ulm university universität
uulm

Universität Ulm | 89069 Ulm | Germany

Fakultät für Ingenieurwissenschaften,
Informatik und Psychologie
Institut für Medieninformatik
Forschungsgruppe Visual Computing

MedInA: Ein Informationssystem zur Archivierung von Daten aus bildgebenden Verfahren der Medizin

Bachelor Thesis in Medieninformatik an der Universität Ulm

Vorgelegt von:
Robin Stüdle
robin.stuedle@uni-ulm.de

Gutachter:
Prof. Dr. Timo Ropinski

Betreuer:
Christian van Onzenoodt

2018

Fassung vom 30. November 2018

Satz: PDF-LaTeX 2$_\varepsilon$

Zusammenfassung

Diese Arbeit beschäftigt sich mit der Entwicklung einer Benutzerschnittstelle zur Archivierung und Verwaltung medizinischer Forschungsdaten. Durch moderne, bildgebende Untersuchungsverfahren wie beispielsweise der Magnetresonanztomographie, kommt es in der medizinischen Forschung zu einem enormen Datenaufkommen. Bestehende Archivierungssysteme bieten zur Verwaltung und Verarbeitung der den bildgebenden Verfahren entspringenden Daten und deren Metainformationen, keine effizient nutzbaren Methoden. Dieses Problem wird durch die Entwicklung einer auf die Daten zugeschnittenen Benutzerschnittstelle zur Archivierung der Datensätze gelöst. Zur Umsetzung dieser Benutzerschnittstelle wurde ein eigens für diesen Einsatzzweck nutzbares Konzept entwickelt und in einer Webanwendung implementiert. Die Konzeptentwicklung, einzelne Implementierungsdetails sowie die finale Anwendung, MedInA (**Med**izinisches **In**formations-**A**rchiv), werden in dieser Arbeit vorgestellt.

Inhaltsverzeichnis

1 Einleitung

Das erste Kapitel gibt eine kurze Einführung in die Thematik dieser Arbeit. Dabei wird in Abschnitt 1.1 Anfangs die Problemstellung, die Zielsetzung und ein Ansatz zur gegebenen Thematik gegeben, bevor in Abschnitt 1.2 die genaue Struktur der Arbeit vorgestellt wird.

1.1 Problem, Ziel und Ansatz

Bei der Computertomographie, der Magnetresonanztomographie sowie weiteren bildgebenden Verfahren fallen große Mengen an Daten an, welche dazu dienen, die genauen Vorgänge innerhalb des durchleuchteten Objekts beziehungsweise Subjekts zu erkennen. Um diese Daten zu archivieren werden bisher einfache Ordnerstrukturen verwendet, in welchen die Daten chronologisch sortiert abgelegt werden. Dies erschwert allerdings die Suche nach bestehenden Daten, insbesondere dann, wenn der genau Zeitpunkt der Akquirierung nicht bekannt ist. Muss beispielsweise ein älterer Datensatz mit einem neueren verglichen werden, muss der alte Datensatz manuell unter den vielen existierenden herausgesucht werden, um den Vergleich durchführen zu können.

Dieses zeitintensive Verfahren erschwert die Arbeit der forschenden Personen und soll mit einem einfacheren und effizienteren ersetzt werden. Das genaue Ziel des gesamten Projektes ist es, eine Anwendung zu entwickeln, welche die Datensätze verwaltet, einen schnellen Zugriff auf sie ermöglicht und dabei einen effizienten Arbeitsfluss gewährleistet.

Zusätzlich zu den bei der Messung anfallenden Rohdaten der medizinischen Geräte, beinhalten die zu archivierenden Datensätze weitere Metadaten, die den Datensatz näher beschreiben. Bei diesen Metadaten handelte es sich sich beispielsweise um Eigenschaften der Messung, wie die Dicke der abgetasteten Schichten oder die Auflösung des Datensatzes. Weiter sind Metainformationen des gescannten Subjekts enthalten wie beispielsweise dem Geschlecht, der Spezies und welche Stelle des Subjekts gescannt wurde. Aufgrund des Detaillierungsgrades der Daten, stellen Anforderungen an die Anonymität der Daten ein Problem dar, da es theoretisch möglich ist einzelne Datensätze bestimmten Subjekten (unter Umständen Personen) zuzuordnen.

Diese Arbeit befasst sich speziell mit der Visualisierung und Organisation der statischen Metadaten, um es in Zukunft zu ermöglichen, die Datensätze nicht nur nach dem

Datum, sondern auch nach weiteren Metadaten zu Sortieren beziehungsweise nach diesen zu Durchsuchen.

1.2 Aufbau der Arbeit

Anfangs in Kapitel 2 werden zu diesem Projekt verwandte Arbeiten vorgestellt, bevor in Kapitel 3 einige für den Aufbau der Arbeit wichtige Grundlagen der Mensch-Maschine-Interaktion präsentiert werden. Nach der Vorstellung der Grundlagen, beschäftigt sich Kapitel 4 mit dem Interaktionskonzept des Systems woraufhin in Kapitel 5 genauer auf die Implementierung und deren Teilaspekte eingegangen wird. In Kapitel 6 wird das finale System anschließend detailliert beschrieben, bevor in Kapitel 7 ein Fazit und weiterer Ausblick zu dieser Arbeit gegeben wird.

2 Verwandte Arbeiten

In diesem Kapitel werden verschiedene zu diesem Projekt verwandte Arbeiten präsentiert. In Abschnitt 2.1 werden verschiedene verwandte Cloudspeicherdienste vorgestellt, bevor Abschnitt 2.2 auf weitere verwandte Arbeiten eingeht.

2.1 Cloudspeicherdienste

Der erste Abschnitt beschäftigt sich mit Cloudspeicherdiensten. Cloundspeicherdienste sind online bereitgestellte Infrastrukturen, auf die Dateien hoch- und wieder heruntergeladen werden können.

2.1.1 Google Drive

Google Drive[1] ist ein von *Google*[2] kostenlos zur Verfügung gestellter Cloudspeicherdienst. Neben der Möglichkeit Dateien hoch- und wieder herunterzuladen, können Benutzer die in der Cloud (deutsch: Wolke) gespeicherten Dateien auch mit anderen Benutzern teilen. Datei können nicht nur mit einem bestimmten Benutzer oder einer bestimmten Benutzergruppe geteilt werden, sondern können auch öffentlich gemacht und so über Suchmaschinen gefunden werden. Neben der Möglichkeit der Indizierung der Dateien über öffentliche Suchmaschinen, ist es möglich eigene Dateien zu durchsuchen und zu filtern. Weitergehend können bestimmte Dateitypen auch direkt innerhalb der Cloud bearbeitet werden, wozu beispielsweise sogenannte *Google Docs*[3] gehören. *Google Docs* ist ein Service mit dem es möglich ist Textdokumente zu erstellen. Neben den eben genannten *Google Docs* zur Textdokumenterstellung gibt es noch *Google Sheets, Slides* und *Forms* um Tabellen, Präsentationen und Fragebögen direkt innerhalb der *Google Drive* erstellen und bearbeiten zu können. Alle in *Google Drive* gespeicherten Dateien können in einer Liste oder einer Kachelansicht eingesehen werden, durch einen Klick auf eine Datei, werden zu der Datei verfügbarer Metadaten und sonstige Informationen geladen und angezeigt. *Google Drive* ist nicht nur als Webanwendung

[1] `Google Drive: https://drive.google.com/` - abgerufen am 22.11.2018
[2] `Google: https://www.google.com/` - abgerufen am 22.11.2018
[3] `Google Docs: https://docs.google.com/` - abgerufen am 22.11.2018

verfügbar, sondern bietet auch jeweils eine native Anwendung für Betriebssysteme wie beispielsweise *Windows*[4], *Android*[5] oder *iOS*[6].

2.1.2 ownCloud

ownCloud[7] ist eine kostenlose, quellenoffene[8] Alternative zu anderen Cloudspeicherdiensten. *ownCloud* muss dabei heruntergeladen und auf einem eigenem Server initialisiert und installiert werden und stellt somit sicher, dass alle in dieser Cloud gespeicherten Daten nicht von Dritten mitgelesen werden können, wie dies zum Beispiel theoretisch bei kostenlosen Cloudspeicherdiensten wie dem in Abschnitt 2.1.1 vorgestellten *Google Drive* möglich ist. Erst nach einer Installation *ownClouds* auf einem eigenen Server beziehungsweise dem Beitritt auf einen durch Vertraute geführten Server, ist es möglich Dateien hoch- beziehungsweise herunterzuladen. Auch *ownCloud* bietet die Möglichkeit Dateien lokal zu durchsuchen und mit anderen zu teilen, indem diese direkt an weitere Benutzer freigegeben oder für ganze Gruppen zur Verfügung gestellt werden. *ownCloud* bietet wie auch andere Cloudspeicherdienste nicht nur eine Webanwendung zum Zugang auf die Daten an, sondern auch für verschiedene Betriebssysteme jeweils native Anwendungen wie beispielsweise für *Windows*, *Android* und *iOS*.

2.2 Weitere Arbeiten

Der zweite Abschnitt stellt neben den bereits in Abschnitt 2.1 vorgestellten Cloudspeicherdiensten weitere verwandte Arbeiten vor.

2.2.1 Google Photos

Google Fotos[9] ist ein kostenlos von *Google* zur Verfügung gestellter Online-Dienst in welchem Bilder und Videos gespeichert werden können, jedoch keine sonstigen Dateien, was *Google Fotos* zu dem in Abschnitt 2.1.1 vorgestellten *Google Drive* unterscheidet. Auf die dort gespeicherten Dateien kann mittels einer Webanwendung oder für die Betriebssysteme *Android* und *iOS* entwickelten Anwendungen zugegriffen werden. Die Besonderheit des Systems ist die automatische Sortierung der Bilder in Kategorien und Ordner beziehungsweise Alben, ohne dass beim Hochladen der jeweiligen Datei weitere Angaben gemacht werden müssen. Alle hochgeladenen Dateien werden dabei

[4] `Windows: https://www.microsoft.com/de-de/windows` - abgerufen am 22.11.2018
[5] `Android: https://www.android.com/` - abgerufen am 22.11.2018
[6] `iOS: https://www.apple.com/de/ios/` - abgerufen am 22.11.2018
[7] `ownCloud: https://owncloud.org/` - abgerufen am 22.11.2018
[8] `ownCloud GitHub: https://github.com/owncloud` - abgerufen am 22.11.2018
[9] `Google Fotos: https://photos.google.com/` - abgerufen am 22.11.2018

neben ihren direkt ablesbaren Metadaten wie beispielsweise dem Ort und der Zeit der Erstellung, auf ihren visuellen Inhalt durchsucht und beispielsweise Bilder von gleichen Personen oder Objekten in Alben sortiert. Somit ist es auch möglich mittels einer Suche nicht nur nach einem Erstellungsdatum sondern auch nach visuellem Inhalt wie beispielsweise *Essen* oder *Personen* zu suchen. Durch die vielen aus den hochgeladenen Bildern und Videos gewonnenen Informationen, können auch Suchen wie *Zeige mir alle Bilder des letzten Spanien Urlaubs* oder *Zeige mir alle Bilder auf denen ein Mensch zu sehen ist* mittels *Google Fotos* durchgeführt werden. Ein weiteres Beispiel für eine spezielle Funktionalität *Google Fotos*, ist die Gesichtserkennung des Services und die anschließende Sortierung und Einordnung aller Bilder einer Person in ein entsprechendes Album.

2.2.2 Github

GitHub[10] ist ein kostenlos nutzbarer Online-Dienst von *Microsoft*[11], der es ermöglicht Dateien online in sogenannten Repositories (deutsch: Behälter) zu speichern und zu versionieren. Aufgrund der direkten Verbindung des online verfügbaren Speicherplatzes mit dem Versionsverwaltungssystem *Git*[12], fungiert *GitHub* primär als Cloudspeicherdienst für Softwareprojekte. Diese Repositories können entweder öffentlich oder nur für bestimmten Personen sichtbar gemacht werden und dementsprechend falls öffentlich über Internet-Suchmaschinen gefunden werden. *GitHub* bietet des Weiteren eine erweiterte Suche zur Suche nach *GitHub* intern erstellten Repositories und Dateien an, mit welcher es möglich ist beispielsweise nach Wertebereichen oder einzelnen Stichwörtern in speziellen Feldern einzelner Metadaten zu suchen. Mittels einer Volltext-Suche ist es weitergehend möglich, den gesamten auf *GitHub* verfügbaren Quelltext zu durchsuchen. Abbildung 2.1 zeigt einen Ausschnitt der eben genannten erweiterten Suchfunktion.

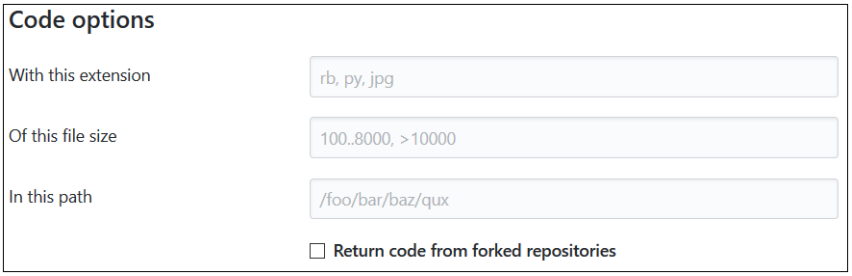


Abbildung 2.1: Diese Abbildung zeigt einen Ausschnitt der Funktion *Erweiterte Suche* des Online-Dienstes *GitHub* (vgl. Abschnitt 2.2.2).

[10] `GitHub: https://github.com/` - abgerufen am 22.11.2018
[11] `Microsoft: https://www.microsoft.com/` - abgerufen am 22.11.2018
[12] `Git: https://git-scm.com/` - abgerufen am 22.11.2018

3 Grundlagen der Mensch-Maschine-Interaktion

Das folgende Kapitel beschäftigt sich mit einigen Grundlagen der Mensch-Maschine-Interaktion, welche während der Entwicklung des Systems zum Einsatz kamen. Dabei wird zuerst auf die Grundsätze der Dialoggestaltung eingegangen und anschließend einige Gestaltgesetze vorgestellt.

