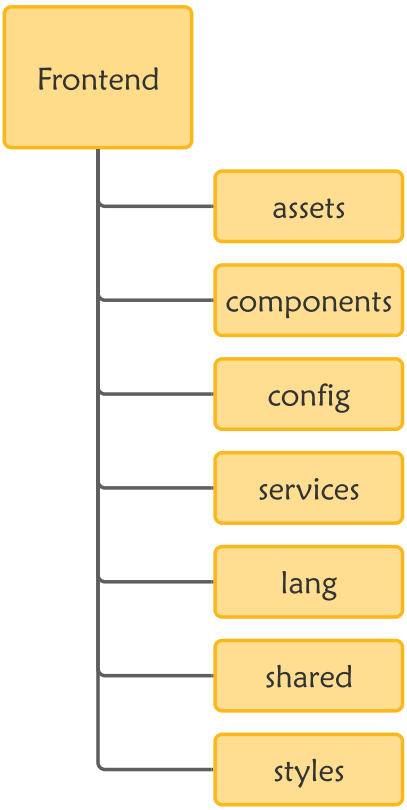


Abbildung 11: Gliederung

Die Struktur des Frontend ist mit folgender Logik aufgebaut.

Alles was global genutzt werden kann / soll, wird im Hauptordner in einem der entsprechenden Unterordner eingesiedelt.

Im Ordner "assets" liegen Bilder und Co. alles was für die Webseite direkt gebraucht wird.

Der Ordner "components" beinhaltet alle components, dazu weiter unten mehr.

Im Ordner "config" werden alle Konfigurationsdateien abgespeichert.

Alle Service Dateien werden im Ordner "services" abgespeichert.

Im Ordner "lang" sind alle .json Sprachdateien abgespeichert.

Der Ordner "shared" beinhaltet alle geteilten Komponenten, dazu weiter unten mehr.

Alle globalen Style Dateien werden im Ordner "styles" abgespeichert.

Gliederung Components

Die Gliederung der Komponenten ist speziell aufgebaut. Jede Komponente erhält ihren eigenen gleichnamigen Ordner. In diesem Ordner werden alle Dateien, welche zu dieser Komponente gehört abgespeichert. Ausserdem erhält die Komponente noch einen Unterordner namens "components", aber auch nur wenn diese Komponente, andere Komponente enthält welche ausnahmslos in dieser Komponente gebraucht wird.

Jede Komponente, welche in mehreren Komponenten verwendet wird, wird im "shared" Ordner abgelegt.

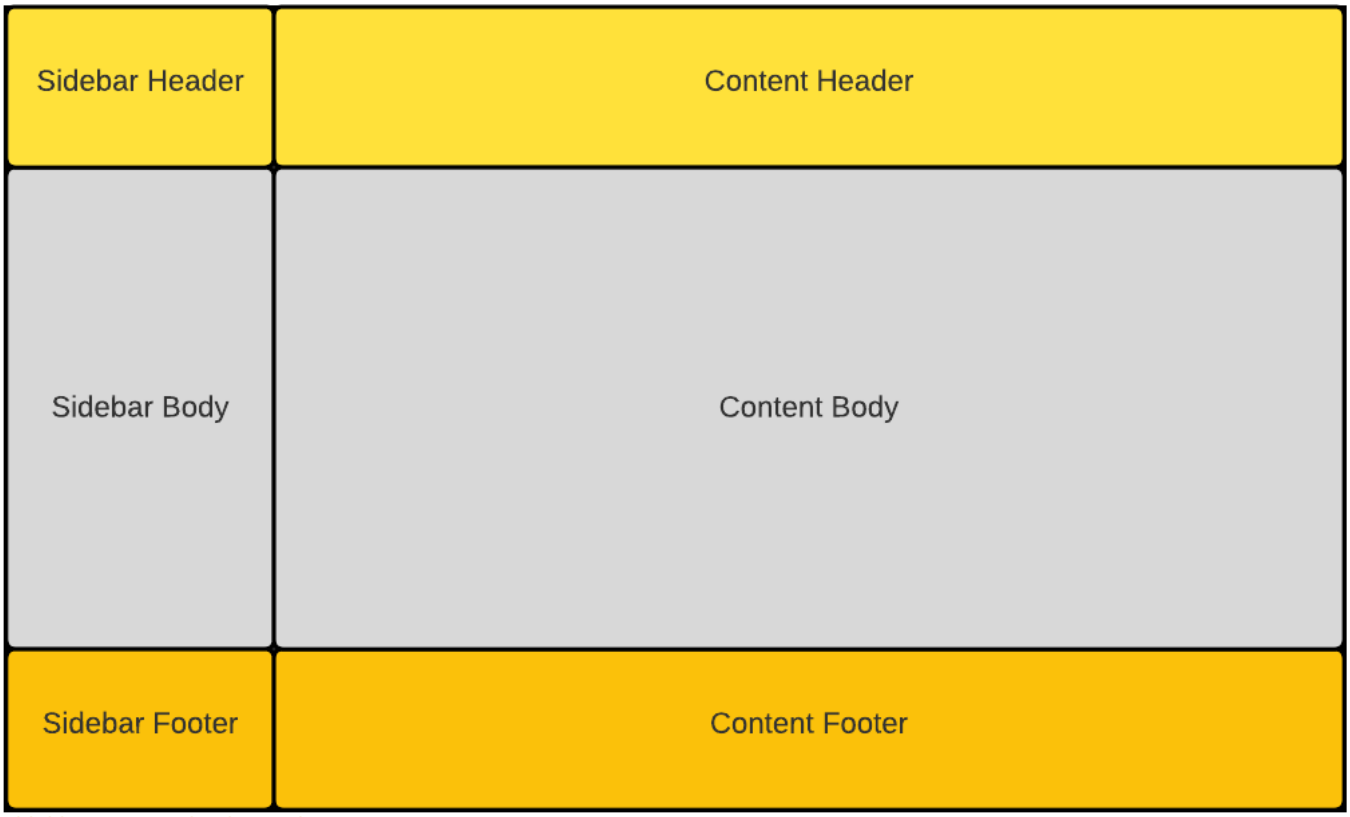


Abbildung 12:Graphische Struktur

Graphische Struktur

Die Webseite ist in Zwei Hauptbereiche aufgeteilt: «Sidebar» und «Content».

Der Bereich «Sidebar» hat eine definierte Breite von 270px im ausgeklappten und 70px im zusammengeklappten Zustand. Der Bereich «Content» verwendet den restlichen Platz, welcher nicht genutzt wird. Beide Bereiche nutzen 100% der Höhe.

Die jeweiligen Hauptbereiche sind nochmals in drei Bereichen unterteilt: «Header», «Body» und «Footer».

Der Bereich «Header» hat eine fest definierte Höhe von 74px. Der Bereich «Body» verwendet den gesamten nicht genutzt Platz, welcher übrigbleibt. Der Bereich «Footer» wird anhand des enthaltenen Inhaltes dargestellt. Der Bereich «Footer» wird nur dargestellt, wenn er Inhalt hat, hat er keinen Inhalt wird er nicht dargestellt.

6.2.2 Icon Sets

Für die Icons im Frontend wird das externe Modul «FontAwesome» verwendet. Dieses Modul enthält eine Vielzahl von Icons welche mit SVG dargestellt werden.

```
import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";
import { faSignInAlt, faFileSignature, faUserCircle, faSignOutAlt, faUserCog} from "@fortawesome/free-solid-svg-icons";
```

Abbildung 13: Import von FontAwesome

Das Integrieren in den Code kann mit dem Import der Hauptklasse und den einzelnen Icons umgesetzt werden. In der Abbildung 13 wird FontAwesome mit den Icons «SignInAlt», «FilesSignature», «UserCircle», «SignOutAlt» und «UserCog» importiert.

Im React HTML Code wird dann das Hauptobjekt definiert und als Parameter das importierte Icon übergeben. In der Abbildung 14 wird das FontAwesome Icon «UserCircle» dargestellt.

```
<FontAwesomeIcon icon={faUserCircle} />
```

Abbildung 14: Verwendung von FontAwesome

6.2.3 Services Entwickeln

Die Services dienen für die Datenverwaltung.

Alle Dateien im Ordner "services" welche den Dateinamen ".service" enthalten, arbeiten mit Daten im Backend.

Axios

Axios ist ein externes Node.js Modul, welches REST Abfragen sauber strukturiert. Mit einfachen Funktionen kann so eine komplexe Abfrage Logik integriert werden. Das Modul wird verwendet, um REST Abfragen mit dem entwickelten Backend auszuführen. Die Axios funktionen liefern jeweils ein Promise zurück. Die Fehlerbehandlung wird direkt in den Components abgewickelt, weshalb das Promise direkt weitergeleitet wird.

Auth-header

```
const authHeader = () => {
    const user = JSON.parse(localStorage.getItem("user"));

    if (user && user.accessToken) {
        return { 'x-access-token': user.accessToken};
    } else {
        return {};
    };
}

export default authHeader;
```

Abbildung 15: Auszug aus Services auth-header

Der Service "authHeader" liefert das strukturierte Backend-Token für die Abfragen für das Backend.

Das Token wird im localStorage als Teil des «user» Objekts abgespeichert.

Auth.service

Im «Auth.service» wird die Sitzung des Benutzers verwaltet.

Der Benutzer besitzt zwei JWT Tokens. Einmal das Access Token und einmal das Refresh Token. Das Access Token ist nur 24 Stunden gültig, das Refresh Token ist hingegen 30 Tage gültig.

Bei jedem Aufruf der Webseite wird das Refresh Token an das Backend zugesendet. Ist das Token noch gültig wird anhand der hinterlegten Benutzer-ID eine neue Sitzung erstellt und dem Frontend zugesendet. Ist das Token nicht mehr gültig verliert der Benutzer die Sitzung und er wird als Gast eingeloggt.

Die E-Mail-Verifikation wird via «verify» auch per JWT Token überprüft. Dabei ist der Token Aufbau der selbe wie beim Access-Token. Ist die Verifizierung erfolgreich, wird die Benutzer Sitzung erstellt und dem Frontend übergeben.

Auch für das Login, Logout und für das Registrieren des Benutzers ist «Auth.service» zuständig.

Document.service

Der «document.service» ist eine Klasse welche sämtliche Abfragen für die Dokumente enthält. Das Dokument braucht zwingen ein «parent». Mit «parent» ist ein Ordner gemeint.

```
create(document) {
    if (!document.parent) throw new Error("Parent id is required!");
    return axios.post(`${API_URL}folders/${document.parent}/documents/`, document, {headers: authHeader()});
}
```

Abbildung 16: Auszug aus service document.service

Die Klasse «document.service» enthält die Funktionen: «get, getList, post, create, update und remove». Alle diese Funktionen werden mit «axios», ein externes Modul, ausgeführt. Dokument muss jeweils einem Ordner zugewiesen werden, weshalb vor der der Erstellung noch überprüft wird, ob der «parent» gesetzt ist (Dies wird in der Abbildung 16 dargestellt). Die Berechtigung für das Erstellen des Dokuments wird im Backend abgewickelt. Die Funktionen liefern alle ein Promise zurück.

Folder.service

Der Service «folder.service» dient zum Erstellen, Laden, Überarbeiten und Löschen der Ordner. Der eigentliche Aufbau des «folder.service» ist gleich wie bei «document.service». Bei «getList» können mehrere Parameter übergeben werden um zum Beispiel nur die eigenen Ordner aufzurufen.

```
getList(folderid,ownOnly=false,parentOnly=false) {
    folderid = (folderid) ? `&folderid=${folderid}` : "";
    const own = (ownOnly) ? "&own=true" : "";
    const parent = (parentOnly) ? "&parent=false" : "";
    return axios.get(`${API_URL}folders/?orderName=name&orderASC=1${parent}${own}${folderid}`, {headers: authHeader()});
}
```

Abbildung 17: Auszug aus services folder.service

Wie bei der Abbildung 17 zu erkenn ist, kann via «folderid» die Ausgaben auf einen Ordner beschränkt werden. Anders als bei «get» dient hier die «folderid» nicht für das Ausgaben des eigentlichen Ordners, sondern für das Ausgaben der enthaltenen Ordner, des definierten Ordner. Standardmässig wird die Liste bereits anhand des Namens sortiert ausgegeben werden.

```
async getParentList() {
    const response = await this.getList();
    return this._buildParentList(response.data.result,null);
}

_buildParentList(list=[], parent=null) {
    const folderList = list.filter(folder => folder.parent === parent);
    folderList.forEach((folder) => {
        folder.childs = this._buildParentList(list,folder.id);
    })

    return folderList;
}
```

Abbildung 18: Die Funktion getParentList

Die Methode getParentList liefert kein Axios Promise Objekt zurück, sondern eine Multidimensionale Liste.

Die Hilfs-Methode _buildParentList wird verwendet, um die Verschachtelung der einzelnen Elemente vorzunehmen.

Die Methode getParentList wird zum Beispiel für die Navigation aufzubauen verwendet. Auch bei der «parent Liste» wird diese Methode genutzt.

Language

Die Ausgabe der Sprache wird mit diesem Service gemacht. Dieser Service greift nicht auf das Backend zu, weshalb er kein ".service" im Namen enthält.

```
import enUS from 'lang/enUS.json';
import deDE from 'lang/deDE.json';


const lang = {
    "enUS": enUS,
    "deDE": deDE
}
```

Abbildung 19: Sprach implementierung

Die Sprachpakete werden als «.json» Dateien im Ordner «lang» abgespeichert und als Variablen implementiert.

Die Variablen werden dann dem Objekt «lang» hinzugefügt. Dies ermöglicht später eine einfache Implementierung weitere Sprachen.

```
get(index, props = {}) {
    const user = authService.getCurrentUser();
    let currentLanguage = null;

    if (user?.lang) {
        currentLanguage = lang[user.lang];
    } else {
        currentLanguage = lang["enUS"];
    }
    if (currentLanguage[index]) {
        let text = currentLanguage[index];
        for (let key of Object.keys(props)) {
            text = this.replaceAll(text, key, props[key]);
        }
        return text;
    } else {
        console.log("Missing Language Entry: " + index);
        return index;
    }
}
```

Abbildung 20: Sprachmodul "get"

Das Sprachmodul enthält die funktion «get», womit der Text ausgegeben wird. Ist der Benutzer eingeloggt enthält das Benutzerobjekt die definierte Sprache des Benutzers. Ist keine Sprache definiert wird standardmässig die Sprache «enUS» ausgewählt. Der Sprachcode «enUS» ist US Englisch.

Der Parameter «index» wird im definierten Sprachpaket gesucht und wenn gefunden ausgegeben. Sind noch «props» definiert, werden diese im Text gesucht und jeweils ersetzt. Die Props dienen als Text-Variablen.

Wir der Parameter «index» nicht gefunden, wird dieser ausgegeben.

Permission.service

Der «permission.service» wird wie jeder «.service» für das kommunizieren mit dem Backend verwendet. Der Service ist für das Erstellen, Laden und Löschen der Berechtigungen zuständig, nicht aber für das Berechnen der Berechtigungen. Die Berechtigungsberechnung wird im Backend gemacht.

```
post(folderid, permission) {
    if (!folderid) throw new Error("folderid is missing");
    const permissionid = (permission.id) ? permission.id : "";
    const perm = {
        read: permission.read,
        write: permission.write,
        mod: permission.mod
    }

    if (permission.id) {
        perm.id = permission.id;
    }

    if (permission.userid !== null) {
        perm.userid = permission.userid;
    }

    return axios.post(`${API_URL}permissions/${folderid}/${permissionid}`,perm, {headers: authHeader()});
}
```

Abbildung 21: Auszug aus service permission.service

In der Abbildung 21 dargestellt wird die Berechtigung jeweils einem Ordner zugewiesen. Mit der Methode «post» kann die Berechtigung erstellt oder bearbeitet werden. Jede Berechtigung ist einzigartig. Pro Benutzer und Ordner wird eine Berechtigung erstellt. Die Berechtigung mit der «userid» «null» ist die Berechtigung für die Gäste.