3.1 Grundsätze der Dialoggestaltung

Der erste Abschnitt beschäftigt sich mit den sieben in der DIN EN ISO 9241 Teil 110 [11] definierten Grundsätzen der Dialoggestaltung und definiert diese kurz in einer eigenen Sortierung beziehungsweise Auflistung der Bestandteile.

3.1.1 Aufgabenangemessenheit

Dieser erste Unterpunkt befasst sich mit der Aufgabenangemessenheit eines Dialoges, welche als eigener Grundsatz in der Norm [11] genannt wird. Ein Dialog gilt laut der eben genannten Norm als der Aufgabe angemessen, sobald er den Benutzer dahingehend unterstützt, seine jeweilige Aufgabe effizient zu erledigen. Das bedeutet, dass der Dialog den Benutzer nicht mit bei der Aufgabenerledigung hinderlichen, weiteren Funktionen ablenkt, sondern seine Kernfunktion auf die effiziente Erledigung der jeweiligen Aufgabe beschränkt. Ein Beispiel hierfür wäre die Funktion des automatischen Formularausfüllens von Passwortmanagern für Internetbrowser.

3.1.2 Orientierung

Die Orientierung als solches wird in der Norm nicht genannt, fasst in dieser Arbeit allerdings die folgenden Grundsätze zusammen:

Selbstbeschreibungsfähigkeit

Die Eigenschaft der Selbstbeschreibungsfähigkeit eines Dialoges bedeutet, dass einem Benutzer zu jedem Zeitpunkt klar beschrieben wird, in welchem Status er

sich befindet und was seine Möglichkeiten sind beziehungsweise wie diese auszuführen sind, ohne dies in vielen Hilfsdialogen anzeigen zu müssen. Ein Beispiel für die Umsetzung der Selbstbeschreibungsfähigkeit sind unter anderem Rückmeldungen nach getätigten Aktionen, Hilfeoptionen durch beispielsweise kleine Info-Buttons, eine klare Abhebung der Aktionselemente zu Elementen mit denen keine Interaktionen getätigt werden können sowie Anzeigen wie Ladebalken.

Lernförderlichkeit

Ein Dialog gilt als lernfördernd, wenn er den Benutzer beim Erlernen des Systems unterstützt, indem alle Interaktionsmöglichkeiten sowie bestimmte Arten von Funktionen des Systems leicht zu erlernenden Prinzipien folgen. Beispielsweise ist der Einsatz von Icons in Anwendungen lernfördernd, da der Benutzer diese wiedererkennen und das Interaktionselement somit mit einer bestimmten Funktion verknüpfen kann.

Erwartungskonformität

Den Grundsatz der Erwartungskonformität erfüllt ein Dialog, sobald er den Benutzer bei der Verwendung des Systems dahingehend unterstützt, dass er alle Interaktionselemente sowie bestimmte Arten von Informationen und Funktionen auf die immer selbe Art und Weise abwickelt beziehungsweise darstellt. Ein Beispiel für einen lernförderlichen Dialog stellt die gleiche Strukturierung und Funktionalität der Navigationselemente einer Navigation einer Anwendung dar.

Fehlertoleranz

Der Grundsatz der Fehlertoleranz für Dialoge beschreibt die Eigenschaft des Dialoges, auf eventuell auftretende Fehler angemessen zu reagieren und dem Benutzer die Möglichkeit der Korrektur anzubieten. Vertippt sich ein Benutzer beispielsweise bei einer Suche, ist ein Dialog fehlertolerant, wenn der falsch eingegebene Suchbegriff selbst nach der Ausführung der Suche in der Suchleiste bleibt, dass der Suchbegriff korrigiert und nicht erneut voll eingegeben werden muss.

3.1.3 Beeinflussung

Die Beeinflussung kommt wie auch schon Abschnitt 3.1.2 nicht als Solches in der Norm vor, fasst in dieser Arbeit allerdings die folgenden Grundsätze zusammen:

Steuerbarkeit

Ein Dialog gilt als steuerbar, wenn er dem Benutzer das Navigieren vor und zurück innerhalb des Systems ermöglicht. Dies kann über Icons, Buttons oder sonstige Schaltflächen und Bedienelemente geschehen. Ein Beispiel hierfür sind

die Interaktionselemente eines Internetbrowsers wie beispielsweise der Zurück-Button, der Schließen-Button oder auch der Einstellungs-Button.

Individualisierbarkeit
Der Grundsatz der Individualisierbarkeit beschreibt die Eigenschaft eines Dialoges, die Möglichkeit zu bieten, gewisse Teile eines Systems auf die individuellen Bedürfnisse des jeweiligen Benutzers zu verändern und anpassen zu können. Ein Beispiel hierfür ist die Möglichkeit Elemente auf einem Dashboard nach den eigenen Bedürfnisse anordnen oder Listen nach Spalten sortieren zu können.

3.2 Gestaltgesetze

Im zweiten Abschnitt werden drei wie von Dahm [10, S. 59ff] beschriebene grundlegende Gestaltgesetze vorgestellt und jeweils anhand einer Abbildung erläutert.

3.2.1 Gesetz der Ähnlichkeit

Das Gesetz der Ähnlichkeit besagt, dass Elemente mit gleicher beziehungsweise ähnlicher Form und Struktur als eine Einheit wahrgenommen werden. Dies gilt nicht nur für die Form und Struktur der Elemente, auch durch den Einsatz von gleicher oder ähnlicher Farbe ist der selbe Effekt zu erzielen. Zwei Beispiele hierfür sind im oberen Teil der Abbildung 3.1 zu sehen. In der linken Hälfte werden die Kreuze anhand ihrer jeweiligen Farbe als Einheit wahrgenommen. Die rechte Hälfte zeigt zwei verschiedene Formen gleicher Farbe, hier werden die verschiedenen Elemente mit der jeweils gleichen Form gruppiert.

3.2.2 Gesetz der Nähe

Das zweite hier vorgestellte Gestaltgesetz ist das Gesetz der Nähe. Das Gesetz der Nähe besagt, dass Elemente welche im Verhältnis zu anderen Elementen nahe aneinander platziert sind, als Einheit wahrgenommen werden. Im mittleren Feld der Abbildung 3.1 ist ein Beispiel für dieses Gesetz zu sehen. Die Punkte der linke Hälfte des mittleren Teils der Abbildung erscheinen als Spalten, wohingegen die Punkte der rechte Hälfte des mittleren Teils der Abbildung als Reihen wahrgenommen werden, was allein aufgrund der Positionierung (Nähe) geschieht.

3.2.3 Gesetz der Fortsetzung

Das letzte hier vorgestellte Gesetz ist das Gesetz der Fortsetzung. Es beschreibt das Phänomen, dass wir alle Elemente die als eine zusammengesetzte, fortgeführte

Linie dargestellt werden verfolgen, wobei die Form der Linie außer Acht gelassen werden kann. Dies gilt nicht nur bei fortgesetzten Linien, sondern auch bei teilweiser Verdeckung durch größere im Vordergrund stehende Elemente. Der untere Teil der Abbildung 3.1 zeigt ein Beispiel für diese genannten Eigenschaften des Gesetzes. In der linken Hälfte sind zwei Linien mit jeweils gleichförmigen und gleichfarbigen Elementen sich kreuzend angeordnet. Die zwei Linien können aufgrund ihrer vorhergehenden und weitergehenden Fortsetzung als solche wahrgenommen und die jeweiligen Elemente entsprechend zugeordnet werden. Die rechte Hälfte des unteren Teils der Abbildung zeigt die Eigenschaft, dass gleichmäßig fortgesetzte Elemente obwohl diese teilweise von einem größeren Element verdeckt werden, als Einheit wahrgenommen werden.

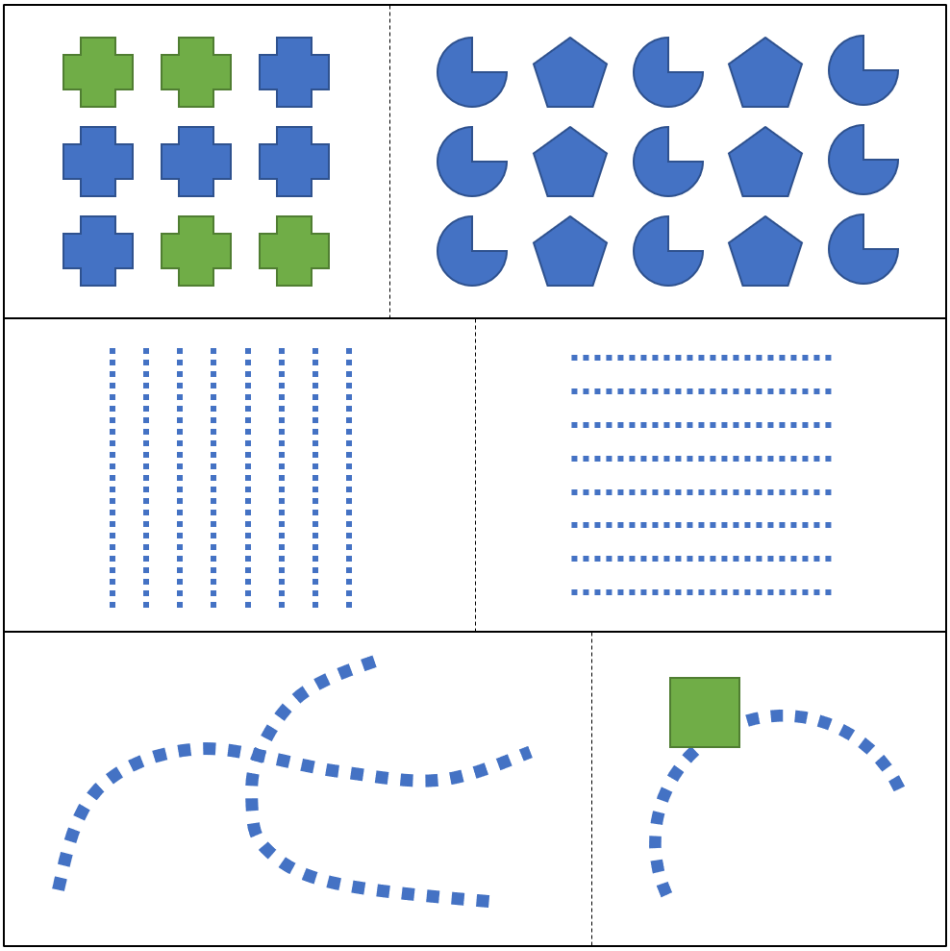


Abbildung 3.1: Diese Abbildung zeigt im oberen Teil zwei Beispiele für das Gesetz der Ähnlichkeit (vgl. Abschnitt 3.2.1), der mittlere Teil zeigt ein Beispiel für das Gesetz der Nähe (vgl. Abschnitt 3.2.2) und der unteren Teil zwei Beispiele für das Gesetz der Fortsetzung (vgl. Abschnitt 3.2.3).

4 Interaktionskonzept

Das folgende Kapitel befasst sich mit dem für dieses System entwickelten Interaktionskonzept. In Abschnitt 4.1 wird zuerst auf die genauen Anforderungen an das System eingegangen, bevor in Abschnitt 4.2 das Interaktionskonzept detailliert vorgestellt wird.

4.1 Anforderungen

In diesem Abschnitt werden die Anforderungen des Systems mithilfe sogenannter User Stories (deutsch: Anwendererzählungen) definiert. Eine User Story beschreibt eine Anforderung an eine Software in maximal zwei Sätzen. Für die genauere Definition einer User Story wird auf die Arbeit von Cohn [9] verwiesen. Nachfolgend werden zuerst in Abschnitt 4.1.1 die für das System aus Auftraggebersicht wichtigen User Stories definiert, woraufhin die aus Benutzersicht wichtigen User Stories in Abschnitt 4.1.2 vorgestellt werden.

4.1.1 Auftraggebersicht

In diesem Abschnitt werden alle aus Auftraggebersicht wichtigen Anforderungen vorgestellt.

Gruppenverwaltung
: Als Auftraggeber möchte ich Benutzer in eine oder mehrere Gruppen einteilen können, um das System für viele Benutzergruppen nutzbar zu machen.

Datenintegrität
: Als Auftraggeber möchte ich, dass Benutzer nur die für ihre jeweilige Gruppe sichtbaren Dateien sehen können, um die Integrität der Daten zu gewährleisten.

Benutzerverwaltung
: Als Auftraggeber möchte ich Benutzer anlegen und wieder löschen können, um neuen Mitarbeitern Zugriff auf die Daten zu gewähren und ehemaligen Mitarbeitern den Zugriff wieder zu entziehen. Außerdem soll bestimmt werden können, auf welche Datensätze ein Benutzer Zugriff hat.

Datenverwaltung

Als Auftraggeber möchte ich Dateien löschen können, um fälschlicherweise in das System geladene Dateien aussortieren zu können. Da es sich bei diesem System um ein Archivierungsdienst handelt, soll dies allerdings nur einem sehr kleinen, speziell berechtigtem Personenkreis möglich sein.

Sicherheit

Als Auftraggeber möchte ich, dass zu jedem Zeitpunkt der Benutzung des Systems durch einen Benutzer für die Sicherheit der Daten des Benutzers sowie die Sicherheit der Dateien gesorgt ist, um alle Daten sicher vor unbefugten Dritten abzuschirmen.

4.1.2 Benutzersicht

Dieser Abschnitt präsentiert alle aus Benutzersicht wichtigen User Stories für das System.

Anmeldung

Als Benutzer möchte ich mich im System anmelden können, um die verfügbaren Datensätze einzusehen.

Abmeldung

Als Benutzer möchte ich mich nachdem ich angemeldet bin auch wieder abmelden können, falls ich meinen Arbeitsplatz verlasse.

Navigation

Als Benutzer möchte ich von jeder Stelle im System leicht zu den anderen verfügbaren stellen wechseln können, um schnell mit dem System arbeiten zu können.

Listenansicht

Als Benutzer möchte ich alle verfügbaren Datensätze an einer zentralen Stelle einsehen können, damit ich einen guten Überblick über meine Datensätze habe.

Suche

Als Benutzer möchte ich alle Datensätze zentral durchsuchen können, um nicht in allen verfügbaren Datensätzen einzeln nach meiner gesuchten Datei suchen zu müssen.

Herunterladen

Als Benutzer möchte ich einen oder mehrere Datensätze herunterladen können, um sie lokal auf meinem Computer ansehen zu können.

Hochladen

Als Benutzer möchte ich Dateien ohne eine weitere Angabe von Details zu dieser Datei hochladen können, um sie später wieder finden zu können.

Datenkontrolle

Als Benutzer möchte ich nach dem Hochladen einer Datei überprüfen können ob die Informationen aus der Datei richtig ausgelesen wurden, um falsche Informationen zu Dateien vermeiden zu können.

Details

Als Benutzer möchte ich zu jedem Datensatz alle verfügbaren Daten einsehen können, ohne diesen zuvor wieder herunterladen zu müssen, um schnell nach dem richtigen Datensatz suchen zu können.

Bearbeiten

Als Benutzer möchte ich, sollten mir bei der Benutzung des Systems Fehler oder Inkonsistenzen in Datensätzen auffallen, diese bearbeiten können, um alle Dateien fehlerfrei verwenden zu können.

Sortieren

Als Benutzer möchte ich die mir momentan angezeigten Dateien nach ihren Metadaten sortieren können, um einen guten Überblick über diese zu erlangen.

Neue Dateien

Als Benutzer möchte ich ohne großen Aufwand alle zuletzt hinzugefügten Datensätze sehen und diese auch direkt von dort verwalten können, um aktuelle Datensätze schnell wiederfinden zu können.

Beabeitete Dateien

Als Benutzer möchte ich alle zuletzt bearbeiteten Dateien einsehen können und diese von dort direkt verwalten können, um aktuelle Datensätze schnell wiederfinden zu können.

Experten Suche

Als Benutzer möchte ich neben einer einfachen suche eine erweiterte Suche, um Datensätze nach mehreren Metadaten durchsuchen zu können, um gezielt einzelne Datensätze wiederfinden zu können.

Zugangsdatenverwaltung

Als Benutzer möchte ich mein Passwort ändern können, um immer ein aktuelles, sicheres Passwort zu haben.

4.2 Prototyping mit Wireframes

Dieser Abschnitt stellt basierend auf den in Abschnitt 4.1 vorgestellten Anforderungen an das System das entsprechende Interaktionskonzept mithilfe von sogenannten Wireframes (deutsch: Drahtgerüste) vor. Mit Wireframes lassen sich einfache Konzeptentwürfe einer Oberfläche darstellen, ohne dabei zu sehr auf die genaue Gestaltung einzugehen. Im Folgenden wird auf die einzelnen Komponenten eingegangen.

4.2.1 Anmeldung

Die Anmeldeseite ist eine der restlichen Anwendung vorgeschaltete Ansicht und besteht aus drei Elementen. Zwei Texteingabefeldern für Benutzername und Passwort sowie einem Button um die eingegebenen Daten abzuschicken. Grundlegend erfüllt die Anmeldeseite die in Abschnitt 4.1 vorgestellte User Story *Anmeldung*.

Um die in Abschnitt 3.1 vorgestellten Grundsätze der Dialoggestaltung einzuhalten, genauer die Aufgabenangemessenheit (vgl. Abschnitt 3.1.1), wird hier besonders darauf geachtet, keine weiteren Informationen als die für die Anmeldung nötigen (vgl. weiter oben im Text) anzuzeigen.

Weitergehend bietet das Passwort-Textfeld die Möglichkeit das eingegebene Passwort in Klartext anzusehen, indem auf das sich rechts im Textfeld befindliche Icon geklickt wird, was die Lernförderlichkeit 3.1.2 und Individualisierbarkeit (vgl. Abschnitt 3.1.3) verbessert. Auf diese Weise muss das Passwort nicht erneut eingegeben werden, falls der Benutzer sich bei der Eingabe nicht sicher über dessen Korrektheit ist, um die Fehlertoleranz (vgl. Abschnitt 3.1.2) des Systems bei der Anmeldung zu verbessern.

4.2.2 Navigation

Die Navigation des Systems erfüllt grundlegend die in Abschnitt 4.1 vorgestellte User Story *Navigation*. Sie besteht um den Grundsatz der Aufgabenangemessenheit zu erfüllen aus zwei Teilen. Einer im oberen Bereich der Anwendung angezeigten Navigationsleiste sowie einer Leiste im linken seitlichen Bereich (vgl. Abbildung 4.2) und ist neben den Interaktionsmöglichkeiten mit den Unterseiten, die Schnittstelle mit dem der Benutzer die Anwendung steuern (vgl. Abschnitt 3.1.3) kann. Außer auf der in Abschnitt 4.2.1 vorgestellten Anmeldeseite, sind beide Teile der Navigation auf jeder im Folgenden vorgestellten Ansicht sichtbar.

Obere Navigationsleiste Die obere Navigationsleiste ist in Abbildung 4.2 zu sehen und besteht aus einem länglichen Kasten, der wiederum aus drei Teilen besteht. Diese drei Teilelemente unterscheiden sich in der Form von einander, werden allerdings aufgrund des in Abschnitt 3.2.3 vorgestellten Gesetztes der Fortsetzung als eine Einheit wahrgenommen.

Auf der linken Seite der oberen Navigationsleiste ist der Schriftzug mit dem Titel des Systems *MoMAN* zu sehen. Dieser dient der Selbstbeschreibungsfähigkeit (vgl. Abschnitt 3.1.2) um dem Benutzer in jedem Moment mitzuteilen, dass er sich auf der richtigen Plattform befindet. Mit einem Klick auf den Schriftzug, landet er auf der Startseite der Anwendung. Dieses Verhalten ist sinnvoll, da sich der Logo Schriftzug mit der weiter unten im Text vorgestellten seitlichen Navigationsleiste in einer Linie befindet und so die Erwartungskonformität (vgl. Abschnitt 3.1.2) der Anwendung verbessert wird.

Die Suchleiste befindet sich im mittleren Teil der oberen Navigationsleiste. Sie durchsucht alle Datensätze auf einem fest definierten Eigenschaftsfeld und ermöglicht die Suche mit sogenannten Regulären Ausdrücken [4]. Die Suchleiste erfüllt die User Story *Suche*. Nach der Eingabe eines Suchbegriffs und dem Abschicken dieser Anfrage, leitet die Suchleiste von jeder Stelle im System aus auf die Listenansicht (vgl. Abschnitt 4.2.4) weiter.

Das letzte Element dieser Reihe ist das ganz rechts stehende, welches einen Gruß sowie weitere Interaktionsmöglichkeiten bietet. Die beiden Elemente werden aufgrund des in Abschnitt 3.2.2 vorgestellten Gesetzes der Nähe als Einheit wahrgenommen.

Der Gruß bestätigt dem Benutzer, dass er eingeloggt ist und trägt somit zur Selbstbeschreibung der Umgebung bei, wohingegen ein Klick auf das direkt neben dem Text stehende Chevron, zwei Interaktionsmöglichkeiten mit dem System anzeigt (vgl. Abschnitt 4.2). Dies sind zwei klickbare Felder, wovon das Eine zu den Einstellungen navigiert und das Andere den Benutzer ausloggt, womit dieses Element die User Story *Abmeldung* erfüllt.

Seitliche Navigationsleiste Den zweiten Teil der Navigation stellt die in den Abbildungen 4.1 und 4.2 ersichtliche seitliche Navigationsleiste dar. Sie besteht aus zwei verschiedenen Elementen. Einem sich durch seine Form abhebenden Button um zur Uploadansicht (vgl. Abschnitt 4.2.3) zu gelangen, sowie drei weiteren gleich aussehenden Schaltflächen mit verschiedenem Inhalt, um zu den weiteren Unterseiten Liste (vgl. Abschnitt 4.2.4), Zuletzt (vgl. Abschnitt 4.2.5) und Suche (vgl. Abschnitt 4.2.6) zu gelangen. Diese drei eben genannten Schaltflächen werden aufgrund ihrer gleichen Form durch das Gesetz der Ähnlichkeit (vgl. Abschnitt 3.2.1) sowie aufgrund ihrer Positionierung durch das Gesetz der Nähe als gleichwertige Elemente wahrgenommen. Dies führt zu einer verbesserten Lernförderlichkeit des Systems, da der Benutzer nach einem Klick auf eines der Elemente damit rechnen kann (Erwartungskonformität), dass die jeweils anderen Schaltflächen ihn an das dort angegebene Ziel navigieren. Ist eine der eben genannten Schaltflächen die aktuell aktive angezeigte Unterseite des Systems, wird die entsprechende Schaltfläche mit einem kleinen Balken am linken Rand hervorgehoben um dem Grundsatz der Selbstbeschreibungsfähigkeit zu entsprechen.

4.2.3 Uploadansicht

Die Uploadansicht besteht um den Grundsatz der Aufgabenangemessenheit nicht zu verletzen beim Aufrufen aus zwei Komponenten. Einem großen Feld, das aufgrund seiner Beschriftung dazu aufruft, auf selbiges zu klicken um Dateien in das System zu laden. Nachdem die Dateien ausgewählt wurden, kann der Benutzer durch einen Klick auf die zweite Komponente, den Upload Button, die aktuell ausgewählte Datei beziehungsweise die ausgewählten Dateien hochladen. Dies entspricht der Anforderung, die in der User Story *Hochladen* gestellt wurde. Nach dem erfolgreichen Hochladen der Dateien, wird direkt die in Abschnitt 4.2.9 vorgestellte Bearbeitungsansicht aufgerufen, um die User Story *Datenkontrolle* zu erfüllen.

4.2.4 Listenansicht

Die Hauptansicht des Systems stellt die in Abbildung 4.1 gezeigte Listenansicht dar, welche der User Story *Listenansicht* entspricht. In der linken oberen Ecke ist wie bei allen über die Navigationsleiste erreichbaren Unterseiten der Anwendung eine Überschrift zu lesen, welche in diesem Fall der Titel *List* ist. Dies trägt der Selbstbeschreibungsfähigkeit des Systems bei, indem es dem Benutzer in jedem Moment kommuniziert, auf welcher Unterseite er sich momentan befindet.

Die Tabelle stellt das Zentrale Element der Listenansicht dar. Sie ist in der Mitte des Bildschirms platziert (vgl. Abbildung 4.1) und streckt sich über nahezu den ganzen Bildschirm der Anwendung. Die Tabelle selber besteht aus einem Tabellenkopf und dem restlichen Tabelleninhalt.

Der Tabellenkopf hat ganz links eine Checkbox, mit der es möglich ist direkt alle Datensätze auf einmal auszuwählen. Neben der Checkbox werden die Namen der Felder der angezeigten Dateiinformationen innerhalb des Tabellenkopfs angezeigt. Um das System weitestgehend individualisierbar zu machen, können durch einen Klick auf die entsprechende Spalte im Tabellenkopf, alle angezeigten Daten nach dieser Spalte sortiert werden, was die User Story *Sortieren* erfüllt. Dies funktioniert analog für jede Spalte, was dem Grundsatz der Erwartungskonformität entspricht.

Der Tabelleninhalt besteht aus den einem Benutzer sichtbaren Datensätzen, wobei Benutzer wie in der User Story *Gruppenverwaltung* gefordert, einer oder mehreren Gruppen zugeteilt sind. Jeder Datensatz wird in einer eigenen Zeile angezeigt. Diese Zeilen sind durch einen Strich getrennt und werden neben der Trennung auch aufgrund der Anordnung der Daten nach dem Gesetz der Nähe als jeweils eine Einheit wahrgenommen. Im linken Teil jeder Zeile sind dabei drei klickbare Icons platziert. Ein Klick auf die Checkbox führt zur Öffnung der Schnellauswahlleiste, ein Klick auf das Stift-Icon öffnet die Bearbeitungsansicht (vgl. Abschnitt 4.2.9), ein Klick auf das Herunterladen-Icon

startet das Herunterladen der Datei und realisiert somit die User Story *Herunterladen*. Ein Klick auf die restliche Fläche der Zeile öffnet die Seiten-Detailansicht.

Neben der eben genannten Tabelle existiert die Schnellauswahlleiste, mit welcher mehrere Dateien auf einmal verwaltet werden können. Bei der Auswahl eines Datensatzes durch das klicken einer Checkbox eines Datensatzes aus der angezeigten Liste, wird diese Sichtbar. Sie liegt über der oberen Navigationsleiste (vgl. Abbildung 4.1) und bietet die Möglichkeit durch einen Klick auf die Checkbox ganz links oder den Schließen-Button ganz rechts die ausgewählten Dateien wieder Abzuwählen.

Zwei weitere Einträge der Schnellauswahlleiste sind einmal der Bearbeiten-Button sowie der Herunterladen-Button, welche um das System lernförderlich zu gestalten mit den selben Icons ausgestattet sind, wie die Buttons innerhalb der Zeilen der Tabelle. Im linken Teil der Schnellauswahlleiste, rechts neben der Checkbox, wird außerdem angezeigt wie viele Dateien aktuell ausgewählt sind, um den Benutzer über den aktuellen Zustand des Systems zu informieren.

Um auch innerhalb der Listenansicht die User Story *Details* zu erfüllen, gibt es als dritte Komponente die Seiten-Detailansicht. Sie kann entweder über einen Klick auf das Info-Icon das sich rechts oberhalb der Tabelle befindet oder durch einen Klick auf den entsprechenden Datensatz aus der Liste geöffnet werden (vgl. Abbildung 4.1). In ihr werden von oben nach unten alle zum jeweiligen Datensatz verfügbaren Daten in einer Tabelle angezeigt. Dabei werden die Namen der jeweiligen Daten fett geschrieben und die Werte dieser Daten mit normaler Schriftstärke dargestellt, womit aufgrund des Gesetzes der Ähnlichkeit diese zwei verschiedenen Typen von Informationen jeweils der richtigen Gruppe zugeordnet werden können, beziehungsweise zwischen Überschrift und Wert unterschieden werden kann.