Section.service

Der Service "section.service" dient zum Erstellen, Laden, Überarbeiten und Löschen der Sektionen in den Dokumenten. Der Service ist gleich aufgebaut wie die anderen. Für die REST Abfragen wir auch hier das externe Modul «axios» verwendet, welches ein Promise zurückgibt.

```
remove(section) {
    if(!section.id) throw new Error("Section id is required!!");
    if(!section.parent) throw new Error("Parent id is required!");

    return axios.delete(`${API_URL}documents/${section.parent}/sections/${section.id}/`,{headers: authHeader()});
}
```

Abbildung 22: Auszug aus service section.service

Speziell an der «Section» ist, dass diese jeweils an einem Dokument gebunden ist. Weshalb die Abfrage auch anders aufgebaut ist. Das Laden, Erstellen, Bearbeiten und Löschen der Sektion, bedingt somit immer beide Parameter. In der Abbildung 22 ist die Löschung der Sektion abgebildet, in dieser Abbildung ist zu sehen, dass beide Parameter erforderlich sind.

User.service

Der «user.service» wird für das Erstellen, Laden, Ändern und Löschen der Benutzer gebraucht. Jeder dieser funktionen ist normal mit dem externen Modul «axios» aufgebaut.

```
getDiskSpaceUsage(user) {
    return axios.get(`${API_URL}users/${user.id}/diskspace/`, {headers: authHeader()});
};
```

Abbildung 23: Auszug aus service user.service

Anders als die anderen «.service» Objekten enthält dieses noch die spezielle Methode «getDiskSpaceUsage». Der Benutzer hat eine Beschränkung des ihm zur Verfügung stehenden Speicherplatzes. Diese Methode dient zur Abrufung des benutzen Speicherplatzes.

6.2.4 Components Entwickeln

Die «Components» werden als Funtional React Components umgesetzt. Der Vorteil gegenüber klassenspezifische Components ist, dass das Update Statement via Hooks ausgeführt werden kann. Dies hat der Vorteil, dass bei einer Änderung eines Parameters direkt die Funktion ausgeführt wird und nicht noch eine Abfrage beim Update Statement gemacht werden muss. Es soll also die Laufzeit verbessern und die Verzögerung minimieren.

Anders als bei klassenspezifischen Components muss der «State» selbst definiert und abgefragt werden. Dafür können mehrere States an einem Component hinzugefügt werden. Ziel ist es so wenig States wie möglich auf der Component zu haben.

Shared Components

Im "shared" Ordner befinden sich alle geteilten Komponenten. Komponente welche in jeder anderen Komponente verwenden können.

contentContainer

Viele Componenten enthalten Inhalt, welcher geladen werden muss, bevor dieser dargestellt werden kann. Damit die Darstellung überall einheitlich ist und diese Darstellung nur einmal definiert werden muss, gibt es die Component «contentContainer»

Der ContentContainer zeigt im «Load» Status einen sogenannten Spinner (eine Grafik welche sich dreht) um den Leuten ein kleines Feedback zugeben, dass was geladen wird.

Ist das Laden abgeschlossen, wird entweder der Fehler angezeigt oder der eigentliche Inhalt dargestellt. Der Inhalt wird als «children» Parameter übergeben. (Siehe Abbildung 24)

```
{load ? (
    <span className="spinner-border spinner-border-sm"></span>
) : (
    error.message ? (
        <div className="error">
            {language.get(error.message)}
            {console.log(error)}
        </div>
    ) : (
        children
    )
)}
```

Abbildung 24: Auszug aus contentContainer

```
<ContentContainer error={error} loading={loading}>
```

Abbildung 25: ContentContainer implementierung

Der ContentContainer kann danach, wie in Abbildung 25 dargestellt, implementiert werden. Dabei erwartet die Component den Parameter «error» und «loading».

Link

Die Shared Component «Link» enthält weitere Funktionalitäten gegenüber dem vordefinierten «Link» Component von React.js. Diese Component wurde für spätere zwecke entwickelt, damit später eine Benutzerfreundliche Steuer implementiert werden kann.

```
const onMouseDown = (e) => {
    setState(now());
}

const onClick = (e) => {
    const time = now() - state;
    if (time > 200) {
        if (typeof onHoldClick === "function") {
            e.preventDefault();
            onHoldClick();
        }
    }
}
```

Abbildung 26: Auszug aus Shared Link

Doch was genau macht diese «Link» Component speziell? Die «OnClick» Funktion hat zwei Funktionen. Wird die Maus länger geklickt wird die «onHoldClick» Funktion aufgerufen. Diese Funktion habe ich selbst entworfen und soll für die gleiche Funktionalität wie bei Windows dienen. Beim längeren Klick soll der Name editierbar sein. Diese Funktion wurde noch nicht integriert, soll aber später integriert werden. Damit nicht alles umgebaut werden muss, wurde bereits dieses Element entwickelt und eingesetzt.

controlBar / controlBarItem

Die «ControlBar» wird vor allem im Content Bereich verwendet, doch kann «ControlBar» überall implementiert werden. Die «ControlBar» ist ein Bereich welcher «ControlBarItem» erwartet.

```
hasPermission.write() && (
    <ControlBar>
        <>
        {hasPermission.write() && (
            <>
            <ControlBarItem title={language.get("addDocument")} className="green" icon={<FontAwesomeIcon icon={faFile} />} onClick={controller.addDocument}/>
            <ControlBarItem title={language.get("addFolder")} className="green" icon={<FontAwesomeIcon icon={faFolder} />} onClick={controller.addFolder}/>
            <div className="clear" />
            </>
        )}
        {hasPermission.mod() ? (
            <>
            <ControlBarItem title={language.get("editFolder")} className="yellow" icon={<FontAwesomeIcon icon={faTools} />} onClick={controller.edit}/>
            <ControlBarItem title={language.get("editPermissions")} className="blue" icon={<FontAwesomeIcon icon={faUsersCog} />} onClick={controller.permission}/>
            <ControlBarItem title={language.get("deleteFolder")} className="red" icon={<FontAwesomeIcon icon={faTrashAlt} />} onClick={controller.delete}/>
            </>
        ) : null}
        </>
    </ControlBar>
)
```

Abbildung 27: ControlBar Implementierung (Folder Component)

Wie in der Abbildung 27 dargestellt, wird die «ControlBar» Als Vater Element eingefügt. In das «ControlBar» Element können «ControlBarItem» hinzugefügt werden. Diese werden dann wie in der Abbildung 28 dargestellt.

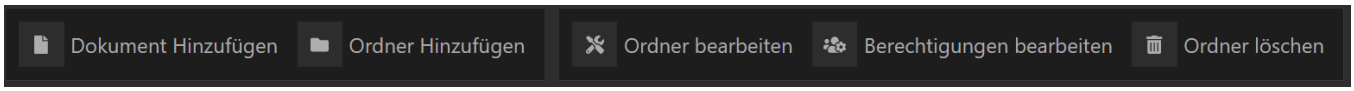


Abbildung 28: ControlBar auf der Webseite

Die «ControlBarItem» enthalten jeweils den Titel, zusätzliche CSS-Klassen, das Icon und die onClick Funktion.

Mit dieser Shared Component kann mit einfachen Mitteln, eine sauber strukturierte Control Bar erstellt werden.

ParentList

Die Shared Componenet «ParentList» dient zur Selektierung des Parent Folders.

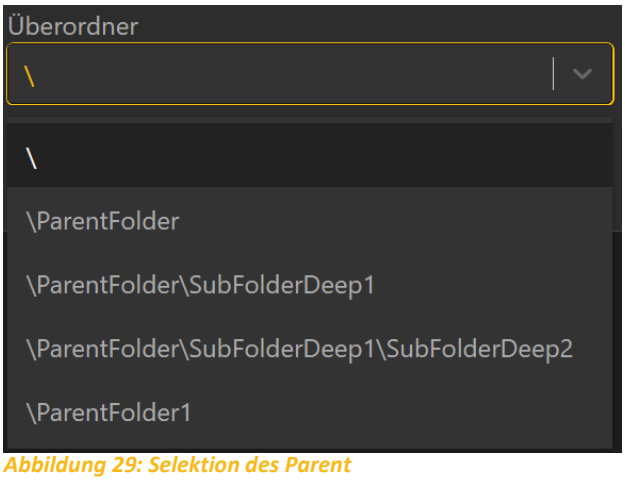


Abbildung 29: Selektion des Parent

Wie in Abbildung 29 dargestellt, wird die Liste verschachtelt dargestellt. Die «ParentList» ist so aufgebaut, dass der Parent jeweils vor dem eigentlichen Ordner Name geschrieben wird. Dies erlaubt eine klare Übersicht der «ParentList». Ausserdem ist die Liste alphabethisch und numerisch sortiert.

Die Abbildung 30 zeigt, wie die «ParentList» aufgebaut wird. Da die Ordnerstruktur nicht begrenzt ist, ist die «folderList» ein Multidimensionales Array, welches endlos in die Tiefe gehen kann. Jedes «folder» Element besitzt ein Array namens «childs» welche wieder «folder» Elemente besitzen kann. Ist die Liste leer wird die Leere Liste ausgegeben, enthält die Liste Elemente wird jedes Element dieser Liste hinzugefügt und ausgegeben.

```
const buildOptionList = (folderList=[], parent="\\",optionList=[]) => {
    folderList.forEach((folder) => {
        optionList.push({value:folder.id, label:`${parent}${folder.name}`});
        optionList = buildOptionList(folder.childs, `${parent}${folder.name}\\`,optionList);
    })
    return optionList;
}
```

Abbildung 30: BuildOptionList - Auszug aus ParentList

Die generierte Liste wird danach wie bei Abbildung 29, sauber an das «Select» Objekt übergeben und dargestellt.

Table

Die Component «Table» wandelt eine Liste mit Objekten zur Tabelle um.

```
const Table = React.forwardRef((
    {children, className, list, header, link, ...props}, ref,
) => {
```

Abbildung 31: Table Parameter

Die Tabelle erwartet mehrere Parameter, wobei nur die der Parameter «list» Pflicht ist, alle anderen Parameter können, müssen aber nicht übergeben werden.

Wie bei Abbildung 32 wird die «Table» als Component implementiert. In diesem Beispiel wird die Liste «objectList» als Liste übergeben, zusätzlich wird eine Liste als Header übergeben und ein Link Objekt.

```
<Table list={objectList} header={["name","updatedAt","type"]} link={link} />
```

Abbildung 32: Table Implementierung

```
const headerList = (header) ? header : (list.length !== 0) ? Object.keys(list[0]) : [];

return (
    <div ref={TableRef} className={classNames('table',className)} {...props}>
        <TableHeader list={headerList} sortFunction={sortFunction} />
        <TableContent list={list} link={link} headerList={headerList} />
        <TableFooter>{children}</TableFooter>
    </div>
)
```

Abbildung 33: Table Aufbau

In der Abbildung 33 ist dargestellt, wie die Table aufgebaut ist, sie besteht aus drei Components, «TableHeader», «TableContent» und «TableFooter» wobei der «TableFooter» nur die «children» bekommt. Ist keine «header» Objekt definiert, wird anhand der übergebenen Liste ein «header» Objekt erstellt.

```
<div ref={TableHeaderRef} className="tr">
    {list.map((h,index) => {
        return (
            <div
                className="td th"
                key={index}
                keyvalue={h}
                onClick={onClick}
            >
            {h}
            </div>
        )
    })}
</div>
```

In der Abbildung 34 ist zu sehen, wie der «Table Header» aufgebaut ist. Jedes Objekt der übergebenen Liste «list» erstellt eine neue Spalte der Tabelle. Wird auf den Spalten «Header» geklickt wird die «onClick» Funktion ausgeführt. Diese «onClick» Funktion wird für das Sortieren der Tabelle verwendet.

Es ist nicht möglich einzugrenzen welche Spalte sortiert werden kann, aktuell kann jeder Spalte sortiert werden.

Eine Filtrierung ist aktuell auch nicht implementiert.

Abbildung 34: Table Header

```
list.map((obj, index) => {
    return (
        <div className="tr" key={index}>
            {headerList.map((h,index) => {
                return (
                    <div className="td" key={index}>
                        {link[h] ? (
                            link[h].to && link[h].onHoldClick && link[h].onContextMenu ? (
                            <Link to={link[h].to(obj)} onHoldClick={link[h].onHoldClick(obj)} onContextMenu={link[h].onContextMenu(obj)} >
                                {obj[h]}
                            </Link>
                        ) : (
                            obj[h]
                        )) : obj[h]}
                    </div>
                )
            })}
        </div>
    )
})
```

Abbildung 35: Table Body

In der Abbildung 35 wird dargestellt, wie der Tabellen Inhalt aufgebaut wird. Jeder Eintrag der «HeaderList» wird als Spalte und jeder Eintrag der Liste als Zeile definiert. Das übergebene Link Objekt dient zum Verlinken der eigentlichen Einträge.

Popup

Die shared Component «Popup» wird nicht zur direkten Implementierung als Component verwendet, sondern wird funktional implementiert. Speziell an dieser Component, sie wird nicht wie jede andere Component in den Content gerendert, sondern erhaltet ihren eigenen render Container. Im vordefinierten Container «popup-container» welcher im Hauptdokument der Webseite erstellt ist, wird das Popup hinzugefügt. Dabei wird der restliche Inhalt des «popup-container» überschrieben. In der Abbildung 36 wird die openPopup Funktion dargestellt, welche den übergabe Parameter «children» als Component im Popup Element wiedergibt. Der zweite Parameter der Funktion «openPopup» ist ein Objekt mit möglichen Optionen, welches die Popup Component erwartet. Auch können noch weitere nicht definierte Optionen in diesem Optionen Objekt übergeben werden.

```
const openPopup = (children,
    {
        className = null,
        title = null,
        onClose = null,
        ...rest
    }) => {
    ReactDom.render(
        <Popup
            className={className}
            title={title}
            onClose={onClose}
            {...rest}
        >
            {children}
        </Popup>,
        document.getElementById("popup-container")
    );
}
```

Abbildung 36: openPopup Funktion

Die Popup Component wird als einzige Component nicht exportiert und kann ausserhalb der openPopup Funktion nicht verwendet werden.

Die Popup Component ist nicht für den Inhalt des Popups zuständig, sondern baut einen sauberen Container, welcher geschlossen werden kann. Die Kopfzeile (Header) welche in der Abbildung 38 abgebildet ist, wird mit dem Parameter «title» (Popup Titel) dargestellt. Durch den klick auf den Close button (kreuz) wird die funktion «close» ausgeführt. Welche die mitgeliefert «onClose» Funktion und die «closePopup» Funktion ausführt.

Die Popup Component kann durch diesen einfachen Aufbau überall verwendet werden.

Abbildung 38: Popup Kopfzeile (Header)

```
const Popup = React.forwardRef((
    { children, className, title, onClose, ...rest},
    ref,
) => {

    const close = () => {
        if (typeof onClose === "function") {
            onClose();
        }

        closePopup()
    }

    const popupRef = ref || React.createRef();

    return (
        <div
        ref={popupRef}
        className={classNames('popup', className)}
        {...rest}
        >
            <div className="popup-inner">
                <div className="popup-inner-header">
                    <span className="title">
                        {title}
                    </span>
                    <span className="close-wrapper" onClick={(close)}>
                        <span className="close">
                            <FontAwesomeIcon icon={faTimes} className="close" />
                        </span>
                    </span>
                </div>
                <div className="popup-inner-content">
                    {children}
                </div>
            </div>
        </div>
    )
})
```

Abbildung 37: Popup Component

```
import { openPopup } from "shared/components/popup";
openPopup(<AddFolder baseFolder={baseFolder} />, {title: language.get("Add Folder")})
```

Abbildung 39: Implementierung Popup

Wie bereits erwähnt wird anders als alle anderen Components das Popup funktional implementiert. Der import, importiert die Funktion «openPopup» welche dann wie im Beispiel Abbildung 39 die Component «AddFolder» als erster Parameter und das Option Objekt mit der Eigenschaft «title» als zweiter Parameter übergeben bekommt. Wird diese Funktion nun ausgeführt, öffnet sich der Popup mit der Component «AddFolder».