Durch einen Klick auf das eben genannte Info-Icon sowie der innerhalb der Seiten-Detailansicht rechts oben und rechts unten angezeigten Schließen-Button, lässt sich die Detailansicht schließen.

Um zusätzlich eine redundante Steuerung des Systems zu gewährleisten, befinden sich innerhalb der Seiten-Detailansicht ein Bearbeiten- und ein Download-Button, was für möglichst kurze Wege bei der Benutzung des Systems sorgt. Alle genannten Buttons haben die gleiche Form und das gleiche Verhalten, um das System erwartungskonform und lernförderlich zu halten.

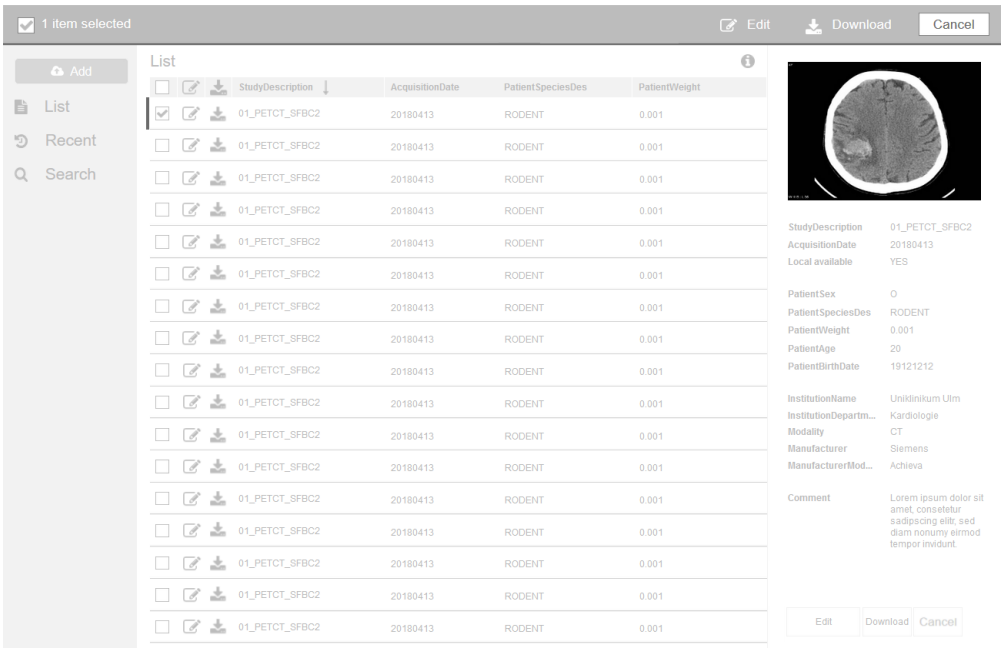


Abbildung 4.1: In dieser Abbildung ist ein sogenannter Wireframe der in Abschnitt 4.2.4 vorgestellten Listenansicht zu sehen. Auf dem Wireframe sind die Schnellauswahlleiste sowie die Seiten-Detailansicht geöffnet.

4.2.5 Zuletztansicht

Die Zuletztansicht stellt neben der Listenansicht eine weitere Möglichkeit dar, im System verfügbare Datensätze einzusehen und ist in Abbildung 4.2 zu sehen. Sie besitzt in der oberen linken Ecke eine Überschrift, um den Benutzer über den aktuellen Zustand des Systems zu informieren und besteht im Gegensatz zur Listenansicht, aus nicht einer, sondern zwei Tabellen und erfüllt die User Stories *Neue Dateien* und *Beabeitete Dateien*.

Die Zuletztansicht besteht aus einer Tabelle, welche in chronologisch absteigender Reihenfolge die zuletzt hinzugefügten Datensätze anzeigt sowie aus einer Tabelle, welche in absteigender Reihenfolge die zuletzt veränderten Datensätze auflistet. Die Tabellen sind jeweils wie die Tabelle der in Abschnitt 4.2.4 vorgestellten Listenansicht aufgebaut. Sie besitzen des Weiteren die gleiche optische Darstellung wie die Tabelle der Listenansicht, um die Wiedererkennung der möglichen Interaktionen mit der Tabelle zu fördern. Dies geschieht aufgrund des Gesetzes der Ähnlichkeit und bringt das System somit durch Konsistenz dazu, sich erwartungskonform zu verhalten.

Aufgrund der Anordnung zweier Tabellen nebeneinander, ist es nötig die Seiten-Detailansicht der Listenansicht durch ein passendes Pendant zu ersetzen. Dies wurde mittels der Detailansicht (vgl. Abschnitt 4.2.8) realisiert. Diese wird wie auch die Seiten-

Detailansicht der Listenansicht, durch einen Klick auf einen Eintrag der entsprechenden Tabelle geöffnet.

Durch den gleichen Aufbau der Tabellen der Zuletztansicht, wie die Tabelle der Listenansicht, besteht auch auf der Zuletztansicht die Möglichkeit der Mehrfachauswahl der Datensätze. Wird ein Datensatz durch das anklicken der Checkbox ausgewählt, öffnet sich die Schnellauswahlleiste, welche bis auf einen Eintrag gleich aufgebaut ist wie die Schnellauswahlleiste der Listenansicht.

Der Detail-Button ist als weiteres Interaktionselement in der Schnellauswahlleiste der Zuletztansicht gegeben. Dieser wurde in der Listenansicht ausgelassen, um das System Aufgabenangemessen zu gestalten, da hier die Seiten-Detailansicht eine einfachere Möglichkeit der Informationsdarstellung bietet. Durch einen Klick auf den eben genannten Detail-Button öffnet sich wie auch durch einen Klick auf einen Eintrag einer der Tabellen, die Detailansicht (vgl. Abschnitt 4.2.8) mit den zuvor ausgewählten Elementen. Somit unterstützt auch die Zuletztansicht die User Story *Details*.

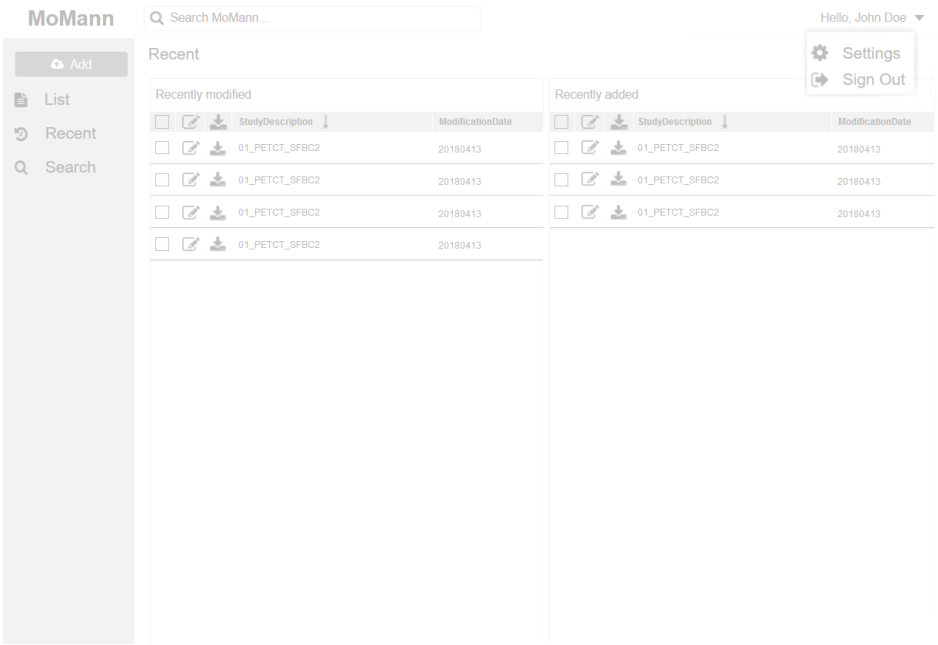


Abbildung 4.2: Diese Abbildung zeigt den Wireframe der in Abschnitt 4.2.5 vorgestellten Zuletztansicht. Des Weiteren ist oben rechts in der Abbildung das geöffnete Detailmenü der oberen Navigationsleiste (vgl. Abschnitt 4.2.2) zu sehen.

4.2.6 Erweiterte Suche

Die Unterseite der erweiterten Suche stellt eine detaillierte Suchmaske für das System bereit, um der User Story *Experten Suche* gerecht zu werden. Dabei befindet sich auch hier im linken oberen Eck wieder eine Überschrift mit dem Titel der aktuellen Unterseite. Die Suchmaske selber besteht aus zwei verschiedenen Typen von Suchfeldern.

Ein Typ ist die direkte Textsuche in Feldern der Datensätze, die mit Text befüllt sind. Hierfür existiert links ein Dropdown Menü, welches jeweils alle verfügbaren Felder zur Suche anbietet und rechts neben sich sein Eingabefeld hat. Diese beiden Elemente können jeweils aufgrund des Gesetzes der Nähe und des Gesetzes der Fortsetzung eindeutig zugeordnet werden.

Der andere verfügbare Typ von Suchfeld, sind Suchfelder welche nach Wertebereichen und Datumsbereichen filtern, wie beispielsweise dem Erstellungsdatum des Datensatzes. Diese bestehen aus zwei Textfeldern sowie einer Beschriftung mit dem Titel des zu durchsuchenden Feldes, welche alle in einer Reihe nah beieinander angeordnet sind und somit auch hier aufgrund des Gesetzes der Fortsetzung und des Gesetzes der Nähe eindeutig zueinander zugeordnet werden können. Weitergehend sind um das System fehlertolerant zu gestalten, in den jeweiligen Feldern Hinweise auf das geforderte Eingabeformat vorgegeben.

Sollte eine Suche keinen Treffer ergeben, wird oberhalb des Suchen-Buttons angezeigt, dass kein Element in der Datenbank auf die aktuelle Anfrage zutrifft und neben dem Button ein Kreuz angezeigt. In diesem Fall bleiben die eingegebenen Werte in den Textfeldern stehen, um diese anpassen zu können, ohne dabei alle Felder erneut ausfüllen zu müssen, was dem Grundsatz der Fehlertoleranz entspricht. Kann zu einer Anfrage allerdings mindestens ein Datensatz gefunden werden, wird der Benutzer wie auch bei der Suche über die Navigationsleiste auf die in Abschnitt 4.2.4 vorgestellte Listenansicht weitergeleitet (Erwartungskonformität).

4.2.7 Einstellungsansicht

Die Einstellungsseite ist um sie der Aufgabe angemessen zu halten, nur mit vier Textfeldern mit jeweiliger Beschriftung sowie einem Button zum Abschicken der Anfrage ausgestattet. Sie erfüllt die User Story *Zugangsdatenverwaltung* und bietet somit die Möglichkeit der Passwort- sowie der Benutzernamensänderung.

Das Textfeld des Benutzernamens ist bereits ausgefüllt, wohingegen die drei weiteren Felder für die Passwortänderung jeweils mit einem Hinweis auf die geforderte Eingabe befüllt sind, welcher sich beim Fokus des Feldes automatisch entfernt.

Um dem Benutzer Informationen zum Status seiner Anfrage zu geben, taucht nachdem die Anfrage abgeschickt und bearbeitet wurde, die jeweilige Antwort des Servers auf der Einstellungsansicht auf. War die Anfrage erfolgreich, wird neben dem Button

ein Haken angezeigt. War die Anfrage nicht erfolgreich, wird oberhalb des Buttons die Fehlermeldung sowie neben dem Button ein Kreuz angezeigt. Die Eingaben in die entsprechenden Felder bleiben auch nach einer fehlerhaften Eingabe bestehen, um dem Grundsatz der Fehlertoleranz zu entsprechen.

4.2.8 Detailansicht

Die Detailansicht ist das Pendant zur Seiten-Detailansicht der Listenansicht (vgl. Abschnitt 4.2.4). Sie erfüllt die User Story *Details* und zeigt auf einem den ganzen Bildschirm überspannenden, teilweise durchsichtigen Hintergrund alle zu einem Datensatz verfügbaren Informationen in einer Tabelle an. Auch hier werden wie bei der Seiten-Detailansicht die Titel der jeweils angezeigten Felder fett dargestellt und die dazugehörigen Werte in normaler Schriftstärke, was aufgrund des Gesetzes der Ähnlichkeit auch hier dazu führt, dass alle Daten eindeutig der entsprechenden Gruppe zugeordnet werden können.

Im unteren Bereich bietet die Detailansicht des Weiteren auch die Möglichkeit, den aktuell angezeigten Datensatz direkt durch einen Klick auf den Bearbeiten-Button zu bearbeiten beziehungsweise durch einen Klick auf den Herunterladen-Button herunterzuladen.

Die Besonderheit der Detailansicht sind die jeweils rechts und links am Rand ersichtlichen Schaltflächen. Mittels dieser Schaltflächen kann der Benutzer zwischen allen zuvor ausgewählten Datensätzen umschalten. Diese Schaltflächen sind ausgegraut, falls vor beziehungsweise nach dem aktuellen Element kein weiteres Element mehr ausgewählt wurde, was das System selbstbeschreibend und steuerbar macht.

4.2.9 Bearbeitungsansicht

Im gleichen Stil wie die in Abschnitt 4.2.8 vorgestellte Detailansicht ist auch die Bearbeitungsansicht aufgebaut. Sie zeigt alle verfügbaren Informationen zum aktuell ausgewählten Datensatz in einer Tabelle an, bietet jedoch die Möglichkeit der Veränderung bestimmter Felder durch die Darstellung dieser Werte innerhalb eines Textfelds.

Neben den entsprechenden Feldern und Werten des Datensatzes verfügt die Bearbeitungsansicht über einen Bestätigen-Button, mit welchem sich die aktuellen Änderungen speichern lassen. Das Ergebnis der Speicheranfrage wird wie bei der erweiterten Suche (vgl. Abschnitt 4.2.6) oder auch der Einstellungsansicht (vgl. Abschnitt 4.2.7) neben dem Bestätigen-Button angezeigt. Im erfolgreichen Fall erscheint dort ein Haken, war die Anfrage nicht erfolgreich, erscheint dort ein Kreuz.

Wie auch die Detailansicht bietet die Bearbeitungsansicht links und rechts zwei Schaltflächen um zwischen eventuell vorher ausgewählten Datensätzen zu wechseln. Hat ein Benutzer die vorgenommenen Änderungen noch nicht gespeichert, wird er vor

der Weiterleitung auf das nächste Element gefragt, ob er die Seite sicher verlassen will, was auch hier wiederum die Fehlertoleranz des Systems verbessert. Somit erfüllt die Bearbeitungsansicht die User Story *Bearbeiten*.

5 Implementierung

Das folgende Kapitel beschäftigt sich mit der Implementierung der Anwendung. Neben den Anforderungen an das System und den hierfür benötigten Technologien, werden die Konzepte der einzelnen Komponenten sowie vereinzelte Beispiele der entsprechenden Implementierung vorgestellt.

5.1 Anforderungen, Technologien und Architektur

Der erste Abschnitt stellt die Anforderungen an das System, die passend hierfür ausgewählten Technologien sowie die den Anforderungen und den verwendeten Technologien entsprechende Architektur vor.

5.1.1 Anforderungen an die Anwendung

Aus den in Abschnitt 4.1 vorgestellten Anforderungen geht hervor, dass beispielsweise sehr große Datensätze hoch- und heruntergeladen und auf dem Gerät verarbeitet werden sollen. Da zum Beispiel allein diese Anwendungsfälle auf einem mobilen Endgerät nur schwer realisierbar aber theoretisch möglich sind, bietet sich für das System entweder eine native Desktop Anwendung für das entsprechende Zielsystem oder eine plattformunabhängige Webanwendung an. In diesem Fall wurde aus nachfolgenden Gründen entschieden, das System als Webanwendung zu entwickeln:

Weit verbreitete Technologie
: Nach einer jährlich stattfindenden Umfrage[1] des Onlineforums *Stack Overflow*, sind die hauptsächlich in einer Webanwendung verwendeten Technologien des Jahres 2018 *JavaScript*, *HTML* und *CSS*.

Plattformunabhängig
: Webanwendungen laufen auf jedem System, dass einen Inernetbrowser besitzt.

Zugang von Überall
: Eine Installation ist nicht nötig, was bedeutet, dass ein Zugriff auf das System

[1] `Stack Overflow Umfrage: https://insights.stackoverflow.com/survey/2018` - abgerufen am 17.11.2018

von nahezu überall möglich ist. Auch wenn weiter oben im Text erwähnt wurde, dass ein Einsatz des Systems auf einem mobilen Endgerät nicht geplant ist, ist es mit Webanwendungen dennoch möglich, diese ebenfalls auf beispielsweise einem Smartphone zu öffnen.

5.1.2 Architektur und Relationen

Eine Webanwendung beschreibt eine Anwendung, die auf einem Server läuft. Zu diesem Server können sich Benutzer (auch Clients genannt) verbinden und die entsprechende auf dem Server laufende Webanwendung zu sich lokal herunterladen. In diesem Abschnitt wird die genaue Architektur und die Relationen zwischen den einzelnen Komponenten genauer erläutert.

Abbildung 5.1 zeigt den Aufbau der Architektur. Grundlegend wird das gesamte System wie auch bereits bei der Wahl der Technologien in zwei Teilen unterschieden:

Frontend (vgl. oberen Bereich mit dem Buchstaben **F** in Abbildung 5.1)
Das Frontend bestehend aus den in Abschnitt 5.1.3 vorgestellten Technologien bilden eine Einheit, die als Benutzerschnittstelle (vgl. Ziffer 1 in Abbildung 5.1) dient. Diese Benutzerschnittstelle kommuniziert mittels *HTTP* [5]/*HTTPS* [14] mit dem Backend.

Backend (vgl. unteren Bereich mit dem Buchstaben **B** in Abbildung 5.1)
Das Backend besteht aus einem mithilfe des Frameworks *Django* (vgl. Abschnitt 5.1.4) implementierten Server (nachfolgend als Server bezeichnet), einer Datenbank (nachfolgend als Datenbank bezeichnet) sowie zwei weiteren Servern, einem lokalen (nachfolgend als Zwischenspeicher bezeichnet) und einem externen (nachfolgend als Archiv bezeichnet). Der Server (vgl. Ziffer **2** in Abbildung 5.1) dient in diesem System als zentrales Verwaltungs- und Steuerungssystem zwischen allen eben genannten Komponenten. Wie aus den in Abschnitt 4.1 vorgestellten Anforderungen entnommen werden kann, lädt ein Benutzer ohne weitere Angaben eine Datei eines speziellen Typs in das System. Dabei werden die in der Datei befindlichen Metadaten im Server extrahiert und müssen gespeichert werden. Da die Datensätze jeweils sehr groß sein können, werden diese Dateien nicht in der Datenbank, sondern im Zwischenspeicher (vgl. Ziffer **4** in Abbildung 5.1) sowie dem Archiv (vgl. Ziffer **5** in Abbildung 5.1) gespeichert. Die dazugehörigen Metadaten einer Datei sowie die jeweiligen *URLs* [6] der zum Speicherplatz des Datensatz führenden Adressen, werden für eine bessere Übersichtlichkeit und Verwaltbarkeit in der Datenbank (vgl. Ziffer **3** in Abbildung 5.1) gespeichert.

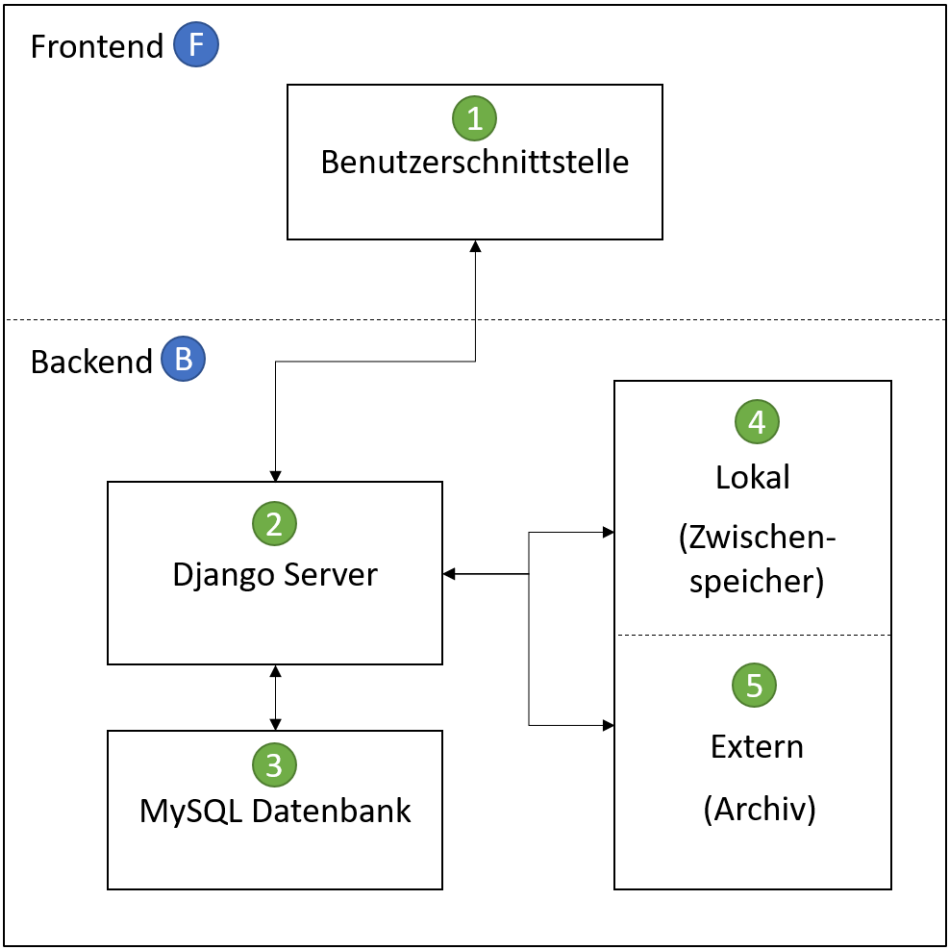


Abbildung 5.1: Diese Abbildung zeigt die grundlegende Architektur des Systems. Dabei kommuniziert das Frontend mit dem Backend (vgl. Abschnitt 5.1.2) über die Benutzerschnittstelle (vgl. Ziffer **1**) und den *Django* Server (vgl. Ziffer **2**). Die Kommunikation zwischen den einzelnen Komponenten ist durch die jeweiligen Pfeile dargestellt, sowie die Richtung der Kommunikation durch die Richtung der Pfeile.

5.1.3 Frontend Technologien

Dieser Abschnitt stellt die für das Frontend (vgl. Abschnitt 5.1.2) verwendeten Technologien einzeln vor und begründet kurz die Wahl der jeweiligen Technologie.

HTML

Die *Hypertext Markup Language (HTML*) ist eine Sprache, welche zur Darstellung beziehungsweise Auszeichnung einer Internetseite zum Einsatz kommt. *HTML* wurde erstmals am 3. November 1992 vom britischen Physiker und Informatiker Tim Berners-Lee spezifiziert und wird vom *World Wide Web Consortium (W3C)*[2] sowie der *Web Hypertext Application Technology Working Group (WHATWG)*[3] stetig weiterentwickelt. *HTML* ist die Kernsprache [12] des Internets, weshalb sie auch in diesem System zum Einsatz kommt.

CSS

Cascading Style Sheets (CSS) ist eine vom *W3C* fortlaufend weiterentwickelte Sprache, die die in Abschnitt 5.1.3 vorgestellte Sprache *HTML* mit Gestaltungsanweisungen unterstützt. Es ist zwar möglich eine Webanwendung ganz ohne *CSS* zu erstellen, ohne *CSS* ist es allerdings nicht möglich die in Abschnitt 4.1 vorgestellten Anforderungen zu erfüllen. Des weiteren ist *CSS* laut der in Abschnitt 5.1.1 vorgestellten Umfrage die am weitest verbreitete Gestaltungssprache für *HTML*.

Eine Weiterentwicklung hiervon stellt die Sprache *Syntactically Awesome Stylesheets (SASS/SCSS)*[4] dar, welche als Präprozessor für *CSS* dient und nach der Entwicklung zu *CSS* zurückübersetzt wird. *SASS/SCSS* kommt in diesem System zum Einsatz, da sie für die Entwicklung sinnvolle Erweiterungen wie Variablen oder beispielsweise Vererbungen für *CSS* mitbringt.

JavaScript

JavaScript (JS) ist eine Programmiersprache, mit der es möglich ist die in Abschnitt 5.1.3 und 5.1.3 vorgestellten Sprachen *HTML* und *CSS* zu manipulieren. Das sogenannte *ECMAScript*[5] bildet dabei den standardisierten Sprachkern *JavaScripts* und wurde 1995 von Brendan Eich entwickelt. Nach der in Abschnitt 5.1.1 vorgestellten Studie ist *JavaScript* zum sechsten Mal in Folge, die am meisten verwendete Technologie der Welt. Nicht nur Aufgrund ihrer Popularität, sondern auch aufgrund ihrer Funktionen und in großen Mengen verfügbaren Bibliotheken, kommt *JavaScript* als dritte ergänzende Technologie in diesem System zum Einsatz.

[2] `W3C: http://www.w3.org/standards/techs/html` - abgerufen am 26.10.2018
[3] `WHATWG: https://html.spec.whatwg.org/multipage/` - abgerufen am 26.10.2018
[4] `SASS: http://sass-lang.com/` - abgerufen am 26.10.2018
[5] `ECMAScript: https://www.ecma-international.org/ecma-262/` - abgerufen am 21.11.2018

Eine dabei sehr bekannte Erweiterung stellt die Bibliothek *jQuery*[6] dar, welche die Sprache um einige nützliche Abkürzungen erweitert und deshalb ebenfalls als Bestandteil der im Frontend des Systems verwendeten Technologien zählt.

5.1.4 Backend Technologien

In diesem Abschnitt werden die für das Backend (vgl. Abschnitt 5.1.2) verwendeten Technologien einzeln vorgestellt und jeweils kurz die Wahl der entsprechenden Komponente begründet.

Django

Das in diesem Projekt eingesetzte Webframework ist das quellenoffene Framework *Django*[7]. *Django* wurde mithilfe der Programmiersprache *Python*[8] entwickelt und nimmt alle vom Entwickler gegebenen Anweisungen durch eben diese Programmiersprache entgegen. Wie der in Abschnitt 5.1.1 vorgestellten Umfrage entnommen werden kann, stellt *Django* eines der bekanntesten Webframeworks der Welt dar. Das Webframework *Django* bietet Features wie unter anderem eine Templatesprache, ein System zur Authentisierung, ein System zur Datenbankverwaltung sowie verschiedene Sicherheitsmechanismen.

Die eben genannten Features stellen nur einen kleinen Teil der Möglichkeiten *Djangos* dar, alle in Abschnitt 4.1 vorgestellten Anforderungen, werden jeweils durch ein Feature *Djangos* abgedeckt. Aufgrund der zahlreichen für die Anforderungen nützlichen Features sowie der leichten Handhabung, einer sehr detaillierten Dokumentation und der großen Popularität, dient *Django* diesem System als der Hauptbestandteil des Backends.