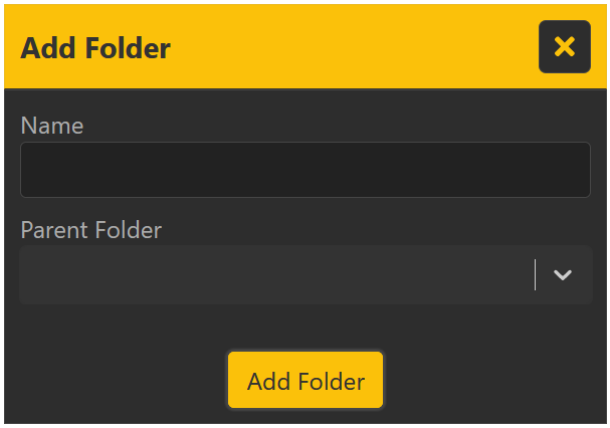


Abbildung 40: Add Folder Popup

In der Abbildung 40 wird dargestellt wie das geöffnete Popup aussieht.

Die eigentliche Funktion des Popups wird in der «AddFolder» Component geregelt.

Durch das Klicken auf den Close Button (Kreuz) wird das Popup geschlossen. Doch kann das Popup auch funktional geschlossen werden. Dies, da bei Erstellung des Ordners zum Beispiel auch das Popup geschlossen werden soll. Jede Component kann die Funktion «closePopup» (siehe Abbildung 41) importieren und ausführen. Die Funktion überschreibt den Inhalt des «popup-containers».

Der ganze Aufbau der Popup Funktionen macht den Umgang mit Popups einfach und kann so überall verwendet werden.

```
const closePopup = () => {
    ReactDom.render(<></>,document.getElementById("popup-container"));
}
```

Abbildung 41: Close Popup Funktion

Formular Validierung

Jedes Formular wird auf dieselbe Weise validiert. Für jedes Inputfeld wird anhand des definierten Namens eine «error» Variable, siehe Abbildung 43. Danach braucht es noch eine «setError» Variable, wo alle SetError funktionen eingefügt werden. Nach dem die Error Variablen definiert sind, kann in der «validation» Funktion (siehe Abbildung 42) jedes Input feld (als Name) definiert werden. Mit der zugehörigen Funktion kann die entsprechende validierung definiert werden. In der Abbildung 42 zu sehen, darf die Länge von Namen nicht null sein, ansonsten wird ein Fehler definiert. Beim «parent» ist es etwas anders. Es ist zwar dasselbe Prinzip, doch der Parameter kann nicht auf die Länge abgefragt werden, da das Objekt initial den Wert «null» besitzt.

```
const [errorName, setErrorName] = useState("");
const [errorParent, setErrorParent] = useState("");

const setError = {
    name: setErrorName,
    parent: setErrorParent
};
```

Abbildung 43: Beispiel Error Variablen

```
const validation = {
    name: async (value = folder.name) => {
        if (value.length === 0) {
            setErrorName(language.get("field is required"));
            return false;
        }
        setErrorName("");
        return true;
    },

    parent: (value = folder.parent) => {

        if (value === null) {
            setErrorParent(language.get("field is required"));
            return false;
        }
        setErrorParent("")
        return true;
    }
}
```

Abbildung 42: Validierungsfunktion

Die Validierungsfunktion kann nun im Absenden des Formulars abgefragt werden, sind alle Validierungen aller Inputfelder okay, kann das Formular abgeschickt werden.

Gliederung «components»

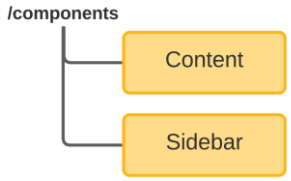


Abbildung 44: Components Struktur

Es gibt zwei Hauptkomponenten "Content" und "Sidebar".

Die Best-Practice wurde bewusst etwas vernachlässigt, da alle Komponenten in einem Ordner bei dieser Anzahl unübersichtlich wurden.

Im "Content" werden alle Komponenten gespeichert, welche zur Darstellung in diesem Container verwendet werden.

Die Sidebar enthält das gesamte Menü, inklusive Administration und Testconsole.

Content

Wie in Abbildung 45 dargestellt, ist die Struktur des Contents auf drei Bereiche unterteilt:
«Content Header», «Content Body», «Content Footer».

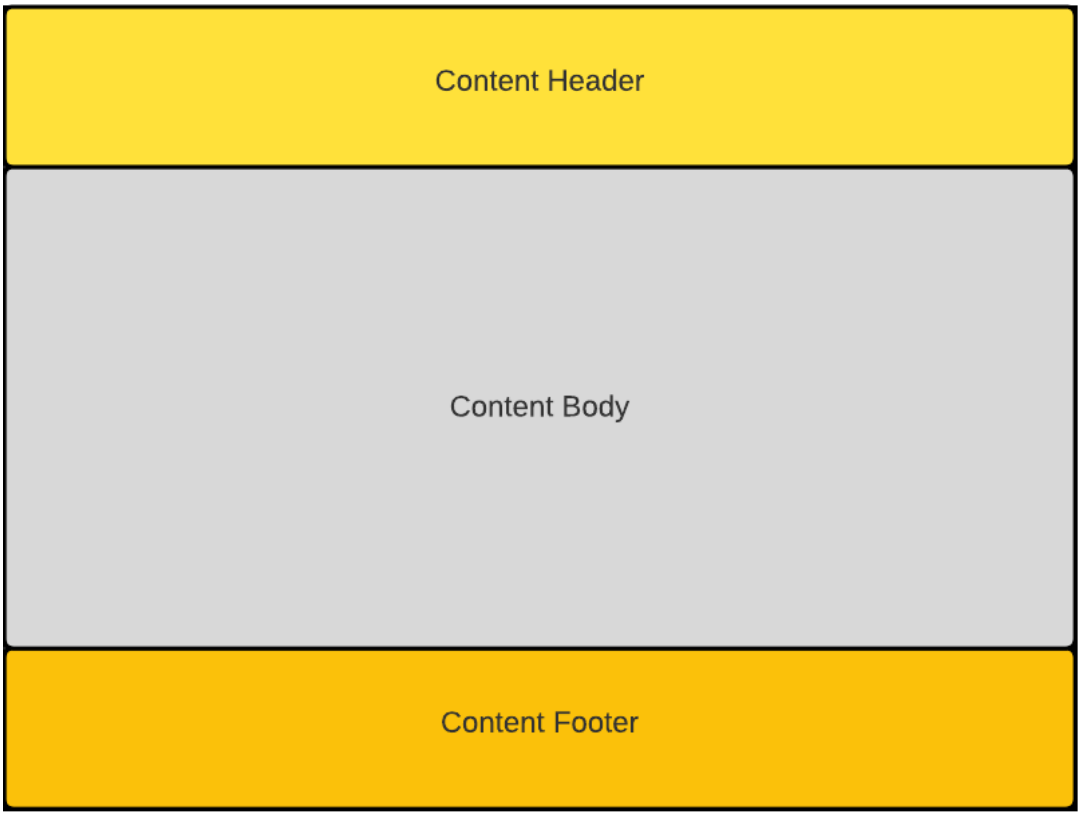


Abbildung 45: Graphische Struktur Content

Der Bereich «Header» hat eine fest definierte Höhe von 74px. Der Bereich «Body» verwendet den gesamten nicht genutzten Platz, welcher übrigbleibt. Der Bereich «Footer» wird anhand des enthaltenen Inhaltes dargestellt. Der Bereich «Footer» wird nur dargestellt, wenn er Inhalt hat, hat er keinen Inhalt wird er nicht dargestellt.

Content Footer

Der Footer Bereich erhält keine Component, da der Footer nur für spezielle Zwecke verwendet wird. Tritt ein solches Fall auf, wird per DOM eine neue Component in den Footer gerendert. Das ist das gleiche System wie beim Popup, nur mit einem anderen Master Container.

Content-Header

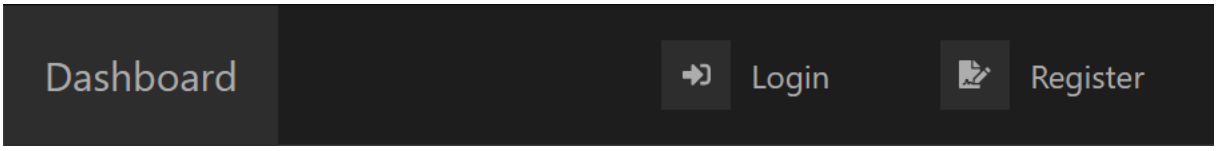


Abbildung 46: Content-Header

Im «Content Header» wird der aktuelle Seiten-Titel ausgegeben und die Komponente «User Panel» an der oben rechten Seite dargestellt.

Seiten Titel

Damit der aktuelle Titel angezeigt werden kann, wird die eingegebene URL abgefragt und bei Veränderung wird der Titel neu geladen.

```
const folderRoute = useRouteMatch({path:'/f/:folderid', exact:true});
const documentRoute = useRouteMatch({path: '/d/:documentid', exact:true});
const profileRoute = useRouteMatch({path: '/p/:profileid', exact:true});
const adminRoute = useRouteMatch("/admin/");
```

Abbildung 47: useRouteMatch Content Header

Die Variablen folderRoute, documentRoute, profileRoute und adminRoute enthalten jeweils das Hook «useRouteMatch» Das Hook dient dazu, die eingegebene URL zu überprüfen und bei Übereinstimmung die erwarteten Parameter zu übergeben. Wie in Abbildung 47 zu sehen ist, wird ein Parameter mit «:» definiert («:folderid» definiert die Variable «folderid»).

```
useEffect(() => {
    if (!folderRoute?.params?.folderid && !documentRoute?.params?.documentid) {

        if (profileRoute?.params?.profileid) {
            document.title = "Profile"
            setState({title: "Profile", obj:null});
            return
        }
         if (adminRoute) {
             document.title = "Administration";
             setState({title: "Administration", obj:null });
             return
         }

        setState({title: "Dashboard", obj: null});
        document.title = "Dashboard";
    }
    if (folderRoute?.params?.folderid) {
        folderService.get(folderRoute.params.folderid).then((response) => {
            setState({title: response.data.result.name, obj: response.data.result, type:"folder"});
            document.title = response.data.result.name
        }).catch((e) => {
            console.log(e);
        })
    }
    if (documentRoute?.params?.documentid) {
        documentService.get(documentRoute.params.documentid).then((response) => {
            setState({title: response.data.result.name, obj: response.data.result, type:"document"});
            document.title = response.data.result.name
        });
    }
},[folderRoute?.params.folderid,documentRoute?.params.documentid,profileRoute?.params.profileid]);
```

Abbildung 48: useEffect mit Routes

Bei der Abbildung 48 ist das useEffect «Hook» dargestellt, welches als Parameter, die oben definierten useRouteMatch Variablen enthält (Abbildung 48 unterste Zeile). Wird ein Dokument oder ein Ordner aufgerufen wird versucht das Dokument oder den Ordner zu laden und den Namen als Seitentitel, sowie als Titel des Contents zu definieren. Wird das Profile eines Benutzers aufgerufen, wird der Seitentitel auf «Profile» gesetzt. Die Administration bekommt den Titel «Administration». Alle anderen Seiten erhalten den Titel «Dashbaord», dabei ist gut zu wissen, dass es keine weiteren Seiten gibt. Die Titel sind bewusst nicht übersetzt worden.

User Panel

Die Komponente «User-Panel» verwaltet den aktuellen Benutzer und die Benutzer-Navigation.

Ist keine aktive Benutzer Sitzung aktiv wird das «User Panel» im nicht eingeloggten zustand geladen, der nicht eingeloggte Zustand wird in der Abbildung 46 dargestellt.

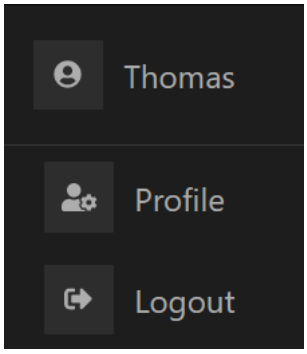


Abbildung 49: User-Panel

Ist der Benutzer eingeloggt wird die Navigation wie bei Abbildung 49 dargestellt.

In diesem Beispiel seht ihr den Benutzer «Thomas».

Der Button «Profile» öffnet die Profile Seite des aktiven Benutzers.

```
const logout = () => {
    authService.logout();
    window.location.reload();
}
```

Abbildung 50: Logout Button Funktion

Wird auf den Button «Profile» geklickt (Siehe Abbildung 49), wird die Funktion «logout» (Siehe Abbildung 50) ausgeführt. Eine Änderung der Sitzung benötigt ein neu Laden der Seite weshalb ein «window.location.reload()» ausgeführt wird.

Content Body

Anders als andere Bereiche, ist der Content Body nicht statisch. Die Component «Content» enthält den «Switch» Block, welcher je nach angegebener URL andere Components lädt und darstellt.

In der Abbildung 51 ist der Switch dargestellt, mit den definierten Routen. Die Pfade «/f», «/d», «/p», «/admin», «/» und «/home» werden geöffnet wenn die URL damit beginnt.

```
<Switch>
  <Route exact path={["/", "/home"]} component={Folder}/>
  <Route path="/f" component={Folder} />
  <Route path="/d" component={Document} />
  <Route path="/p" component={Profile} />
  <Route path="/admin" component={Admin} />
</Switch>
```

Abbildung 51: Content Switch

Wird eine nicht definierte URL eingegeben, öffnet sich eine Seite mit leerem Content, ohne Fehlermeldung.

Folder Component

https://idocu.app/f/fc63657b-b8be-4a10-892c-2a37fd9b4c76

Die oben dargestellte URL öffnet die «Folder» Component.
Als Übergabewert ist die Folderid (UUIDv4) [fc63657b-b8be-4a10-892c-2a37fd9b4c76]

Das heisst, die Base URL «/f» öffnet immer die Component «Folder» auch mit weiteren Parametern.

Die Folder Component listet tabellarisch alle enthaltenen Ordner und Dokumente, des definierten Ordners auf. Wird kein Ordner definiert, wird das Wurzelverzeichnis dargestellt. Dies entspricht allen Dokumenten, welche kein definierten «parent» haben.

Der Link Dashbaord, welche mit dem Link «/» öffnet, verwendet auch die Folder Component, da dort keine Folderid definiert ist, wird das Wurzelverzeichnis dargestellt.

Wie in der Abbildung 27 (Seite 34) dargestellt wird die shared Component ControlBar implementiert. Die implementierte Component wird wie bei Abbildung 52 aussehen, wenn die benötigten Rechte vorhanden sind.

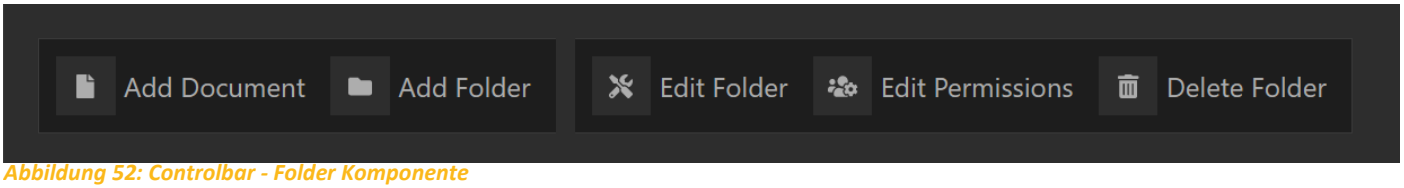


Abbildung 52: Controlbar - Folder Komponente

Mit Klick auf einer der Buttons wird die jeweilige hinterlegte Funktion aufgerufen. Jeder dieser Button öffnet ein Popup.

Add Folder Component

Diese Component kann per Funktion «openAddFolder» als Popup geöffnet oder normal implementiert werden. Die Add Folder Component besitzt ein Formular, welches zur Eingabe der benötigten Daten benutzt wird. Zur Validierung wird die Standardvalidierung verwendet. Wichtig zu erwähnen, der Wert «parent» kann Null sein, da aber die Validierung mit «null» eine Fehlermeldung auslöst, weil «null» nichts ausgewählt bedeutet, wird das Wurzelverzeichnis «/» mit «parent» ausgefüllt. Beim Absenden des Formulars wird nun der Wert «parent» wieder zu «null» und kann so dem Backend übergeben werden.

Ist der Ordner erstellt, wird die Seite neu geladen.