MySQL

MySQL[9] ist ein relationales Datenbankverwaltungssystem, das von der *Oracle Corporation*[10] stetig weiterentwickelt wird. Mit *MySQL* ist es auf sehr leichtem Wege möglich, Daten abzuspeichern und wieder zu laden. Wie in Abschnitt 4.1 vorgestellt, muss das System viele Daten und Dateien behandeln und verwalten, wobei eine Datenbank ein sehr nützliches Hilfsmittel darstellt. Nach der in Abschnitt 5.1.1 erwähnten Umfrage, ist *MySQL* mit 58.7% das am meisten genutzte Datenbankverwaltungssystem der Welt. Da *MySQL* weitergehend ein quellenoffenes System ist und somit kostenlos verwendet

[6] `jQuery: https://jquery.com/` - abgerufen am 26.10.2018
[7] `Django: https://www.djangoproject.com/` - abgerufen am 26.10.2018
[8] `Python: https://www.python.org/` - abgerufen am 26.10.2018
[9] `MySQL: https://www.mysql.com/` - abgerufen am 17.11.2018
[10] `Oracle: https://www.oracle.com/` - abgerufen am 17.11.2018

werden kann, die Datensätze dieses Systems auch relational sind und *MySQL* des Weiteren eine gute Anbindungsmöglichkeit an das zuvor gewählte Webframework *Django* (vgl. Abschnitt 5.1.4) besitzt, kommt *MySQL* als Datenbankverwaltungssystem in diesem System zum Einsatz.

FTP Server

Wie in Abschnitt 4.1 gezeigt, sollen die Datensätze an mehreren Stellen gespeichert werden. Deshalb kommen in diesem System neben dem eigentlichen Server welcher durch *Django* (vgl. Abschnitt 5.1.4) betrieben wird, noch zwei weitere Server zum Einsatz. Ein lokaler Server, auf welchem die Daten wie auf einer Art Zwischenspeicher gespeichert werden, sowie ein externer Server, welcher als Archiv fungiert, auf welchem alle Daten und Dateien gespeichert sind. Für die Kommunikation zwischen diesen Servern, könnte das *File Transfer Protocol (FTP)* [7] verwendet werden. Eine Erweiterung beziehungsweise Alternative zum *FTP* stellt das *SSH File Transfer Protocol (SFTP)* [8] dar, welches die Verschlüsselung der zu übertragenden Daten ermöglicht und somit für mehr Sicherheit beim Übertragen der Daten sorgt, weshalb die eben genannten Server mittels *SFTP* und nicht *FTP* kommunizieren.

5.2 Authentisierung, Authentifizierung und Autorisierung

Wie aus den Anforderungen aus Abschnitt 4.1 hervorgeht, wird aufgrund von verschiedenen Berechtigungen und Möglichkeiten zur Interaktion mit dem System ein Benutzerverwaltungssystem benötigt. Ein solches System besteht aus grundlegend drei Teilen, der Authentisierung, der Authentifizierung sowie der Autorisierung.

Der Nachweis einer Person, dass sie die Person ist für die sie sich ausgibt, wird als Authentisierung bezeichnet. Dies ist beispielsweise die Eingabe von Anmeldedaten wie Benutzername und Passwort in ein System. Direkt nach der Authentisierung folgt die Authentifizierung, welche den Vorgang der Prüfung der Daten einer Person nach der Authentisierung beschriebt. Die Autorisierung beschreibt weitergehend den Vorgang, der Einräumung spezieller Rechte. Konnte eine Person identifiziert werden, muss weitergehend geprüft werden, ob sie die benötigten Rechte besitzt eine bestimmte Interaktion mit dem System ausführen zu können, wie beispielsweise das Löschen oder Bearbeiten von Datensätzen in diesem System.

Ein Framework, welche all die eben genannten Vorgänge verwaltet, stellt *Django* mit dessen sogenanntem *Authentication Framework* zur Verfügung. Neben einer einfachen Benutzer- und Gruppenverwaltung, ist es hiermit unter anderem möglich, in der Datenbank erstellte Datensätze und deren Zugriffsrechte zu verwalten.

Da *HTTP* ein zustandsloses Protokoll ist, es allerdings für einzelne Anfragen nötig ist den Absender eindeutig zuordnen zu können, werden der Anfrage eindeutige Informationen zugeordnet. Diese Informationen werden in sogenannten *HTTP cookies* [13] gespeichert und sind Teil einer *HTTP* Antwort des Servers. Solche Informationen können beispielsweise der Inhalt eines Warenkorbs sein oder wie in diesem Fall Daten für die jeweilige Session (dt.: Sitzung). Eine Session beinhaltet alle Daten, die ein Server bei der Kommunikation mit einem Client zwischenspeichern möchte.

Sobald ein Benutzer mittels einer *HTTPS* Anfrage versucht sich beim Server zu Authentisieren, versucht *Django* diesen Benutzer zu Authentifizieren und sendet falls erfolgreich einen *Cookie* an den Benutzer zurück, welcher eine sogenannte `Session ID` beinhaltet. Ab dieser Anfrage ist es wichtig, anstatt über *HTTP*, mittels *HTTPS* zu kommunizieren, um die zu übertragenden Daten zu verschlüsseln. Die eben genannte `Session ID` ist eine lange einzigartige Kette aus Buchstaben und Zahlen, welche auf einen nach der erfolgreichen Authentifizierung gespeicherten Datensatz in der Datenbank zeigt. Die Tabelle `django_session` der Datenbank ist die entsprechende Tabelle in *Djangos Authentication Framework*, welche jeweils die `Session ID`, die dazugehörigen Daten sowie ein Ablaufdatum speichert. Bei einer Anmeldung wird in diesen Daten die Benutzer `ID` gespeichert, um die Session dem entsprechenden Benutzer zuordnen zu können. Nachdem ein Nutzer angemeldet ist, sendet er bei jeder weiteren Anfrage die ihm gegebene `Session ID` innerhalb des *HTTP Headers* mit an den Server.

Stellt ein angemeldeter Benutzer nun beispielsweise eine Anfrage an den Server bestimmte Datensätze zu laden (beispielsweise beim Aufrufen der Listenansicht aus Abschnitt 4.2.4), sendet er bei dieser Anfrage seine `Session ID` mit an den Server. Im nächsten Schritt identifiziert *Django* diesen Benutzer über die mitgegebene `Session ID`, lädt nach gelungener Autorisierung zu dieser Aktion die entsprechenden Datensätze und sendet diese an den anfragenden Benutzer zurück.

Stellt der Benutzer eine Anfrage zur Abmeldung an den Server, wird hier lediglich der aktuelle Eintrag der Session in der Datenbank gelöscht und eine andere `Session ID` zurück gesendet, welche auf einen leeren Datensatz zeigt.

5.3 Datenbankverwaltung

Im vorhergehenden Abschnitt 5.1.2, wurde bereits die Speicherung der Metadaten der Datensätze und Benutzer in einer *MySQL* Datenbank angesprochen. Es gibt viele verschiedene Datenbankverwaltungssysteme, welche alle eine Datenbanksprache, die *Structured Query Language (SQL)*, unterstützen. Dabei hat jedes Datenbankverwaltungssystem kleine eigene Erweiterungen, der Grundaufbau einer sogenannten *SQL* Anfrage bleibt jedoch gleich. Um die direkte Ansprache der Datenbank mittels *SQL*

aus der jeweiligen Programmiersprache zu abstrahieren, gibt es viele Frameworks, die auf eine Datenbank eine sogenannte *Object-Relational Mapping* (*ORM* - deutsch: Objektrelationale Abbildung) implementieren. Eine objektrelationale Abbildung einer Datenbank übersetzt beziehungsweise abstrahiert die Datensätze einer Datenbank zu in der jeweiligen Programmiersprache veränder- und abrufbaren Objekten. Eine *ORM* einer Datenbank bringt folgende Vorteile mit sich:

Entwicklungsgeschwindigkeit
: Durch die Abstraktion *SQLs* zu in der jeweiligen Programmiersprache veränder- und abrufbaren Objekten, kann schnell und effizient in einer Programmiersprache entwickelt werden.

Entwicklungskosten
: Durch die Möglichkeit der schnelleren Entwicklung wie im ersten Punkt genannt, reduziert sich nicht nur die benötigte Zeit sondern auch die Kosten der Entwicklung.

Dialektsicherheit
: Wie bereits weiter oben im Text erwähnt, gibt es viele Datenbankverwaltungssysteme, welche teilweise einen eigenen Dialekt sprechen. Durch eine *ORM* können diese Dialekte einmal implementiert werden und können danach ohne großen Rechercheaufwand wiederverwendet werden.

Sicherheit
: *ORMs* übersetzen gestellte Anfragen in *SQL* Syntax und können dabei sicherstellen, dass bei Anfragen mit sogenannten *Prepared Statements* (deutsch: vorbereitete Anfragen) gearbeitet wird, was die Verwendung der Datenbank sicherer macht, da diese Vorgänge bei der Entwicklung nicht weiter bedacht werden müssen.

Wie in Abschnitt 5.1.4 erwähnt, besitzt *Django* eine solche objektrelationale Abbildung. Eine Abfrage nach dem Benutzer mit der `ID 3` würde in *Python* für eine *MySQL* Datenbank wie in Listing 5.1 aussehen, wohingegen die gleiche Abfrage in *Djangos ORM* wie in Listing 5.2 implementiert werden kann.

```
1 user = User.objects.get(id=3);
```

Listing 5.2: Eine Abfrage mittels Filter an die *Django ORM*, um den Benutzer mit der `ID 3` zu laden.

```
1  import MySQLdb
2
3  database = MySQLdb.connect(...)
4  cursor = database.cursor()
5
6  cursor.execute("SELECT * FROM USERS WHERE ID = 3;")
7  row = cursor.fetchone()
8
9  user = User(first_name=row["first_name"],
       last_name=row["last_name"], ...);
10
11 db.close()
```

Listing 5.1: Eine *SQL* Anfrage an einen *MySQL* Server nach dem Benutzer mit der `ID 3` in *Python*.

5.4 Anfragespezifische Funktionen (Views)

In diesem System gibt es wie aus den in Abschnitt 4.1 vorgestellten Anforderungen entnommen werden kann, einige verschiedene Anfragen eines Benutzers an den Server. Um diese Anfragen strukturiert bearbeiten zu können, existieren in *Django* sogenannte *Views* (deutsch: Ansichten). Eine solche *View* durchläuft nach dem Aufruf drei Phasen:

1. Das Annehmen einer *HTTP* Anfrage die durch das *URL* Routing (vgl. Abschnitt 5.6) weitergegeben wurde
2. Die Bearbeitung der *HTTP* Anfrage
3. Die Rückgabe des Ergebnisses der Bearbeitung mittels einer *HTTP* Antwort

In einer ersten Phase wird, wie in Abschnitt 5.6 erläuert, eine Anfrage an eine definierte *View* weitergeleitet, bevor sie in einem zweiten Schritt von dieser *View* bearbeitet wird. In der zweiten Phase wird die gegebene Anfrage bearbeitet. Hierbei können entsprechend der *View* beispielsweise entweder Datensätze erzeugt, bearbeitet oder gelöscht werden. Nachdem eine Anfrage abgearbeitet wurde, antwortet eine *View* mit einer *HTTP* Antwort. Diese übergibt einen Statuscode, welcher den Status der Anfrage beschriebt, wie beispielsweise `HTTP 200` für eine erfolgreiche Ausführung, `HTTP 401` falls der Benutzer für diesen Vorgang nicht autorisiert werden konnte oder neben dem `HTTP 200` auch noch Datensätze innerhalb des Inhalts der *HTTP* Antwort, falls angefordert.

In *Django* definierte *Views* können weitergehend auf verschiedene Typen von Anfragen reagieren. So gibt es beispielsweise die Möglichkeit auf eine `HTTP GET` Anfrage

```
class ArchiveListView(...):
  # Bearbeitung einer HTTP GET Anfrage
  def get(...):
    ...
    return render(..., {"object_list": object_list, ...})

  # Bearbeitung einer HTTP POST Anfrage
  def post(...):
    return render(..., {"object_list":
        get_object_list(request)})
```

Listing 5.3: Ein Beispiel für das unterschiedliche Antworten auf verschiedene Typen von *HTTP* Anfragen unter Verwendung der *Django* eigenen Templatesprache (vgl. Abschnitt 5.5).

anders zu reagieren als auf eine `HTTP POST` Anfrage, wie in dem Beispiel 5.3 aus der für die Listenansicht zuständigen *View* des Systems zu sehen ist.

5.5 Template-Engine

Wie den Wireframes aus Abschnitt 4.2 entnommen werden kann, besteht die Webanwendung aus vielen redundanten Teilen und auch Inhalten wie Listen, die dynamisch generiert werden müssen. Um diese sich wiederholenden Elemente wie beispielsweise die Navigationsleisten nicht mehrfach implementieren zu müssen und auch um gelieferte Datensätze aus dem Backend dynamisch auf der jeweiligen Unterseite anzeigen lassen zu können, existiert in *Django* eine sogenannte *Template-Engine* (englisch für Vorlagen-Maschine). Mit einer *Template-Engine* ist es möglich wie eben genannt *HTML* dynamisch zu generieren und ausgesuchte Inhalte an andere Stellen zu laden. Die *Template-Engine* funktioniert wie folgt:

1. Innerhalb der gewünschten *HTML* Datei wird für die *Template-Engine* ein Abschnitt markiert (vgl. Listing 5.4). Dieser beschreibt wenn mit zwei geschwungenen Klammern angegeben (`{{...}}`), dass *HTML* mit dem durch den Parameter übergebenen Inhalt generiert werden soll oder wenn durch einen Block mit geschwungenen Klammern und Prozentzeichen (`{%...%}`) angegeben, dass an dieser Stelle etwas in die Datei beziehungsweise der folgende Block aus der Datei geladen werden soll.

2. Nachdem eine Anfrage erfolgreich an eine *View* (vgl. Abschnitt 5.4) gestellt und diese bearbeitet wurde, werden die vor dem Senden der für die Anzeige beim anfragenden Benutzer notwendigen Dateien, durch die *Template-Engine*

```
{% extends ... %}

{% block content %}
  {% for file in object_list %}
    <tr ...>
      ...
      <td ...>{{ file.file_name }}</td>
      <td ...>{{ file.subject_name }}</td>
      <td ...>{{ file.subject_date }}</td>
    </tr>
  {% endfor %}
{% endblock %}
```

Listing 5.4: Ein Beispiel für die Verwendung der *Template-Engine* (vgl. Abschnitt 5.5) aus der Implementierung der Listenansicht des Systems.

bearbeitet und die entsprechenden Operationen ausgeführt. Dabei kommen in diesem Beispiel wie in Listing 5.3 zu sehen ist Daten als `object_list` von der *View* zurück, auf die in Listing 5.4 in einer for-Schleife zugegriffen wird und die entsprechenden Daten in die dafür vorgesehenen Felder geladen werden.

3. Nach dem Verändern der Dateien, werden diese fertig zur Anzeige an den Benutzer geschickt.

Djangos Template-Engine besteht aus sehr vielen Funktionen. Nähere Details über alle Features der in *Django* enthaltenen *Template-Engine* lassen sich der Spezifikation [1] entnehmen.

5.6 URL Routing

Diese Webanwendung besteht aus mehreren verschiedenen Unterseiten und Funktionalitäten, die teilweise über sogenannte *Ajax*-Anfragen [15] gestellt werden. Da dies zu für einen Menschen unleserlichen *URLs* führen kann, gibt es innerhalb *Djangos* einen sogenannten *URL dispatcher*, der sich um das Routing zwischen angefragter *URL* und entsprechender *View* kümmert. Der *URL dispatcher* ist bei einer Anfrage die erste aufgerufene Komponente (vgl. Abbildung 5.2) des *Django* Servers und versucht die angefragte *URL* auf eine *Python* Funktion (die jeweilige *View* (vgl. Abschnitt 5.4)) abzubilden.

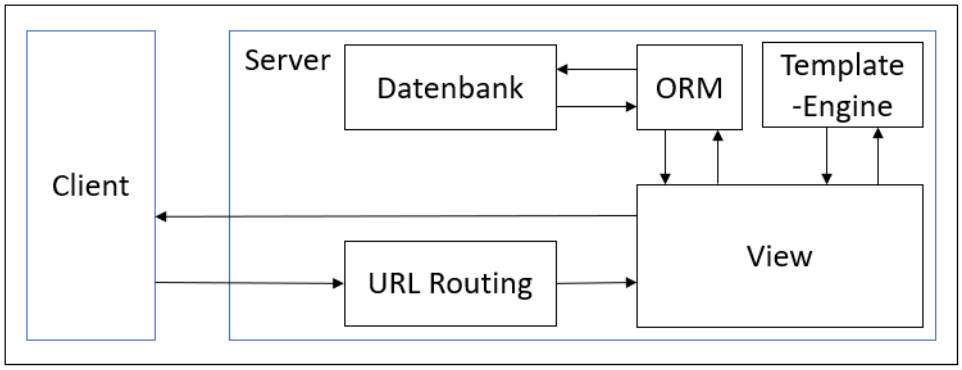


Abbildung 5.2: Diese Abbildung zeigt das Zusammenspiel und die Reihenfolge der Aufrufe und Kommunikation zwischen den einzelnen in den Abschnitten 5.3, 5.4, 5.5 und 5.6 vorgestellten Komponenten nach einer Anfrage an den *Django* Server durch einen Benutzer. Die Pfeile stellen die Kommunikation und Richtung der Kommunikation zwischen zwei Komponenten dar.

Nach dem Eintreffen einer Anfrage arbeitet der *URL dispatcher* wie in der aktuellen Version der Dokumentation *Djangos* [2] nachzulesen ist, folgende Punkte der Reihe nach ab:

1. Da es in einem *Django* System mehrere Konfigurationen für den *URL dispatcher* geben kann, lädt *Django* die aktuell zu verwendende Konfiguration.
2. Nachdem die richtige Konfiguration gefunden wurde, lädt *Django* die vorher definierte Variable `urlpatterns` (vgl. Listing 5.5).
3. Nun geht *Django* in einer Schleife über alle in der eben genannten Variablen stehenden Verweise und stoppt beim Ersten, der der *URL* entspricht.
4. Wurde ein passender Verweis gefunden, importiert *Django* die entsprechende *View* und ruft diese mit den entsprechenden Parametern auf.
5. Sollte kein passender Verweis gefunden werden, ruft *Django* die hierfür definierte *Fehler-View* auf.

5.7 Sicherheit in Django

Neben den in den vorhergehenden Abschnitten genannten Implementierungsvorteilen bei der Verwendung *Djangos*, sind die von *Django* mitgelieferten Sicherheitsmechanismen ein weiterer Vorteil. Das System läuft voraussichtlich auf einem Endgerät mit

```
urlpatterns = [
    ...
    path('item/<int:pk>', ItemFormView.as_view(), name='edit'),
    path('recent', RecentView.as_view(), name='recent'),
    ...
]
```

Listing 5.5: Ein Beispiel für die Konfiguration des in Abschnitt 5.6 vorgestellten *URL dispatchers* mittels der `urlpatterns` Variable.

Internetzugang beziehungsweise wird von einem solchen aufgerufen und ist somit auch vor potentiellen Angreifern nicht verborgen. *Django* bietet zahlreiche Features, die die Sicherheit des Systems verbessern, wie in der aktuellen Dokumentation *Djangos* [3] nachzulesen ist. Im Folgenden werden nun zwei der wichtigsten Features genannt und kurz vorgestellt.

5.7.1 Cross-Site-Request-Forgery (CSRF) Schutz

Ein *Cross-Site-Request-Forgery (CSRF)* Angriff, ist ein Angriff bei dem ein Angreifer sich bei einem System als ein Anderer dort bereits angemeldeter Benutzer ausgibt. Hierfür nimmt der Angreifer die beim Opfer im Internetbrowser gespeicherte `Session ID` (vgl. Abschnitt 5.2) für seine eigenen Anfragen und gibt sich so beim Server als das Opfer aus. Viele Webframeworks bieten für diese Art von Angriff einen Schutz an, so auch *Django*. Bei jedem Senden der anzuzeigenden Elemente, wird mittels der in Abschnitt 5.5 vorgestellten *Template-Engine* ein sogenannter `csrf_token` an den Benutzer mitgesendet. *Django* fordert standardmäßig bei jeder Anfrage auf ein durch die Anmeldung geschützten Bereich diesen weiteren, eigens für diese Anfrage generierten einzigartigen `csrf_token`. Somit kann sich ein Angreifer selbst wenn ihm die `Session ID` bekannt ist, nicht als das Opfer ausgeben, da der Server nur Anfragen bearbeitet, die den aktuell gültigen `csrf_token` zusätzlich zur `Session ID` mitliefern. Ein solcher `csrf_token` kann dank der *Template-Engine* (vgl. Abschnitt 5.5) innerhalb eines *HTML* Dokuments folgendermaßen eingebaut werden: `{% csrf_token %}`.

5.7.2 Schutz vor SQL-Injections

Eine weitere sehr bekannte Art von Angriff ist eine sogenannte *SQL-Injection*. Im Gegensatz zum im vorherigen Abschnitt vorgestellten *CSRF-Angriff*, zielt dieser Angriff im Allgemeinfall nicht direkt auf einen Nutzer ab, sondern auf die komplette Datenbank. Bei einer *SQL-Injection* gelingt es einem Angreifer seinen eigenen *SQL* Code auf der entsprechenden Datenbank auszuführen, was dazu führen kann, dass er Zugriff

auf Daten gewinnt, auf die er normalerweise keinen Zugriff hätte und diese einsehen, verändern oder gar löschen kann. Das Webframework *Django* kann allerdings auch mit dieser Art von Angriff umgehen, indem es Anfragen an die Datenbank innerhalb der *ORM* (vgl. Abschnitt 5.3) parametrisiert und alle von Benutzern eingegebenen Zeichenfolgen auf ungewünschte Symbole und überflüssige Anführungszeichen filtert und verändert.

6 Vorstellung des System

Das folgende Kapitel stellt die finale Webanwendung vor. Die jeweiligen Unterseiten des Systems werden dabei jeweils kurz erläutert und anhand eines Screenshots illustriert. Hierfür wird das System in seinen Kernaufgaben in zwei Abschnitte untergliedert. Abschnitt 6.1 stellt alle benutzerspezifischen Interaktionen vor, bevor in Abschnitt 6.2 alle die Datensätze betreffende Interaktionsmöglichkeiten präsentiert werden.

6.1 Benutzerbezogene Interaktionen

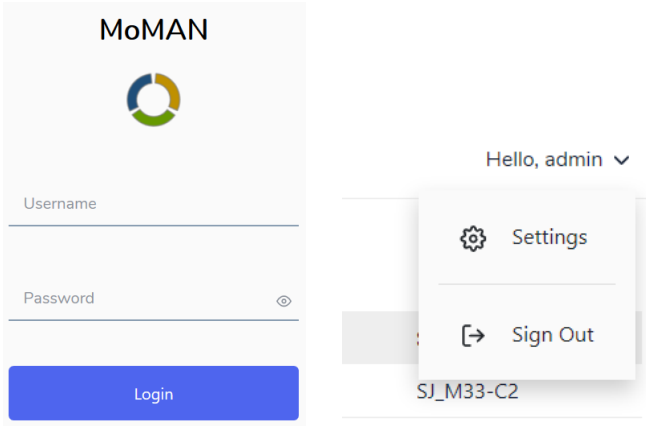
Dieser Abschnitt beschäftigt sich mit allen benutzerbezogenen Interaktionsmöglichkeiten und stellt diese anhand einer Erklärung sowie einem Ausschnitt aus der Anwendung im Folgenden vor.

Anmeldung Abbildung 6.1a zeigt einen Ausschnitt der Anmeldungsseite. Sie ist die erste dem Benutzer ersichtliche Oberfläche des Systems, beim Aufrufen der Webanwendung über einen Internetbrowser und stellt den öffentlich sichtbaren Teil der Anwendung dar. Sie funktioniert wie in Abschnitt 4.2.1 beschrieben und kann nur mit einem bereits vorher durch den Administrator des Systems angelegten Benutzeraccount erfolgreich verwendet werden. Sie dient der Authentifizierung eines Benutzers beim System.

Sollte die Authentifizierung (vgl. Abschnitt 5.2) fehlschlagen, werden die entsprechenden Fehler in roter Schriftfarbe über dem Login-Button angezeigt. Nur nach erfolgreicher Authentifizierung des Benutzers beim Server, kann von dieser Seite des Systems auf eine weitere Unterseite gewechselt werden.

Abmeldung Will sich ein Benutzer vom System abmelden, ist dies über das Detailmenü der oberen Navigationsleiste möglich (vgl. Abbildung 6.1b). Die Abmeldung funktioniert technisch wie in Abschnitt 5.2 beschrieben und ist erst nach einer erfolgreichen Anmeldung am System möglich.

Einstellungen Durch die in Abbildung 6.2 gezeigte Einstellungsansicht ist es einem Benutzer möglich, seine Benutzerspezifischen Daten wie den Benutzernamen oder das Passwort zu ändern, ohne dafür einen Administrator kontaktieren zu müssen. Die Funktionsweise der Einstellungsansicht wurde in Abschnitt 4.2.7 bereits vorgestellt.



(a) (b)

Abbildung 6.1: (a) Zeigt das Anmeldeformular der Webanwendung. (b) Zeigt das über die obere Navigationsleiste erreichbare Untermenü (vgl. Abschnitt 4.2.2).

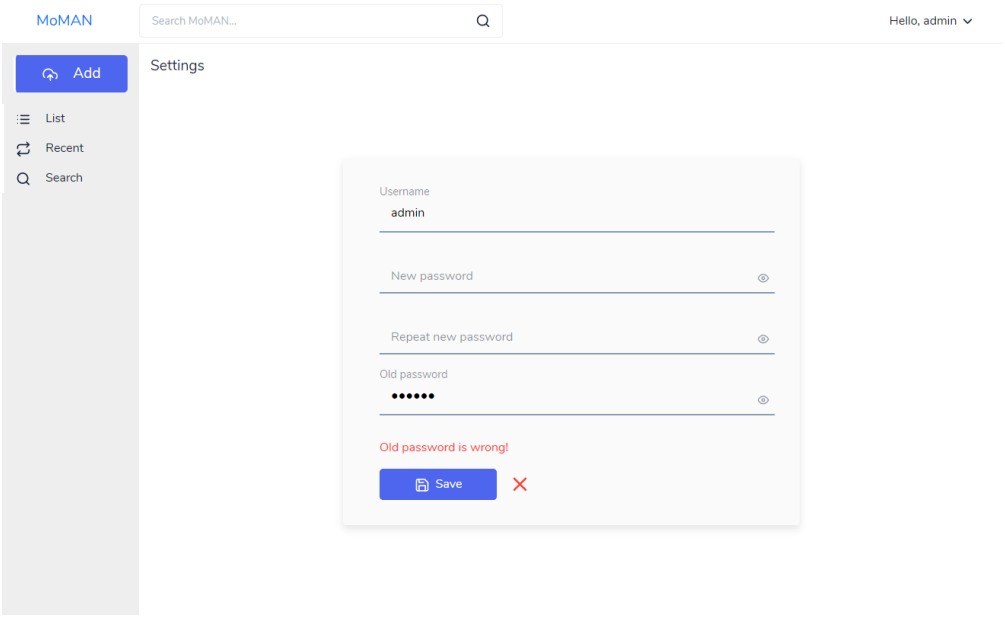


Abbildung 6.2: Diese Abbildung zeigt einen Ausschnitt der Einstellungsseite der Webanwendung, nachdem das Passwort zur Bestätigung falsch eingegeben wurde (vgl. Abschnitt 4.2.7).

6.2 Interaktionen mit Datensätzen

In diesem Abschnitt werden alle Interaktionen mit den im System gespeicherten Datensätzen behandelt und anhand einer kurzen Erklärung sowie direkten Ausschnitten aus der Anwendung vorgestellt.