Add Document Component

Diese Component kann per Funktion «openAddDocument» als Popup geöffnet oder normal implementiert werden. Die Add Document besitzt gleich wie «Add Folder Component» ein Formular, welches zur Eingabe der benötigten Daten benutzt wird. Anders als bei «Add Folder Component» kann der «parent» Wert aber nicht das Wurzelverzeichnis enthalten. Die Selektierung wird entsprechend anders dargestellt. Das Dokument besitzt immer ein Parent, es ist also immer einem Ordner zugewiesen. Ansonsten funktioniert diese Component identisch wie «Add Folder Component».

Edit Folder Component

Diese Component ist identisch zur Add Folder Component mit dem Unterschied, dass diese Werte erhalten und entsprechend in das Formular lädt. Ausserdem werden nur die geänderten Daten gespeichert.

Delete Folder Component

Diese Component kann per Funktion «openDeleteFolder» als Popup geöffnet oder normal implementiert werden. Diese Component dient zur bestätigung des Lösch vorgangs, wird auf den «Delete» button geklickt wird die Löschung initiert. Sobald die Löschung abgeschlossen ist, wird die Meldung «Folder deleted» ausgegeben und der «Delete» button wird zu «Close».

Edit Permission Component

Die Edit Permission Component kann per Funktion “openEditPermission” geöffnet oder normal implementiert warden. Das Erstellen, Bearbeiten und Löschen der Berechtigungen wird in «Edit Permission Component» gemacht. In der Abbildung 53 ist zu sehen, wie der Popup aussieht, wenn der Benutzer «Test» Lese Rechte hat und nach dem Benutzer «Thomas» gesucht wird.

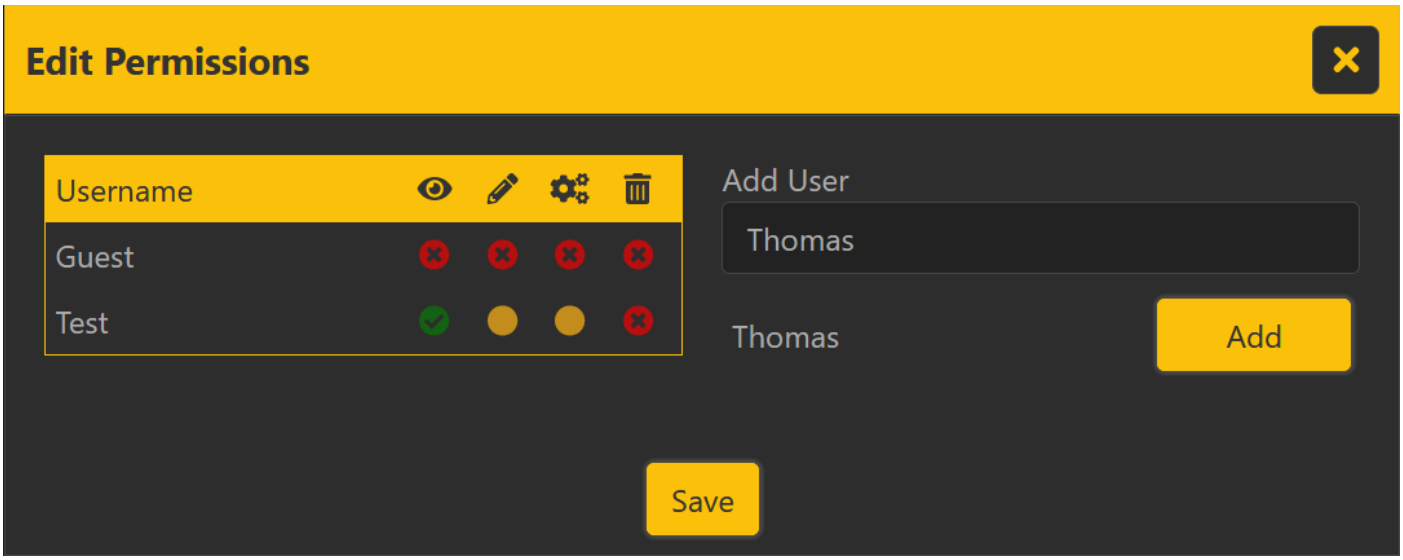


Abbildung 53: Berechtigungs-Editierung

Die Darstellung der Berechtigung hat drei Status. Der Standardstatus ist der Gelbe Punkt (Nicht Berechtigt). Wird nun auf den Punkt geklickt wird die Berechtigung auf den grünen Hacken (Berechtigt) gewechselt. Wird nochmals geklickt wird die Berechtigung auf das rote Kreuz (Verboten) gewechselt.

Damit alle Änderungen einfach entdeckt und gespeichert werden können, gibt es 2 Variablen. Die «PermissionList» und einmal die «ChangeList» beide Listen enthalten die Berechtigungen. Die «PermissionList» wird initial geladen und gesetzt. Die «ChangeList» kopiert initial den Inhalt der PermissionList. Jede Änderung, die gemacht wird, wird in der ChangeList gespeichert.

```
PermissionService.getList(baseFolder).then( async (response) => {
    let hasGuest = false;

    for (const permission of response.data.result) {
        try {
            if (permission.userid === null) {
                hasGuest = true;
                permissionList.push({
                    ...permission, user: {id: null, username: language.get("Guest")}
                })
            } else {
                const res = await userService.get(permission.userid);
                permissionList.push({
                    ...permission, user: res.data.result
                });
            }

        } catch(e) {
            console.log(e);
        }
    }

    setPermissionList(permissionList.map((p) => p));

    if (hasGuest === false) {
        permissionList.push({
            read:null, write: null, mod: null, userid: null, user: {id: null, username: language.get("Guest")}
        })
    }

    setChangeList(permissionList);
```

Abbildung 54: Initialisierung der Berechtigungslisten (Edit Permission)

Das «Guest» Objekt (welches die Berechtigung für die Gäste verwaltet) wird erstellt, wenn dieses nicht in der Liste ist. Der Gast besitzt standardmässig keine Berechtigung.

```
const onUserSearch = (e) => {
    const username = e.target.value;
    if (username.length > 3) {
        userService.getList(username).then((response) => {
            let userList = response.data.result;
            changeList.forEach((permission) => {
                userList = userList.filter((user) => user.id !== permission.user.id);
            });
            setUserList(userList);
        }).catch((e) => {
            console.log(e);
        })
    } else {
        setUserList([]);
    }
}
```

Abbildung 55: Benutzer Suchfunktion (Edit Permission)

In der Abbildung 54 wird die Funktion «onUserSearch» dargestellt. Die «onUserSearch» Funktion sucht anhand des eingegebenen Benutzernamen und gibt allen gefundenen Benutzern aus (Ausgenommen, das eigene Objekt). Die Eingabe muss mehr als 3 Zeichen lang sein, damit die Suche angestossen wird.

Wird nun auf den Button «Save» geklickt, wird die Speicherfunktion angestossen. Mit den beiden Listen («PermissionList» und «ChangeList») werden nun miteinander verglichen. Alle Elemente, welche sich geändert haben, werden gespeichert. Entfernte Objekte werden gelöscht.

Document Component

Die Document Component enthält eine ControlBar Component und das Section Component. Ausserdem lädt die Component das entsprechende Dokument, anhand des URL Parameters.

Das Dokument besitzt nur die eigene Control Funktion, das hinzufügen der Sektion geschieht nicht über einen Popup, sondern direkt im Container.

Da die variable «document» eine reservierte Variable von Javascript ist, wird hier die Variable vom Standard abweichen. Entweder wird die Kurzform «doc» oder die lange Form «documentState» verwendet.

Die Controlbar hat die Funktion EditDocument und DeleteDocument, beide dieser Component funktionieren identisch zu EditFolder und DeleteFolder.

SectionContainer Component

Wie der Name bereits verrät, handelt es sich um «SectionContainer Component» um einen Container, für die Sektionen. Der Container stellt die vorhandenen Sektionen dar und wickelt die Berechtigung für das Erstellen einer neuen Sektion ab.

Section Component

Der Wert «Content» vom Sektionen Objekt wird als Blob zugesendet. Damit der Blob als Inhalt auf der Seite angezeigt werden kann, muss der Blob Objekt in einen String konvertiert werden. Dies funktioniert mit der Funktion «Utf8ArrayToStr». Die Funktion wandelt die Bits in eine lesbare Zeichenkette. Der Vorteil dieser Speicherung ist, dass so kompletter HTML Inhalt und anderer Binärer Inhalt abgespeichert werden kann.

Als Texteditor wird das externe Modul «TinyMCE» verwendet. Dieses Modul ist eines der beliebsten und bekanntesten Module, was WYSIWYG Editore anbelangt. Der Editor hat unzählige Plugins und unterstützt fast alles. Die Bilder können direkt als Base64 in den Inhalt integriert werden. So kann der Benutzer praktisch jeden Inhalt den er sich wünsch in die jeweilige Sektion speichern. Auch wenn späte noch Tabellen, Videos, Präsentation oder ähnliches integriert werden möchte, muss das jeweilige Plugin in den Editor integriert werden und die Funktion steht zur Verfügung.

Die Benutzer besitzten jeweils ein Maximum an Speicherplatz. Dies betrifft nur die Sections, da nur dort wirklich Daten erzeugt werden. Weshalb beim Speichern der Sections jeweils überprüft wird, ob das Limit überschritten wird oder nicht. Wird das Limit überschritten, wird die Fehlermeldung «not enough space» ausgegeben.

```
const oldSize = new Blob([content]).size;
const newSize = new Blob([newContent]).size;
const currentUsed = getCurrentUsedSpace();
const difSize = newSize - oldSize;

if ((currentUsed + difSize) <= getCurrentQuota()) {
```

Abbildung 56: Used Space Berechnung

In der Abbildung 56 wird dargestellt, wie der bestehende Inhalt (content) und der neue Inhalt (newContent) miteinander verglichen wird. Da der Inhalt nicht als Blob versendet werden kann, wird das Objekt in ein Base64 Objekt umgewandelt. Das Base64 Objekt ist um einiges grösser als das eigentliche Blob Objekt. Im Backend wird das Base64 Objekt wieder zurück gewandelt in ein Blob objekt und abgespeichert. Dieses Vorgehen, kann ein kleiner Unterschied zwischen dem errechneten Wert und dem eigentlich genutzten Speicher ergeben.

Müsste man den genauen Wert haben, müsste das Objekt zuerst zum Backend gesendet und berechnet werden. Da dies aber die Laufzeit stark belasten würde, wird das oben erwähnte Vorgehen verwendet.

Der Inhalt «content» wird nicht als Text ausgeben, sondern als HTML Code in ein HTML Container hinzugefügt. So kann jedes HTML Objekt sauber dargestellt werden.

Profile Component

Das Profil Component enthält wie «Folder Component» und «Document Component» eine URL Parameter, als UUIDv4 für die Benutzer ID. Das Benutzerprofil wird dargestellt, wenn die Benutzer-ID identisch zur eigenen Benutzer-ID ist, oder der eigene Benutzer Administratoren Rechte besitzt.

Im Profil kann der Benutzer bearbeitet und gelöscht werden. Das Editieren der Quota und der Benutzergruppe, kann nur von Administratoren durchgeführt werden, weshalb diese beiden ControlBarItems nur mit der entsprechenden Berechtigung dargestellt werden.

EditQuota Component

Die Quota definiert den maximal zur Verfügung stehenden Speicherplatz. Die Quota wird als Int gespeichert. Welches die Kapazität in Bytes definiert.

Damit nun das Einstellen der Quota etwas einfacher ist, wird ein Quota Berechner verwendet.

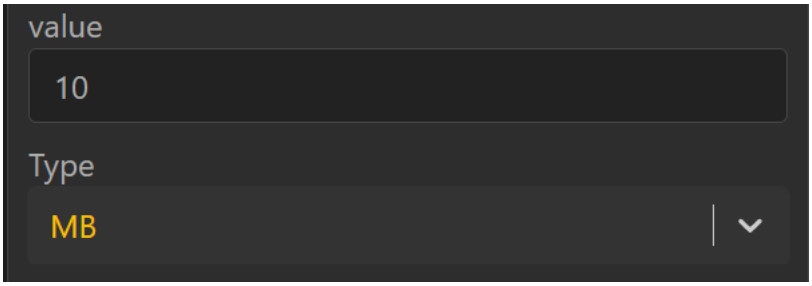


Abbildung 57: Edit Quota

Im EditQuota Component sind zwei Inputfelder, eines, um die Zahl einzugeben und eines, um die Grösse zu definieren.

Die Auswahl geht von B bis zu TB, wobei beim Value auch mehr als 1000 eingegeben werden können.

```
const quotaCalculation = (value, type="B") => {
    if (type === "B") return value;

    switch(type.toUpperCase()) {
        case "TB":
            type="GB";
            break;
        case "GB":
            type="MB";
            break;
        case "MB":
            type="KB";
            break;
        case "KB":
            type="B"
    }

    value = value * 1024;
    return quotaCalculation(value, type);
}
```

Abbildung 58: Quota Calculation

In der Abbildung 58 wird dargestellt, wie die «Quota Calcution» Funktion aufgebaut ist. Je nach übergebener Type wird das Value multipliziert mit 1024. Dies entspricht der offiziellen Byte Berechnung. Diese Multiplikation wird so oft ausgeführt bis der Type «B» also Byte entspricht. So wird zum Beispiel 10 GB zu 10'485'760 Byte.

Es gibt auch eine «byteCalculation» Funktion. Welche die Bytes das höchstmöglichste rechnet. Dabei wird auf 2 Ziffern nach dem Komma gerundet. Die Ausgabe ist eine Zeichenkette «Zahl, Type»

```
const byteCalculation =(value, type="B") => {
    if (value > 1024) {

        value = value / 1024;

        switch(type.toUpperCase()) {
            case "B":
                type="KB"
                break;
            case "KB":
                type="MB"
                break;
            case "MB":
                type="GB"
                break;
            case "GB":
                type="TB"
                break;
        }
        return byteCalculation(value, type);
    } else {
        value = Math.round(value * 100) / 100;
        return `${value} ${type}`;
    }

}
```

Abbildung 59: ByteCalculation

Sidebar

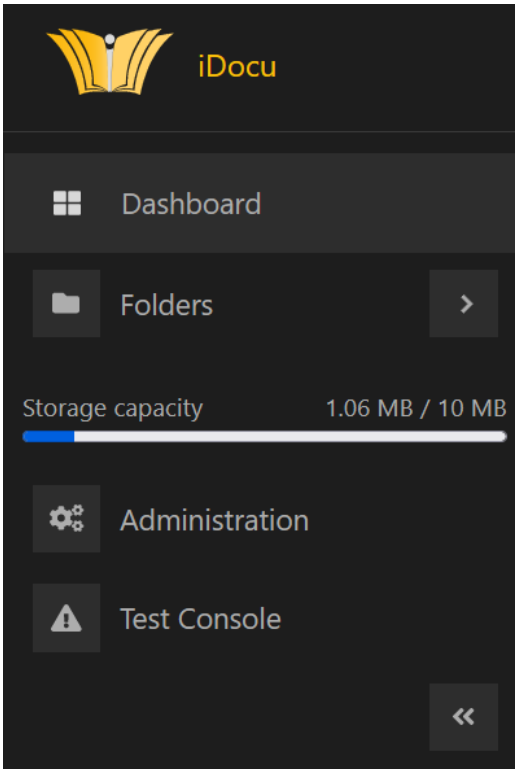


Abbildung 60: Sidebar

In der Kopfzeile der Sidebar wird das Logo mit Schriftzug dargestellt.

Oberhalb der Ordnerstruktur wird der Menüpunkt "Dashboard" dargestellt.

Das Menü ist speziell aufgebaut, da es theoretisch ins Endlose gebaut werden kann. Praktisch ist das nicht möglich, da irgendwann die Darstellung unmöglich ist. Aber jeder Menüpunkt kann Unterpunkte enthalten, es gibt keine Begrenzung.

«Storage Capacity» Zeigt den aktuellen Stand des verwendeten Speicherplatzes. Diese Anzeige wird nur dargestellt, wenn sich der Benutzer angemeldet hat.

Nur wer Administrator ist, sieht den Menüpunkt Administration. Dieser Menüpunkt öffnet die Administration der Benutzer.

Benutzer welcher der Gruppe "Tester" angehören, sehen die spezielle Komponente "Testconsole" diese öffnet eine Console, welche die Fehlermeldungen ausgibt. Ausserdem können so die Aufgaben der Tester ausgegeben werden. Diese Komponente dient aktuell eher als Testfunktion und wird beim Release wieder entfernt.

Ganz unten kann die Sidebar kleiner / grösser gemacht werden.

Shared

6.2.5 Styles definieren

Die Farben in den jeweiligen ".css" / ".scss" Dateien wurden per globale Variable definiert.

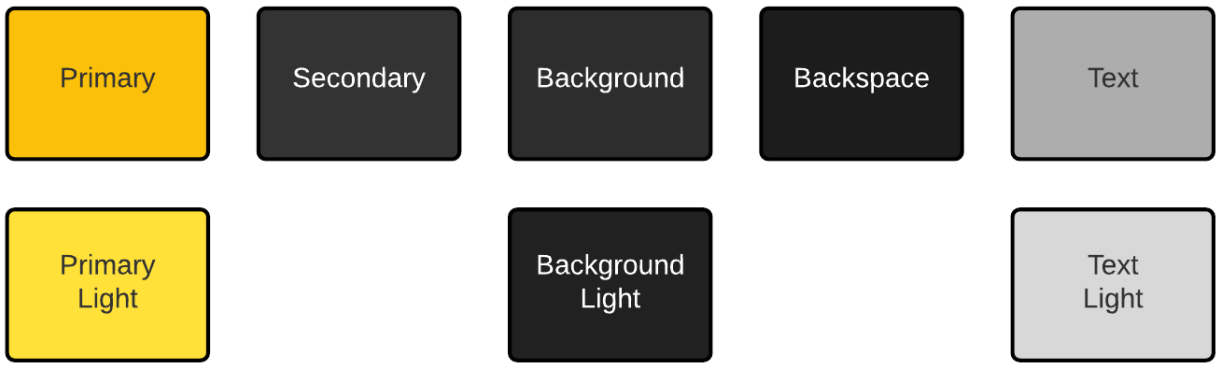


Abbildung 61: Das Farbschema (Darkmode)

Im Farbschema (Abbildung 61) sehen sie die definierten Grundfarben.

6.2.6 Frontend Deployment

Das Frontend wird mit einer Virtual Machine betrieben. Die VM wird bei Hetzner gehostet.

Auf der VM muss nur Node.js und Letsencrypt installiert werden.

Sind beide Packete installiert kann der Frontend Code auf den Server geladen werden. Sobald dieser geladen ist, kann per `npm install` die Node.js Module installiert werden.

Damit der Server funktioniert muss ein gültiges SSL-Zertifikat hinterlegt sein.

SSL-Zertifikat

Im Script 'ssl.js' ist ein ACME definiert, welcher beim Abrufen ausgegeben wird. Für das Verifizieren per letsencrypt muss `npm run-script ssl.js` ausgeführt werden, mit der angegebenen ACME.

Das gültige SSL-Zertifikat muss in den definierten Ordner kopiert werden, danach kann die Webseite gestartet werden.

Automatisierung

Damit der Server kontinuierlich läuft kann via system deamon ein Dienst erstellt und gestartet werden.

7 Projektabschluss

7.1 Evaluation der Projektergebnisse

Erfolgskriterien	Resultat
1) Funktionalität	
a) Ein Benutzer mit E-Mail-Adresse und Passwort ist angelegt.	Der Benutzer «Test» mit der E-Mail-Adresse «test@gaspel.com» und einem Passwort ist angelegt.
b) Ein Ordner ist erstellt. 1) Der Ordner ist öffentlich zugänglich 2) Der Ordner ist öffentlich nicht zugänglich	Zwei Ordner sind angelegt, einer öffentlich einsehbar einer nur privat einsehbar.
c) Der Speicherplatz ist auf 10 MB eingegrenzt	Der Speicherplatz des Users «Test» ist auf 10MB eingegrenzt. Es kann kein weiterer Inhalt in eine «Section» eingefügt werden.
d) Eine Berechtigung für einen Ordner ist erstellt.	Der User «Thomas» hat auf den TestOrdner vom User «Test» Zugriff erhalten.
e) Die Freigabeberechtigung für einen User wurde angepasst.	Die Berechtigung für den TestOrdner wurde angepasst. Thomas hat nun keinen Zugriff mehr.
f) 3 Bilder wurden erfolgreich hochgeladen und dem Dokument zu geordnet.	3 Bilder wurden dem Test Dokument im Ordner Root von Thomas hochgeladen und werden dargestellt.
g) Der Text wurde fett und unterstrichen formatiert.	Der Text im «Test» Dokument im Ordner «Root» von Thomas wurde fett und unterstrichen formatiert.
h) Einem Ordner wurden 3 Dokumente hinzugefügt	Der Ordner «Root» vom Benutzer Thomas enthält 3 Dokumente «Test», «Test2», «Test3».
i) Ein Kunde wurde gelöscht	Der Kunde «Test2» wurde erfolgreich gelöscht.
j) Ein Ordner wurde gelöscht	Der Ordner «TestDelete» von Thomas wurde gelöscht.
k) Ein Dokument wurde gelöscht	Das Dokument «TestDelete» Im Ordner «Root» vom Benutzer «Thomas» wurde erfolgreich gelöscht.

Tabelle 5: Evalurieren Erfolgskriterien

Erfolgskriterien	Resultat
2) Benutzeroberfläche	
a) Die Struktur der Ordnerübersicht unterscheidet sich zu maximal 30%	Die Ordnerstruktur listet die Ordner mit Data Modified und Folder Type auf. Somit sieht die Strukturierung fast identisch aus.
b) Die App wird mit Deutscher und Englischer Sprache angeboten	Im Profil kann die Sprache umgestellt werden. Die Sprachen «Deutsch» und «Englisch» stehen zur Auswahl. Wird die entsprechende Sprache ausgewählt wird diese geladen und dargestellt.

Tabelle 6: Evaluierung Erfolgskriterien - Benutzeroberfläche

Erfolgskriterien	Resultat
3) Technische Aspekte	
a) Das Frontend ist mit React.js entwickelt	Ja, das Frontend wurde mit React.js entwickelt
b) Das Backend ist mit Node.js entwickelt	Ja, das Backend wurde mit Node.js entwickelt
c) Die Datenverwaltung ist mit PostgreSQL umgesetzt.	Ja, Die Datenverwaltung ist mit PostgreSQL umgesetzt.

Tabelle 7: Evaluierung Erfolgskriterien - Technische Aspekte

7.2 Webseite Testen

Anhand der Erfolgskriterien habe ich ein Testformular definiert, welches bekannte von Mir ausgefüllt haben. So konnte ich die Webseite auf Funktionalität und Verständnis überprüfen lassen.

7.2.1 Tester

Vorname	Nachname	Beruf	Sprache
Martin	Plüss	Polymechaniker	Deutsch / Englisch
Hans	Plüss	ICT – Account Manager	Deutsch / Englisch
Brigitte	Weidmann	Damenschneiderin	Deutsch
Daniel	Weidmann	Werkstattleiter	Deutsch
Samuel	Hansen	Dipl. HF Automationstechniker	Englisch
Roy	Hansen	System Engineer	Englisch
Romina	Engweiler	Logistikerin	Deutsch
Robin	Engweiler	Kaufmann	Deutsch / Englisch
Marko	Manke	2nd Level Supporter	Deutsch / Englisch
Christian	Hoffmann	Medien Gestalter	Deutsch / Englisch

Tabelle 8: Webseiten Tests - Tester

Das Formular wurde jeweils anonym abgesendet, weshalb ich die Antworten nicht zuordnen kann.

7.2.2 Ergebnis

Die ganzen Testergebnisse sind im Anhang zu finden.

Die meisten Tester konnten die Webseite erfolgreich Testen. Nur wenige hatten Probleme, die meisten Probleme waren aber die Bedienung der Webseite und nicht die Funktionalität. Die Auswertung entstand durch direkte Rückfragen der Tester.

Frage 1: Kunden können sich auf der Webseite registrieren und anmelden

Alle Tester konnten sich Registrieren und Anmelden. Wobei einige User Mühe hatten bei der E-Mail-Verifikation.

Frage 2: Der Kunde kann mindestens einen privaten Ordner erstellen

Die meisten Tester (9/10) konnten einen privaten Ordner erstellen. Wobei der eine Tester, welche keinen privaten Ordner erstellen konnte, hat die Frage bzw. die Antwort nicht richtig gelesen. Sein Ordner war privat.

Frage 3: Jeder Kunde kann seine eigenen Ordner verwalten, inklusive Berechtigung

Alle Tester konnten den Ordner umbenennen und die Berechtigungen ändern. Was einem Tester nicht gelungen ist, das Verschieben in einem Ordner. Auch da war wieder ein Bedienungsfehler.

Frage 4: Der Kunde kann mindestens einen öffentlich zugänglichen Ordner erstellen

Die meisten Tester (8/10) konnten die Freigabe einstellen und testen. Die zwei Benutzer, welche dies nicht konnten, haben einen nicht «root» Ordner freigegeben. Diese Ordner werden nicht im Dashboard angezeigt, weshalb sie die Freigabe nicht richtig geprüft haben. Da die beiden Tester die freigegebenen Ordner gelöscht haben, konnte ich diese Behauptung aber leider nicht bestätigen. Weshalb ich bei diesem Punkt nochmals Zeit investieren muss.

Frage 5: Einem Ordner können Dokumente hinzugefügt werden

Alle Tester konnten ein Dokument erstellen.

Frage 6: Bilder können hochgeladen und eingefügt werden können

Alle Tester bis auf einen konnten Bilder hochladen und einfügen. Der Tester, welcher das Bild nicht hochladen konnte, versuchte eine nur für Mac Kompatible Bilddatei hochzuladen.

Frage 7: Der Text kann formatiert werden

Bei dieser Frage sind mehrere Antwort Möglichkeiten vorhanden.

Alle Benutzer konnten den Text Fett Kursiv und farblich anpassen. Allerdings frage ich mich wie sie den Text unterstreichen konnten, denn diese Option wurde bewusst deaktiviert. Da muss ich nochmals die Tester fragen und analysieren.

Frage 8: Der Speicherplatz, den der Kunde zur Verfügung hat, ist eingegrenzt

Alle Tester konnten die Speicherplatz Begrenzung testen und verifizieren. Während der Testphase hat sich ein gravierender Fehler bemerkbar gemacht, welcher ich währenddessen korrigiert habe. (Beim Erstellen der Sektion und Überschreitung des Limits stürzte die Webseite ab.)

Frage 9: Die Benutzeroberfläche soll sich am Windows Explorer (Ordner Ansicht) orientieren

Die Antworten waren unterschiedlich, aber alle haben Ähnlichkeiten entdeckt.

Frage 10: Zu wie viel Prozent orientiert sich die Benutzeroberfläche dem Windows Explorer (Ordner Ansicht)?

Der Durchschnittswert liegt bei 81%

Frage 11: Die App soll mehrsprachig entwickelt werden

Bei dieser Frage sind mehrere Antwort Möglichkeiten vorhanden.

Alle Tester hatten die Sprache Englisch (Was Standard ist). Nur ein Tester hat es nicht geschafft die deutsche Sprache auszuwählen. Ein Tester konnte sogar Französisch und Russisch auswählen, obwohl diese Option nicht zur Verfügung steht.

Frage 12: Dokumente können gelöscht werden

Jeder Tester konnte seine Dokumente löschen und nicht mehr öffnen.

Frage 13: Ordner können gelöscht werden

Jeder Tester konnte seine Ordner löschen und nicht mehr öffnen.

Frage 14: Kunden können gelöscht werden

Jeder Tester konnte sein Account löschen und nicht mehr einloggen.

Frage 15: Siehst du einen Verwendungszweck in deinem Umfeld

Die meisten Personen 8/10 konnten das App in Ihrem Umfeld verwenden. Für mich ein klares Zeichen, dass die Webapp ihre Berechtigung hat.

Frage 16: Wie gefällt dir das WebApp?

Die Durchschnittliche Bewertung liegt bei 4.00. Dies liegt wohl an der Bedienung der WebApp.

Frage 17: Dein Feedback zu meiner Arbeit

Einige Tester haben hier ihr persönliches Feedback geschrieben. Die meisten bemängeln die Bedienbarkeit, was leider stimmt. Die Bedienbarkeit ohne Anleitung ist etwas schwierig, da absolut keine Hilfestellung vorhanden ist.

7.2.3 Reflektion des Tests

Der öffentliche Test hat mir geholfen, vor allem die Bedienbarkeit zu optimieren. Aber auch Fehler zu finden, welche ich nicht gefunden hatte. Für mich hat es auch gezeigt, dass einige Leute sich genau ein solches Tool wünschen würden.

7.3 Reflektion und Erkenntnisse

Ich habe einiges beim Planen und Umsetzen des Projektes gelernt. In den folgenden Seiten werde ich tiefer in diese Themen eingehen.

7.3.1 Reflektion zum Projekt

Frage	Welche Begebenheiten waren für das Erreichen der Projektziele förderlich?
Antwort	Ich habe inmitten der Diplomarbeit 2 Wochen «Urlaub» genommen. Weshalb ich so sehr viel Zeit in das Projekt stecken konnte. Bevor das Projekt richtig begonnen hat, habe ich viel Zeit in die Vorbereitungen gesteckt. Vorgängig habe ich mich mit React.js und Node.js auseinandergesetzt.

Frage	Welche Begebenheiten waren für das Erreichen der Projektziele hinderlich?
Antwort	Ich wurde kurz vor beginn der Diplomarbeit, das erste Mal Papa, was dazu geführt hat. Dass mein gewohnter Tagesablauf kurz vor Projektbeginn über den Haufen geworfen wurde.

Frage	Welche Handlungsweisen haben die Effizienz positiv beeinflusst?
Antwort	Während meiner Diplomarbeit, habe ich immer wieder nach Feedback von Drittpersonen verlangt. Dies hat mich immer weiter Motiviert das Projekt voranzutreiben.

Frage	Welche Handlungsweisen haben die Effizienz negativ beeinflusst?
Antwort	Trotz Vorbereitung im Thema React.js habe ich zu beginn des Projekts eine unübersichtliche Ordner Struktur definiert, weshalb ich inmitten des Projekts die gesamte Ordner Struktur des Frontend neu definieren und aufbauen konnte. Dies hat mich mindestens 3 Arbeitstage gekostet.

Frage	Was habe Ich im Rahmen dieses Projektes gelernt?
Antwort	Ich habe sehr viel über den Aufbau von Node.js (Express) und React.js gelernt. Auch über den Aufbau und Programmierung einer Ordner Struktur plus Freigabesystems.

Frage	Was würdest du anders machen beim nächsten Projekt?
Antwort	Beim nächsten Projekt würde ich nicht mehr React.js verwenden, sondern React TypeScript. Bei komplexeren Programmen sind Typen Definitionen einfach von Vorteil. Ausserdem würde ich das Berechtigungssystem anders aufbauen. Aktuell benötigt das Vererben der Berechtigung zu viel Ressourcen.

Frage	Was machst du beim nächsten Projekt ähnlich?
Antwort	Ich habe das Backend sowie Frontend von nicht Projektbezogenen Personen testen lassen. Dies gab mir ein guter Einblick in die Nutzungsweise meines Produkts und zeitgleich ein unabhängiges Feedback.

7.3.2 Reflektion zur Umsetzung

Backend – Module

Der Aufbau der Module ist okay, doch die Relationen hätte ich anders nennen sollen. Beim Definieren der «parent» Eigenschaft, dachte ich, wenn ich alles gleich nenne, wird dies übersichtlicher und einfacher. Doch dies war gerade das Gegenteil. Im Nachhinein würde ich diese Relationen beim Namen nennen, welche sie haben. Das Heisst das Dokument hätte nicht die Eigenschaft «parent», sondern «folderid». Sektion hätte nicht die Eigenschaft «parent», sondern «documentid». So wäre jedem klar, um was für ein Parent «Objekt» es sich jetzt handelt.

Dasselbe gilt für die Eigenschaft «owner». Obwohl der «owner» bei jedem Objekt derselbe type ist, wäre die Relation «userid» deutlicher.

Backend - Routes

Nach dem umsetzten der Diplomarbeit ist mir aufgefallen, dass der für mich logisch gewählte Aufbau der Routen keinen Sinn ergibt. Beim Dokument wird der «parent» als Objekt-Eigenschaft übergeben, wohingegen bei der Sektion der «parent» als übergeordnete Route übergeben wird. Für das bessere Verständnis hätte ich dort eine einheitliche Form wählen sollen.

Frontend – DiskSpace

Bei der Webapp muss es möglich sein, den Speicherplatz einzugrenzen. Ich habe diese Funktion etwas unterschätzt und zu spät integriert. Die eigentliche Abfrage hätte mittels Middleware ins Backend integriert werden müssen. Doch als ich diese implementiert habe, wäre es nur mit grossem Aufwand machbar gewesen, weshalb ich die Funktion dann im Frontend implementiert habe. Im Backend ist lediglich die Abfrage des benötigten Speicherplatzes.

Dieser Fehler hat mir einmal mehr gezeigt, wie wichtig Planung ist.

Frontend – Axios

Für die REST-Abfragen führe ich mit dem externen Modul Axios aus. Damit ich standardisierte Abfragergebnisse erhalte. Bei der Planung hat dies gut ausgesehen, doch bei der Umsetzung ist es mir bereits aufgefallen und als ich fertig war, war es mir auch klar. Dieses Modul ist in meinem Programm doppelt gemoppelt. Ich sende bereits via Backend standardisierte Ergebnisse und mache im Frontend nochmals dasselbe. Beim Umsetzen ist es mir aufgefallen, dass ich jeweils nur das Ergebnis selbst brauche, alle anderen Daten welche Axios bietet verwende ich nicht. Da ich beim Backend bereits alle Daten im Ergebnis habe, welche ich benötige.

Was habe ich daraus gelernt? Ich brauche nur auf einer Seite der Schnittstelle eine Standardisierte Abfrage (Dies meiner Meinung nach am besten im Backend). Im Frontend hätte ich eine einfache RESTful Abfrage per Javascript ausführen können und hätte kein externes Modul installieren und implementieren müssen.

8 Schlusswort

Die Grundlage für eine saubere Struktur und Gliederung von Dokumentationen und Anleitung wurde gemacht. Doch fehlen noch einige Bausteine. Das Ziel ist es, Vorlagen zu definieren können. Die Dokumente sollen via Grid ein Raster Aufbau erhalten und eine ToDo Liste soll erstellbar sein.

Ausserdem wurde ein «Level-System» gewünscht, welches den Leuten die Möglichkeit bietet die Anleitungen und Dokumentationen nur einblenden zu lassen, wenn es dem entsprechenden Level entspricht.

Mir hat es Spass gemacht, die Webapplikation zu entwickeln und so viel positives Feedback zu erhalten, ich bin guter Dinge, dass diese Applikation gebraucht wird.

9 Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit mit dem Titel «iDocu (Intelligent Dokumentation Manager)» selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Sämtliche Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, habe ich als solche kenntlich gemacht.

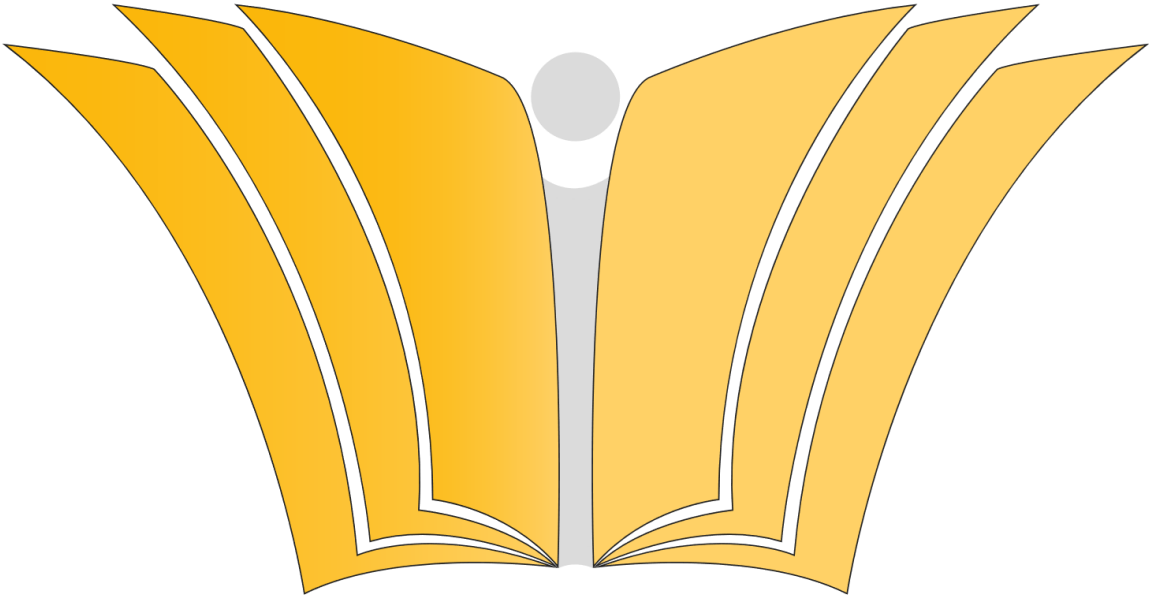
Hiermit stimme ich zu, dass die vorliegende Arbeit in elektronischer Form mit entsprechender Software überprüft wird.

Safenwil, 24. Oktober 2021

Thomas Engweiler

10 Anhang

- Pflichtenheft
- Testergebnisse



Idocu

Pflichtenheft

Autor: Thomas Engweiler
Projekt: Intelligenter Dokumentation Manager
Projekt-Art: Diplomarbeit für Informatiker HF

Inhaltsverzeichnis

1 Einleitung

Das Tool «Intelligenter Dokumentation Manager» kurz iDocu, soll sich auf Dokumentationen und Anleitungen jeglicher Form fokussieren.

Mit vielen Programmen, werden bereits Dokumentationen oder Anleitungen geschrieben, doch keines bietet eine saubere Struktur und intelligente Verwaltung an. Aus dem Gedanken entstand das Projekt «iDocu».

Im Rahmen der Diplomarbeit werden die Grundlagen des Programmes entwickelt und möglichst umgesetzt.

1.1 Begriffe

Hier werden alle Themen spezifische Begriffe erklärt.

Begriff	Bedeutung
iDocu	Intelligenter Dokumentation Manager
Folder(s)	Ordner
Document(s)	Dokument(e)
Section(s)	Bereich(e)
Header	Kopfzeile der Webseite.
React.js	**React** ist eine JavaScript-Softwarebibliothek, die ein Grundgerüst für die Ausgabe von User-Interface-Komponenten von Webseiten zur Verfügung stellt (Webframework).[1]
Node.js	**Node.js** ist eine plattformübergreifende Open-Source-JavaScript-Laufzeitumgebung, die JavaScript-Code außerhalb eines Webbrowsers ausführen kann. [2]
PostgreSQL	Ist ein Open Source objektrelationales Datenbanksystem.
Nginx	Ist eine Webserver-Software fürs Bereitstellen von Webseiten, Reverse Proxy und weiteren Webserver Dienste.
Docker	**Docker** ist eine Freie Software zur Isolierung von Anwendungen mit Hilfe von Containervirtualisierung.[3]
Docker compose	Docker Compose ist ein Tool zum Definieren und Ausführen von Docker-Anwendungen mit mehreren Containern.[4]
vCPU	Eine virtuelle CPU (vCPU), auch bekannt als virtueller Prozessor, ist eine physische CPU, die einer virtuellen Maschine (VM) zugeordnet ist.[5]

[1] Quelle: https://de.wikipedia.org/wiki/React
[2] Quelle: https://de.wikipedia.org/wiki/Node.js
[3] Quelle: https://de.wikipedia.org/wiki/Docker_(Software)
[4] Quelle: https://en.wikipedia.org/wiki/Docker_(software)
[5] Quelle: https://www.computerweekly.com/de/definition/Virtuelle-CPU-vCPU

2 Zielbestimmung

Das zu entwickelnde Produkt soll, für das Erstellen von Dokumentationen und Anleitungen genutzt werden können.

Jeder Kunde kann so viele Ordner erstellen, wie er möchte. Dabei soll er entscheiden können, wer alles auf diesen Ordner Zugriff hat.

Auch Gäste sollen Zugriff erhalten können.

2.1 Musskriterien

2.1.1 Funktionalität

- Kunden können sich auf der Webseite registrieren und anmelden
- Jeder Kunde erhält mindestens einen Ordner
- Jeder Kunde kann öffentliche Ordner erstellen
- Jeder Kunde kann private Order erstellen
- Der Speicherplatz des Kunden kann eingegrenzt werden
- Jeder Kunde kann seine eigenen Ordner verwalten, inklusive Berechtigungen
- Ob ein Kunde einen privaten Ordner für andere Freigeben kann, soll steuerbar sein
- Bilder müssen hochgeladen und eingefügt werden können
- Text muss formatierbar sein
- Einem Ordner können Dokumente hinzugefügt werden
- Ein Kunde kann gelöscht werden
- Ein Ordner kann gelöscht werden
- Ein Dokument kann gelöscht werden

2.1.2 Benutzeroberfläche

- Die Benutzeroberfläche soll sich am Windows Explorer orientieren.
- Die App soll mehrsprachig entwickelt werden. (Deutsch, Englisch)

2.1.3 Technische Aspekte

- Das Frontend soll mit React.js entwickelt werden
- Das Backend soll mit Node.js entwickelt werden
- Die Datenverwaltung wird mit PostgreSQL gehandhabt

2.2 Wunschkriterien

2.2.1 Funktionalität

- Die Dokumente können mit einem Klick zu einer ToDo Liste umgewandelt werden
- Die fertigen Dokumente können als PDF exportiert werden

2.2.2 Benutzeroberfläche

- Es soll möglich sein, zwischen "dark" und "white" Theme zu wechseln

2.3 Auswahlbegründung

2.3.1 Frontend mit React.js

Das Frontend einer Webapplikation wird mit Javascript entwickelt. Javascript hat viele Frameworks unter anderem React.js diverse Frameworks bieten verschiedene Module. React.js ist relativ simple gehalten, was das Einbinden von eigenen Modulen leicht macht. Was genau optimal ist für das Projekt.

2.3.2 Backend mit Node.js

Das Backend einer Webapplikation kann mit jeder Programmiersprache umgesetzt werden, die es gibt. Die beliebtesten sind dabei: C# (Mit .NET), Python, PHP oder eben Javascript (Mit Node.JS). Node.JS wird immer mehr von Programmierern verwendet und wird immer wieder weiterentwickelt. Ausserdem ist Javascript stark verbreitet, was eine Weiterentwicklung fördert.

2.3.3 Datenverwaltung mit PostgreSQL

Das Projekt ist eine Webapplikation, welche viele Manipulationen und Datenhandling benötigen wird. Die Performance von PostgreSQL im Vergleich zu MySQL ist um einiges besser. Ausserdem können in PostgreSQL direkt JSON Daten gespeichert werden, wohingegen dies in gängigen SQL-Datenbanksystem nicht ohne Performance Einbussen möglich ist.

3 Produkteinsatz

Das Produkt soll Kunden ermöglichen Anleitungen einfach und übersichtlich zu erstellen. Dabei gibt es keine Eingrenzungen in welchem Fachbereich das Tool verwendet wird.

3.1 Anwendungsbereich

In jedem nur denkbaren Anwendungsbereich, wo Dokumentationen oder Anleitungen verwendet werden.

3.2 Zielgruppen

Projekt- / Line-Manager sollen einfach und unkompliziert das Programm nutzen.

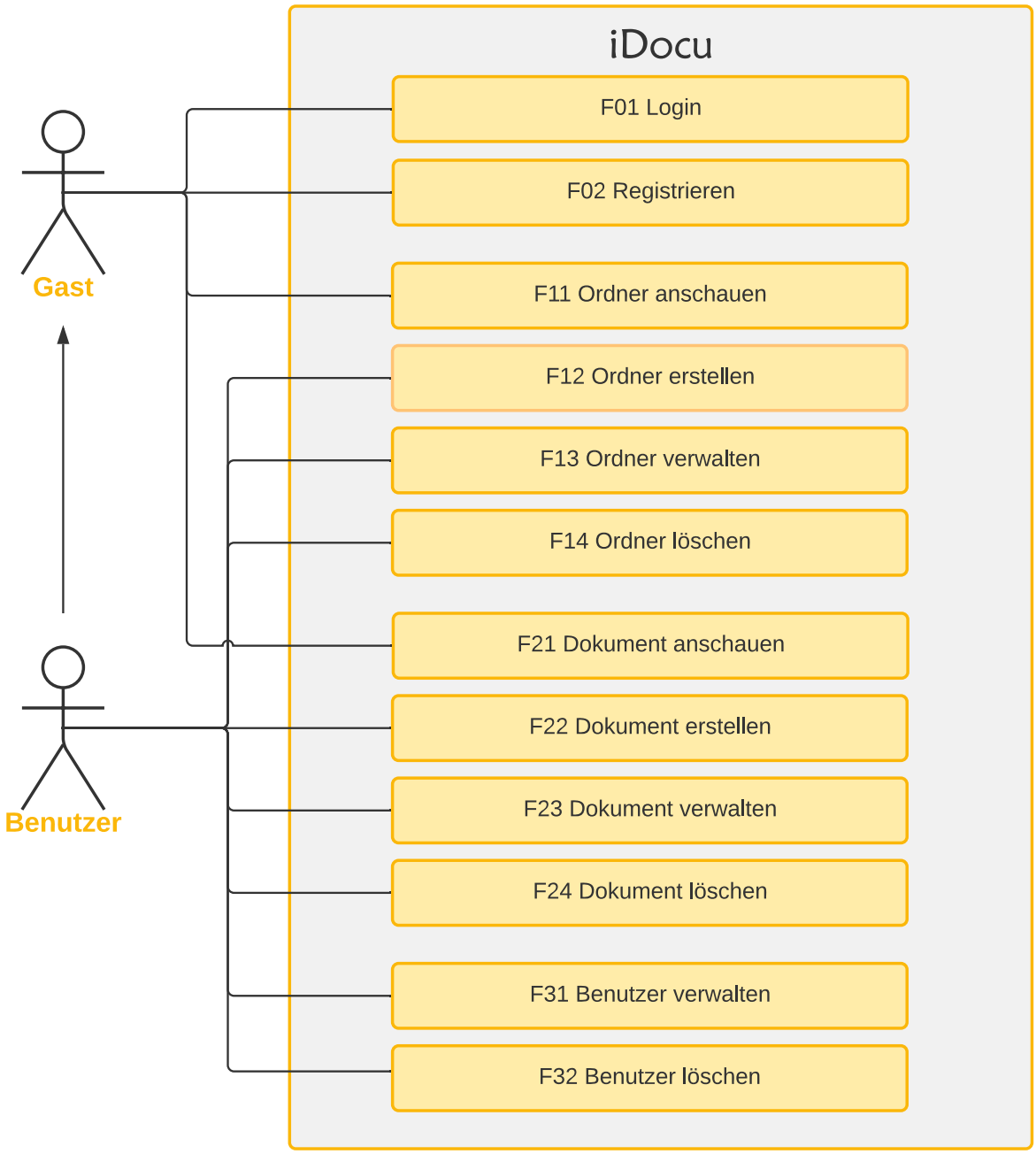
3.3 Betriebsbedingungen

Das komplette Programm soll auf mindestens einem Server mit mehreren Instanzen laufen. Frontend und Backend sollen voneinander getrennt werden.

Das Programm soll 24 Stunden erreichbar sein.

4 Produktübersicht

Bei der Produktübersicht wird gezeigt welche Funktionen wer ausführen kann.



Mit «Benutzer» ist ein Registrieren Kunde gemeint, welcher eingeloggt ist.

4.1 Geschäftsprozesse

Eine Auflistung aller Funktionalen Prozesse.

F01 Login		
Geschäftsprozess	**Login**	
Ziel	D01 Sitzung wird erstellt	
Vorbedingung	D02 Benutzer existiert	
Nachbedingung Erfolg	D01 Sitzung wurde erstellt	
Nachbedingung Fehlschlag	D01 Sitzung wurde nicht erstellt	
Akteur	Gast	
Auslösendes Ereignis	Aufruf der Webseite	
Beschreibung	01 02	Logindaten eingeben D01 Sitzung wird erstellt
Erweiterungen		
Alternativen	01a 02a	Gespeicherte Sitzung laden Fehlermeldung wird ausgegeben

F02 Registrieren		
Geschäftsprozess	**Registrieren**	
Ziel	D02 Benutzer wird erstellt	
Vorbedingung	D02 Benutzer existiert nicht	
Nachbedingung Erfolg	D02 Benutzer wird erstellt	
Nachbedingung Fehlschlag	D02 Benutzer konnte nicht erstellt werden	
Akteur	Gast	
Auslösendes Ereignis	Interaktion der Benutzeroberfläche	
Beschreibung	01 02 03 04 05	Benutzerdaten eingeben Datenüberprüfung via Backend E-Mail Verifikation Registrierung beenden F01 Login
Erweiterungen		
Alternativen		

F11 Ordner anschauen		
Geschäftsprozess	**Ordner anschauen**	
Ziel	D11 Folder mit D13 Documents geladen und dargestellt	
Vorbedingung	D11 Folder existiert	
Nachbedingung Erfolg	UI erfolgreich geladen	
Nachbedingung Fehlschlag	D02 User hat kein Zugriff	
Akteur	Gast, Benutzer	
Auslösendes Ereignis	Interaktion der Benutzeroberfläche	
Beschreibung	01 02 03 04	D01 Session auf Gültigkeit prüfen Berechtigung mit D12 Permission überprüfen D11 Folder Daten laden Seite darstellen
Erweiterungen		
Alternativen	01a	F01 Login ausführen

F12 Ordner erstellen		
Geschäftsprozess	**Ordner erstellen**	
Ziel	D11 Folder erfolgreich erstellt	
Vorbedingung	D11 Folder existiert nicht	
Nachbedingung Erfolg	D11 Folder existiert	
Nachbedingung Fehlschlag	D02 User hat keine Berechtigung	
Akteur	Benutzer	
Auslösendes Ereignis	Interaktion der Benutzeroberfläche, F02 Registrieren	
Beschreibung	01	D01 Session überprüfen
	02	Berechtigung mit D12 Permission überprüfen
	03	Formular ausfüllen
	04	D11 Folder Datensatz erstellen
	05	F11 Ordner anschauen ausführen
Erweiterungen		
Alternativen		

F13 Ordner verwalten		
Geschäftsprozess	**Ordner verwalten**	
Ziel	D11 Folder erfolgreich geändert	
Vorbedingung	D11 Folder existiert	
Nachbedingung Erfolg	D11 Folder wird gespeichert	
Nachbedingung Fehlschlag	D02 User hat keine Berechtigung, Fehler beim Speichern der Daten	
Akteur	Benutzer	
Auslösendes Ereignis	Interaktion der Benutzeroberfläche	
Beschreibung	01	D01 Session überprüfen
	02	Berechtigung mit D12 Permission überprüfen
	03	Änderungen eingeben
	04	D11 Folder Datensatz speichern
	05	F11 Bereich anschauen ausführen
Erweiterungen		
Alternativen		

F14 Ordner löschen		
Geschäftsprozess	**Ordner löschen**	
Ziel	D11 Folder mit allen D13 Document löschen	
Vorbedingung	D11 Folder existiert	
Nachbedingung Erfolg	D11 Folder mit allen D13 Document gelöscht	
Nachbedingung Fehlschlag	D02 User hat keine Berechtigung	
Akteur	Benutzer	
Auslösendes Ereignis	Interaktion der Benutzeroberfläche	
Beschreibung	01	D01 Session überprüfen
	02	Berechtigung mit D12 Permission überprüfen
	03	Löschen bestätigen
	04	D11 Folder Datensatz löschen
	05	F11 Ordner anschauen ausführen
Erweiterungen	04a	F24 Ordner löschen auf alle Kinder ausführen
Alternativen	03a	Löschen abbrechen - F11 Ordner anschauen ausführen

F21 Dokument anschauen		
Geschäftsprozess	**Anleitung anschauen**	
Ziel	D11 Folder erfolgreich gelöscht	
Vorbedingung	D11 Folder existiert	
Nachbedingung Erfolg	D11 Folder Datensatz und alle Kinder sind gelöscht	
Nachbedingung Fehlschlag	D02 User hat keine Berechtigung, Fehler beim Löschen der Daten	
Akteur	Gast, Benutzer	
Auslösendes Ereignis	Interaktion der Benutzeroberfläche	
Beschreibung	01	D01 Session überprüfen
	02	Berechtigung mit D12 Permission überprüfen
	03	D21 Document Daten laden
	04	Seite darstellen
Erweiterungen		
Alternativen	01a	F01 Login ausführen

F22 Dokument erstellen		
Geschäftsprozess	**Dokument erstellen**	
Ziel	D13 Document erfolgreich erstellt	
Vorbedingung	D11 Folder existiert	
Nachbedingung Erfolg	D13 Document existiert	
Nachbedingung Fehlschlag	D02 User hat keine Berechtigung	
Akteur	Benutzer	
Auslösendes Ereignis	Interaktion der Benutzeroberfläche	
Beschreibung	01	D01 Session überprüfen
	02	Berechtigung mit D12 Permission überprüfen
	03	Formular ausfüllen
	04	D13 Document Datensatz erstellen
	05	F21 Dokument anschauen ausführen
Erweiterungen		
Alternativen		

F23 Dokument verwalten		
Geschäftsprozess	**Dokument verwalten**	
Ziel	D13 Document erfolgreich erstellt	
Vorbedingung	D13 Document existiert	
Nachbedingung Erfolg	D13 Document wird gespeichert	
Nachbedingung Fehlschlag	D02 User hat keine Berechtigung, Fehler beim Speichern der Daten	
Akteur	Benutzer	
Auslösendes Ereignis	Interaktion der Benutzeroberfläche	
Beschreibung	01	D01 Session überprüfen
	02	Berechtigung mit D12 Permission überprüfen
	03	Änderungen Eingeben
	04	D13 Document Datensatz speichern
	05	F21 Dokument anschauen ausführen
Erweiterungen		
Alternativen		

F24 Dokument löschen		
Geschäftsprozess	**Dokument löschen**	
Ziel	D11 Folder mit allen D13 Document laden	
Vorbedingung	D11 Folder existiert und besitzt D13 Document	
Nachbedingung Erfolg	UI geladen	
Nachbedingung Fehlschlag	D02 User hat keine Berechtigung	
Akteur	Benutzer	
Auslösendes Ereignis	Interaktion der Benutzeroberfläche	
Beschreibung	01	D01 Session überprüfen
	02	Berechtigung mit D12 Permission überprüfen
	03	Löschen bestätigen
	04	D13 Dokument Datensatz löschen
	05	F21 Dokument anschauen ausführen
Erweiterungen		
Alternativen	03a	Löschen abbrechen – F21 Dokument anschauen ausführen

F31 Benutzer verwalten		
Geschäftsprozess	**Benutzer verwalten**	
Ziel	D02 User geändert	
Vorbedingung	D02 User existiert	
Nachbedingung Erfolg	D02 User wurde erfolgreich gespeichert	
Nachbedingung Fehlschlag	D02 User hat keine Berechtigung	
Akteur	Benutzer	
Auslösendes Ereignis	Interaktion der Benutzeroberfläche	
Beschreibung	01	D01 Session überprüfen
	02	Änderungen eingeben
	03	Speicher bestätigen
	04	D02 User Datensatz speichern
Erweiterungen		
Alternativen	03a	Änderung verwerfen

F31 Benutzer löschen		
Geschäftsprozess	**Benutzer löschen**	
Ziel	D02 User gelöscht	
Vorbedingung	D02 User existiert	
Nachbedingung Erfolg	D02 User wurde erfolgreich gelöscht	
Nachbedingung Fehlschlag	D02 User hat keine Berechtigung	
Akteur	Benutzer	
Auslösendes Ereignis	Interaktion der Benutzeroberfläche	
Beschreibung	01	D01 Session überprüfen
	02	Löschung bestätigt
	03	F14 Bereich löschen (Benutzer Bereiche)
	04	D02 User Datensatz löschen
Erweiterungen		
Alternativen	03a	Löschen abbrechen

4.2 Produktdaten

D01 Session		
Produktdaten	**Session**	
Enthaltet **Wird nicht serialisiert.**	Token	String
	Language	String(2)
	User	D02 User
	Release	DateTime

D02 User		
Produktdaten	**User**	
Enthaltet	Username	String
	Password	String
	Email	String

D11 Folder		
Produktdaten	**Space**	
Enthaltet	ID	UUID4
	Name	String
	Parent	D11 Folder
	Owner	D02 User

D12 Permissions		
Produktdaten	**Permissions**	
Enthaltet	ID	UUID4
	Space	D11 Folder
	User	D02 User
	Permission	String

D13 Document		
Produktdaten	**Document**	
Enthaltet	ID	UUID4
	Space	D11 Folder
	SectionList	ArrayList<D14 Sections>
	lastChange	DateTime
	lastChangeUser	D02 User

D14 Section		
Produktdaten	**Section**	
Enthaltet	ID	Int
	Space	D13 Document
	Titel	String
	Content	String
	lastChange	DateTime
	lastChangeUser	D02 User

5 Qualitätsanforderungen

Funktionalität	Nicht relevant	Normal	Gut	Sehr gut
Angemessenheit		X		
Richtigkeit		X		
Interoperabilität			X	
Ordnungsmässigkeit				X
Sicherheit			X	

Zuverlässigkeit	Nicht relevant	Normal	Gut	Sehr gut
Reife		X		
Fehlertoleranz			X	
Wiederherstellbarkeit				X

Benutzbarkeit	Nicht relevant	Normal	Gut	Sehr gut
Verständlichkeit				X
Erlernbarkeit			X	
Bedienbarkeit			X	

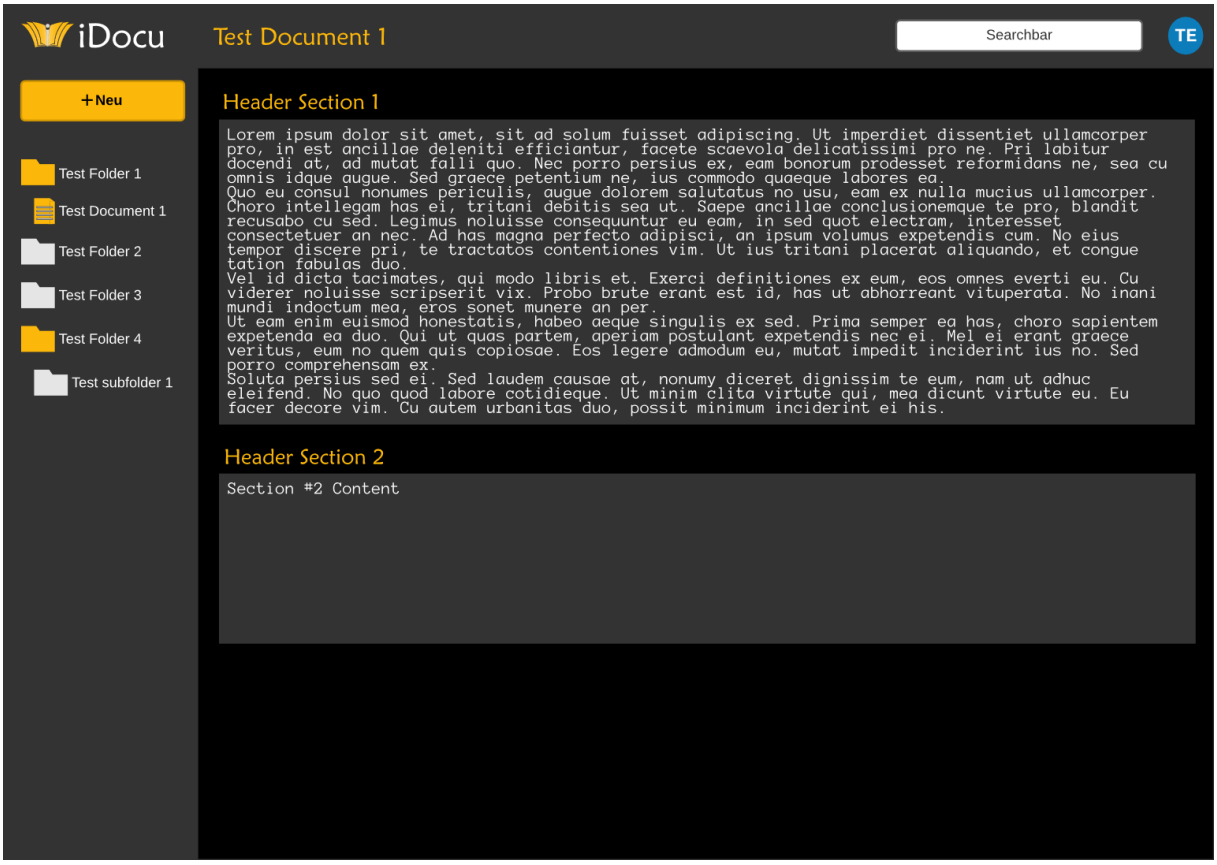
Effizienz	Nicht relevant	Normal	Gut	Sehr gut
Zeitverhalten		X		
Verbrauchsverhalten		X		

Änderbarkeit	Nicht relevant	Normal	Gut	Sehr gut
Analysierbarkeit		X		
Modifizierbarkeit			X	
Stabilität				X
Prüfbarkeit		X		

Übertragbarkeit	Nicht relevant	Normal	Gut	Sehr gut
Anpassbarkeit			X	
Installierbarkeit	X			
Konformität			X	
Austauschbarkeit	X			

6 Benutzeroberflächen

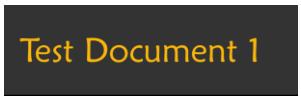
6.1 Übersicht



So soll die Webseite aussehen, wenn der Benutzer sich eingeloggt hat und ein Dokument geöffnet ist.

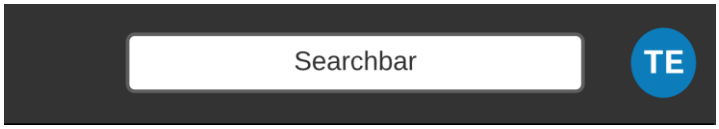
6.2 Header

Auf dem Header soll auf der Linken Seite das Logo mit Schriftzug dargestellt werden. Klickt man auf das Logo oder Schriftzug wird die Startseite geöffnet.



Rechts vom Logo und Schriftzug kommt der Titel der aktuellen Seite hin.

Auf der rechten Seite des Headers werden zusätzliche Tools aufgelistet. Hier zu sehen die Searchbar,



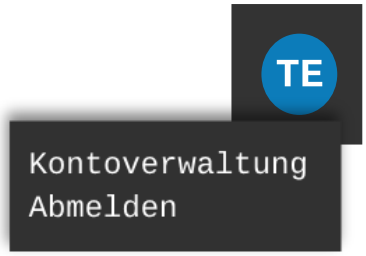
welche zum Suchen verwendet wird. Das Account Logo wird Rechts von der Searchbar dargestellt.

6.3 Benutzer

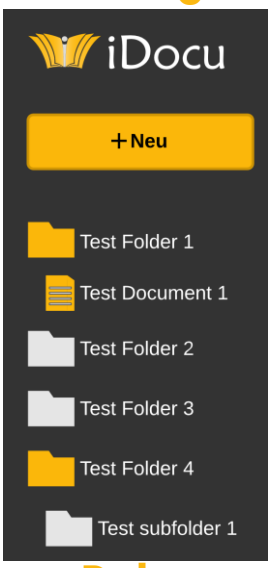
Wird auf das Account Logo geklickt öffnet sich das Account-Menü.

Bei Kontoverwaltung wird der aktuelle `D02 User` geöffnet.

Bei Abmelden wird die aktuelle `D01 Session` geschlossen.



6.4 Navigation



Unterhalb des Logobereiches wird der Button "+ Neu" positioniert. Wird auf den Button geklickt öffnet sich ein Popup welches eine Auswahl zwischen Dokument und Ordner gibt.

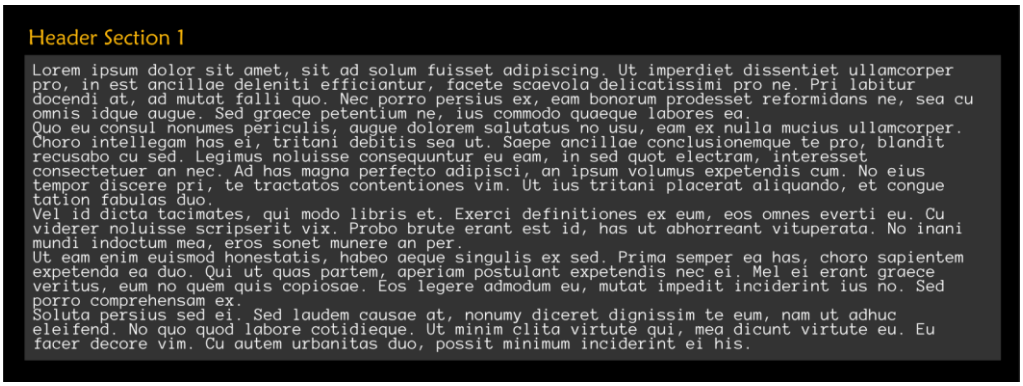
Unterhalb des Buttons "+ Neu" wird die Ordnerstruktur aufgelistet. Die Ordner welche aktiv (aufgeklappt) sind, werden farblich markiert. Jeder Ordner kann Dokumente oder Ordner enthalten.

Es können mehrere Ordner gleichzeitig geöffnet sein.

Der Ordner kann mit einem Klick geöffnet werden.

Mit einem klick auf das Dokument, wird das Dokument in der Mitte geöffnet. Das Dokument welches aktiv ist, wird auch farblich markiert.

6.5 Dokument

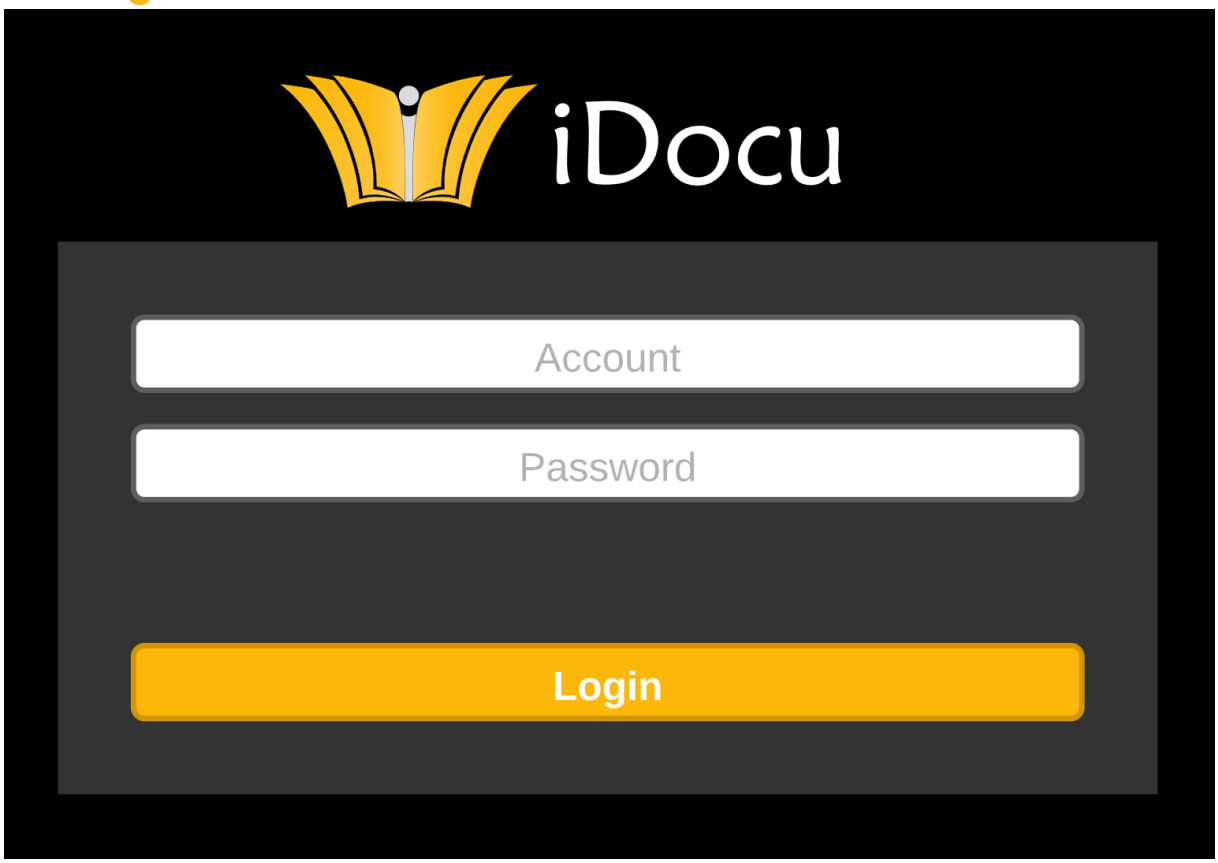


Jedes Dokument enthält mindesten eine `D14 Section`, jede `D14 Section` enthält einen eigenen Titel.

Hat man die benötigten Rechte, wird oberhalb der ersten `D14 Section` der Button "+ Neu" hinzugefügt.

Jede `D14 Section` muss einzeln überarbeitet werden.

6.6 Login



So sieht die Webseite aus, wenn ein Login verlangt wird. Kein Header oder sonstiges.

Mit dem Klick auf den Login Button wird `F01 Login` ausgelöst.

Die Fehlermeldung wird zwischen den Eingabefeldern und dem Login Button dargestellt.

7 Nicht funktionale Anforderungen

- Das System soll während 5 Jahren einsetzbar sein
- Die Software soll über die neusten Webbrowser erreicht werden können
- Die Dialog- und Reaktionszeit dürfen maximal eine Sekunde betragen

8 Techische Produktumgebung

8.1 Software

8.1.1 Anwender

Betriebssystem: Betriebssystem unabhängig

Installierte Software: Webbrowser (Chrome 42+, Firefox 39+, Edge 14+, Opera 29+, Safari 10.3+, Android 92+, Opera Mobile 64+, Chrome for Android 92+, Firefox for Android 90+, UC Browser for Android 12+, Samsung Internet 4+, QQ Browser 10+, Baidu Browser 7.12+, KaiOS Browser 2.5+)

8.1.2 Backend Server

Betriebssystem: Debian 10 oder höher, oder Ubuntu 20.04 LTS oder höher

Installierte Software: Node.js, PostgreSQL, Letsencrypt

8.1.3 Frontend Server

Betriebssystem: Debian 10 oder höher, oder Ubuntu 20.04 LTS oder höher

Installierte Software: Node.js, Letsencrypt

8.2 Hardware

8.2.1 Anwender

4GB Arbeitsspeicher, 2.6GHz Prozessor (1 Kern)

8.2.2 Backend Server

2vCPU, 4GB Arbeitsspeicher, mind. 80GB Speicher

8.2.3 Frontend Server

2vCPU, 4GB Arbeitsspeicher, mind. 20GB Speicher

9 Deployment

Die Software soll auf zwei verschiedene Server deployed werden. Dabei sollen Backend und Frontend physisch voneinander getrennt sein. Beide Instanzen sollen bei Hetzner.com auf Virtuellen Maschinen gehostet werden.

9.1 Backend

Auf dem Backend Server läuft die Node.js Backend Instanz inklusive Datenbank. Per Letsencrypt wird das SSL Zertifikat gehandhabt.

9.1.1 SSL Zertifikat

Das SSL Zertifikat wird von Letsencrypt verwaltet. Die Einbindung in die Applikation muss Manuell gemacht werden.

9.1.2 Ports

Die Backend Applikation läuft auf Port 80 (http) und Port 443 (https), diese Ports müssen beide von aussen erreichbar sein (Firewall Freigabe)

PostgreSQL läuft auf dem Standard-Port 5432, dieser Port darf von aussen **nicht** erreichbar sein.

9.1.3 Domain

Das Backend erhält die Domain "api.idocu.app".

9.2 Frontend

Auf dem Frontend Server läuft eine Node.js Applikation, welche das Frontend der Webapplikation betreibt.

9.2.1 SSL Zertifikat

Das SSL Zertifikat wird von Letsencrypt verwaltet. Die Einbindung in die Applikation muss Manuell gemacht werden.

9.2.2 Ports

Die Frontend Applikation läuft auf Port 80 (http) und Port 443 (https), diese Ports müssen beide von aussen erreichbar sein (Firewall Freigabe)

9.2.3 Domain

Das Frontend erhält die Domain "idocu.app".

10 Unterschrift

Die Unterschreibende Personen versichern, dass die in diesem Dokument aufgestellten Anforderungen das zu entwickelnde System vollständig beschreiben. Es sind zum Zeitpunkt der Unterschriftsleistung keine weiteren Anforderungen bekannt. Die nicht in diesem Dokument beschrieben wurden.

Es gelten keinerlei Anforderungen, die in weiteren Dokumenten beschrieben werden, ausser diese detaillierten die im Pflichtenheft beschriebenen Anforderungen. (z.B. bezüglich des Aufbaus)

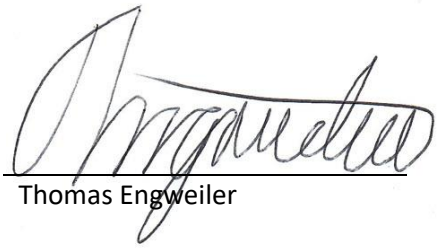
Zusätzliche Anforderungen oder Änderungen an den bestehenden Anforderungen bedürfen der Schriftform (Änderungsantrag).

Diplomlehrer

Olten, 13.09.2021
Ort, Datum

Benjamin Bäni

Projektleiter



Safenwil, 13.09.2021
Ort, Datum

Thomas Engweiler

iDocu Verifikations Formular

10	18:57	Geschlossen
Antworten	Durchschnittliche Zeit für das Ausfüllen	Status

1. Kunden können sich auf der Webseite registrieren und anmelden

Insights

Ja, Ein Benutzer mit E-Mail-Ad...	10
Nein, ich kann mich trotz Regi...	0

2. Der Kunde kann mindestens einen privaten Ordner erstellen

Insights

Der Ordner ist erstellt und "Gä...	9
Das Erstellen eines Ordners ha...	0
Obwohl ich keine Freigabe ein...	1

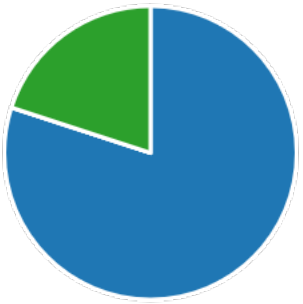
3. Jeder Kunde kann seine eigenen Ordner verwalten, inklusive Berechtigung

Ich konnte meinen Ordner um...	10
Ich konnte meinen Ordner ver...	8
Ich konnte die Berechtigunge...	9
Keine der möglichen Antworten	0

4. Der Kunde kann mindestens einen öffentlich zugänglichen Ordner erstellen

Insights

Die Freigabe für "Gäste" konnt...	8
Ich konnte die Berechtigung f...	0
Ich konnte zwar die Berechtig...	2



5. Einem Ordner können Dokumente hinzugefügt werden

Insights

Ich konnte in meinem Ordner,...	10
Das funktioniert leider nicht	0

6. Bilder können hochgeladen und eingefügt werden können

Insights

Ich konnte ein Bild einer Sekti...	9
Das funktioniert leider nicht	1

7. Der Text kann formatiert werden

Mein Text konnte ich "Fett" for...	10
Mein Text konnte ich "Kursiv" f...	10
Ich konnte die Farbe meines T...	9
Mein Text konnte ich untersch...	4
Keine der möglichen Antworten	0

8. Der Speicherplatz den der Kunde zur Verfügung hat ist eingegrenzt

Insights

Mein Speicherplatz ist auf 10...	10
Mein Speicherplatz ist nicht ei...	0
Obwohl mein Speicherplatz a...	0

9. Die Benutzeroberfläche soll sich am Windows Explorer (Ordner Ansicht) orientieren

Insights

10

Antworten

Neueste Antworten

"*Die Ordner Auflistung auf der Linken seite sieht ähnlich aus.*"

"*Ja es sieht ähnlich aus*"

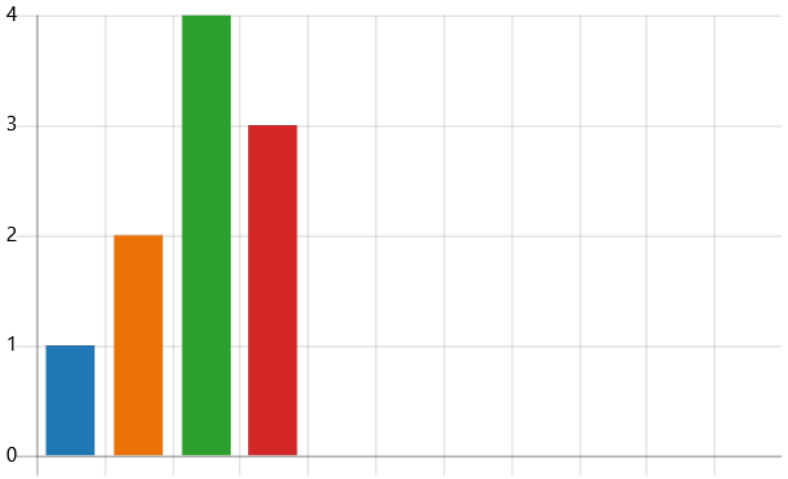
"*Mehr oder weniger. (tendenz zum mehr)*"

2 Befragten (**20**%) antworteten **Windows Explorer** für diese Frage.

10. Zu wie viel Prozent orientiert sich die Benutzeroberfläche dem Windows Explorer (Ordner Ansicht)?

Insights

~100%	1
~90%	2
~80%	4
~70%	3
~60%	0
~50%	0
~40%	0
~30%	0
~20%	0
~10%	0
~0%	0



11. Die App soll mehrsprachig entwickelt werden

Die Sprache "enUS" (Englisch) ...	10
Die Sprache "deDE" (Deutsch) ...	9
die Sprache "frFR" (Französisc...	1
die Sprache "ruRU" (Russisch) ...	1
Keine der angegebenen Mögli...	0

12. Dokumente können gelöscht werden

Insights

Ich konnte mein erstelltes Dok...	10
Das funktioniert leider nicht	0

13. Ordner können gelöscht werden

Insights

Ich konnte meinen erstellten ...	10
Das funktioniert leider nicht	0

14. Kunden können gelöscht werden

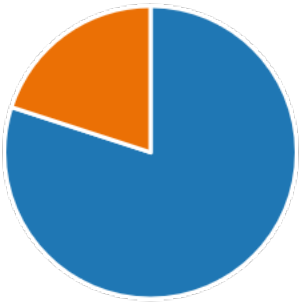
Insights

Ich konnte mein Konto lösche...	10
Das funktioniert leider nicht	0

15. Siehst du einen Verwendungszweck in deinem Umfeld

Insights

Ja	8
Nein	2



16. Wie gefällt dir das WebApp?

Insights

10

Antworten

★★★★☆

Durchschnittliche Bewertung 4.00

17. Dein Feedback zu meiner Arbeit

Insights

7

Antworten

Neueste Antworten

"Das Programm gefällt mir, ich könnte mir vorstellen es zu nutzen."

"Hilfefunktion wären wünschenswert."

2 Befragten (**29**%) antworteten **Zweck** für diese Frage.


ersten Benutzung
Dokumenten Ablage
Strukutiertere Dokumente
erneuten Setzen
online dokumentations Tool
grösseren Rahmen
Möglichkeit
Allem
Zweck
User
gute Lösung
flachere Strukturen
Basis-Funktionalität
umständliche Bedienung
Art abgespecktes online Office
10 MB Grenze
ansprechendes Design
meisten Funktionen
privaten Rahmen
existierende Dokumente