Anzeige aller Datensätze Nachdem sich ein Benutzer erfolgreich bei der in Abschnitt 4.2.1 vorgestellten Anmeldeseite authentifizieren konnte, wird er automatisch auf die Listenansicht weitergeleitet. Die in Abbildung 6.3 ersichtliche Listenansicht stellt die Hauptseite der Webanwendung dar und kann wie in Abschnitt 4.2.4 beschrieben verwendet werden. Hier werden einem Benutzer alle seiner Gruppen sichtbaren Datensätze angezeigt. Da es sich hierbei um sehr viele Datensätze handeln kann, wurden um die Webanwendung auch auf Geräten mit wenig Leistung flüssig ausführen zu können, sowie die Ladezeit zu minimieren, Seiten eingeführt. Pro Seite werden 50 Datensätze angezeigt, was das System beim Betrachten der Listenansicht performanter macht. Diese nach dem Aufrufen der Listenansicht angezeigten Datensätze, werden absteigend nach dem Datum des Hochladens in das System sortiert und angezeigt. Auch die Breite der jeweiligen Spalten der Tabelle wurde auf ein Maximum begrenzt, um zu große Einträge auch auf kleinen Bildschirmen sichtbar zu machen.

Neben der Anzeige aller Dateien beim Aufrufen der Listenansicht, dient die Listenansicht auch als Anzeige für alle in diesem System generierten Suchergebnisse. Eine Suche aus der Navigationsleiste sowie eine Suche aus der erweiterten Ansicht (vgl. Abschnitt 4.2.6) leiten jeweils auf die Listenansicht weiter. Sollte bei einer Suchanfrage über die Navigationsleiste kein passender Datensatz gefunden werden können, beziehungsweise generell kein für die Gruppe des Benutzers sichtbarer Datensatz vorhanden sein, wird in der Listenansicht eine entsprechende Meldung angezeigt.

Anzeige zuletzt veränderter und hinzugefügter Dateien Die in Abbildung 6.4 zu sehende Zuletztansicht, kann über die linke Navigationsleiste aufgerufen werden und funktioniert wie in Abschnitt 4.2.5 beschreiben. Auf ihr werden in zwei Tabellen die zehn zuletzt hinzugefügten sowie die zehn zuletzt bearbeiteten Datensätze angezeigt. Im Gegensatz zur Listenansicht bietet die Zuletztansicht keine Seiten mit allen Datensätzen an, sondern wie eben beschrieben nur jeweils die letzten zehn Datensätze. Sollten den Gruppen eines Benutzers keine Datensätze ersichtlich sein, wird auch hier je Tabelle angezeigt, dass keine Datensätze zur Anzeige vorhanden sind.

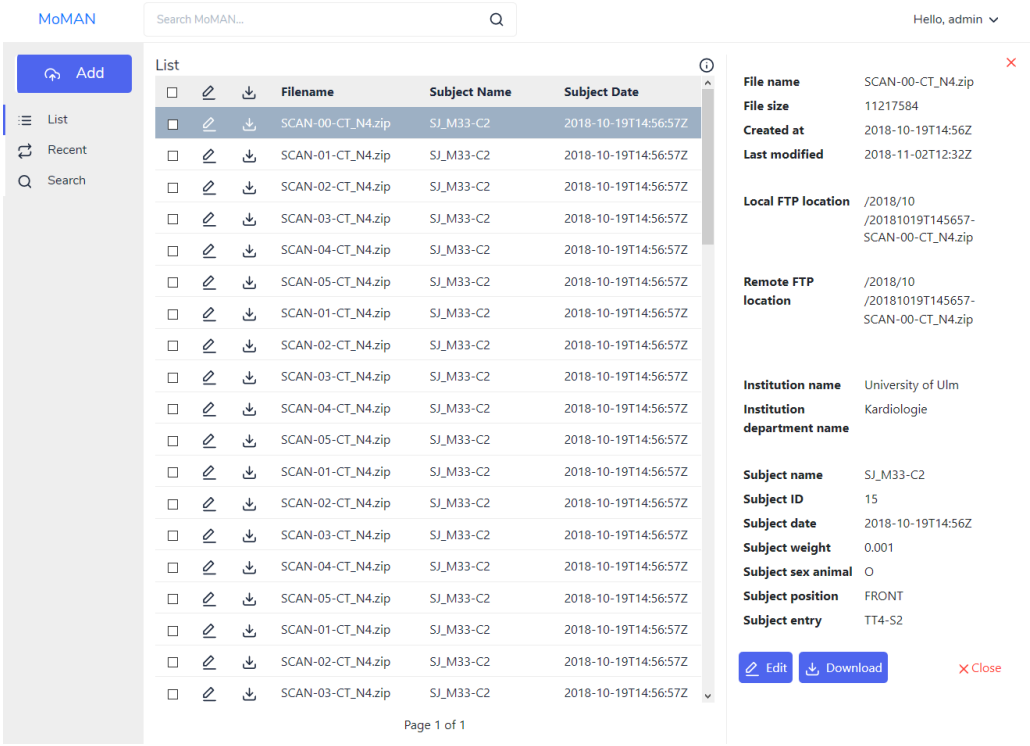


Abbildung 6.3: Diese Abbildung zeigt die Listenansicht der Webanwendung. In diesem Ausschnitt wurde durch einen Klick auf einen der Einträge die Seiten-Detailansicht geöffnet (vgl. Abschnitt 4.2.4).

Hochladen von Dateien Zum Hochladen von Dateien dient die Uploadansicht (vgl. Abschnitt 4.2.3). Sie bietet dem Benutzer die einzige Möglichkeit, dem System neue Dateien hinzuzufügen und funktioniert wie in Abschnitt 4.2.3 beschreiben. Nach dem erfolgreichen Hochladen der gewünschten Datei beziehungsweise der gewünschten Dateien, ruft die Uploadansicht die in Abschnitt 4.2.9 vorgestellte Bearbeitungsansicht auf und bietet dem Benutzer vor dem finalen Speichern der Datensätze im System noch die Möglichkeit, alle aus den Dateien herausgelesenen Metadaten und sonstige Informationen zu bearbeiten.

Herunterladen von Dateien Das Herunterladen ist wie in Abschnitt 4.2 erwähnt entweder über die Listenansicht, die Zuletztansicht oder die Detailansicht möglich. Hierfür muss beim entsprechenden Datensatz entweder in entsprechendem Listeneintrag das Herunterladen-Icon (vgl. Abbildung 6.3 und 6.4) oder der Herunterladen-Button in der Seiten-Detailansicht (vgl. Abbildung 6.3), der Schnellauswahlleiste (vgl. Abbildung 6.4) oder der Detailansicht (vgl. Abbildung 6.6) geklickt werden.

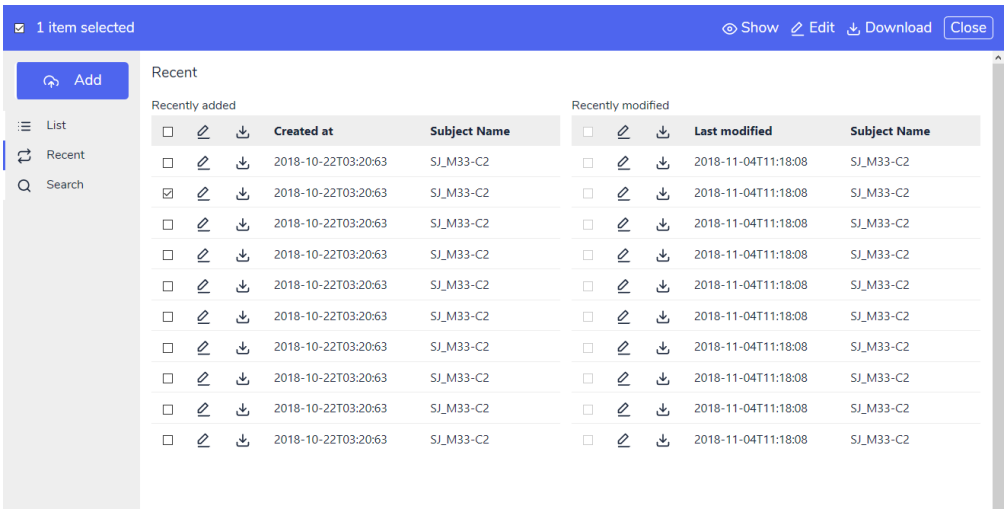


Abbildung 6.4: In dieser Abbildung ist ein Ausschnitt der Zuletztansicht (vgl. Abschnitt 4.2.5) der Webanwendung zu sehen, wobei durch einen Klick in eine der Checkboxen die Schnellauswahlleiste geöffnet wurde.

Gezieltes Suchen Um alle verfügbaren Datensätze zu filtern existieren zwei Möglichkeiten. Die erste Möglichkeit bietet eine einfache Suche (vgl. beispielsweise Abbildung 6.3), welche alle verfügbaren Datensätze nach einer Eigenschaft durchsucht. Sollte ein Benutzer einen gewünschten Datensatz über die sich in der Navigationsleiste befindliche Suchleiste (vgl. Abschnitt 4.2.2) nicht finden, beziehungsweise kann seine Anfrage nicht speziell genug stellen und bekommt zu viele Datensätze als Antwort, hat er mit der Unterseite der erweiterten Suche die Möglichkeit eine Suchanfrage zu starten, welche alle Datensätze auf mehrere Eigenschaften gleichzeitig durchsuchen kann. Die Ansicht der erweiterten Suche ist in Abbildung 6.5 zu sehen, sowie dessen Funktionsweise in Abschnitt 4.2.6. Wurde über eine der beiden Suchoptionen ein Ergebnis zu einer Anfrage gefunden, wird der Benutzer an die Listenansicht weitergeleitet.

Nach einer direkten Absprache mit den Auftraggebern wurde herausgearbeitet, dass es sich bei dem Namen des Subjektes um die am häufigsten durchsuchte Information eines Datensatzes handelt. Aufgrund dieser Information wurde entschieden, die einfache Suche beziehungsweise das Filtern über die Navigationsleiste auf diese Eigenschaft der Datensätze anzuwenden.

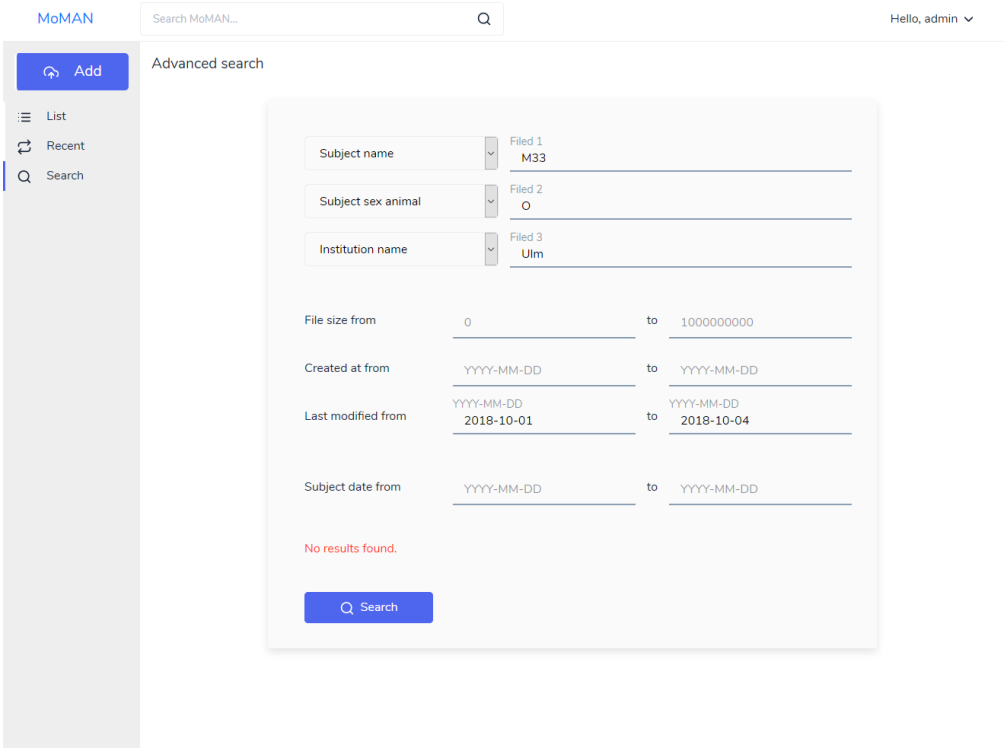


Abbildung 6.5: Diese Abbildung zeigt die Ansicht der erweiterten Suche (vgl. Abschnitt 4.2.6) der Webanwendung. Für die in der Abbildung eingegebenen Werte konnte in diesem Beispiel kein passender Datensatz gefunden werden, weshalb dies dem Benutzer in der Fehlermeldung oberhalb des Such-Buttons mitgeteilt wird.

Anzeige von Details Nachdem sich ein Benutzer auf der in Abschnitt 4.2.5 vorgestellten Zuletztansicht dazu entschieden hat alle zu einem Datensatz verfügbaren Informationen anzusehen, öffnet sich die in Abbildung 6.6 dargestellte Detailansicht. Die Detailansicht bietet die Möglichkeit, alle Informationen eines Datensatzes kompakt darzustellen, ohne diesen Datensatz vorher herunterladen zu müssen und funktioniert wie in Abschnitt 4.2.8 beschrieben. Dabei können mehrere Datensätze ausgewählt und nacheinander mittels der Schaltflächen der Detailansicht eingesehen werden.

Bearbeitung von Datensätzen Die in Abbildung 6.7 zu sehende Bearbeitungsansicht, bietet einem Benutzer die Möglichkeit Datensätze zu bearbeiten. Fällt einem Benutzer bei der Nutzung des Systems ein Fehler beziehungsweise eine Inkonsistenz in einem Datensatz auf, kann er von der Listenansicht, der Zuletztansicht sowie der Detailansicht die Bearbeitungsansicht aufrufen. Die Bearbeitungsansicht funktioniert wie in Abschnitt 4.2.9 beschreiben.

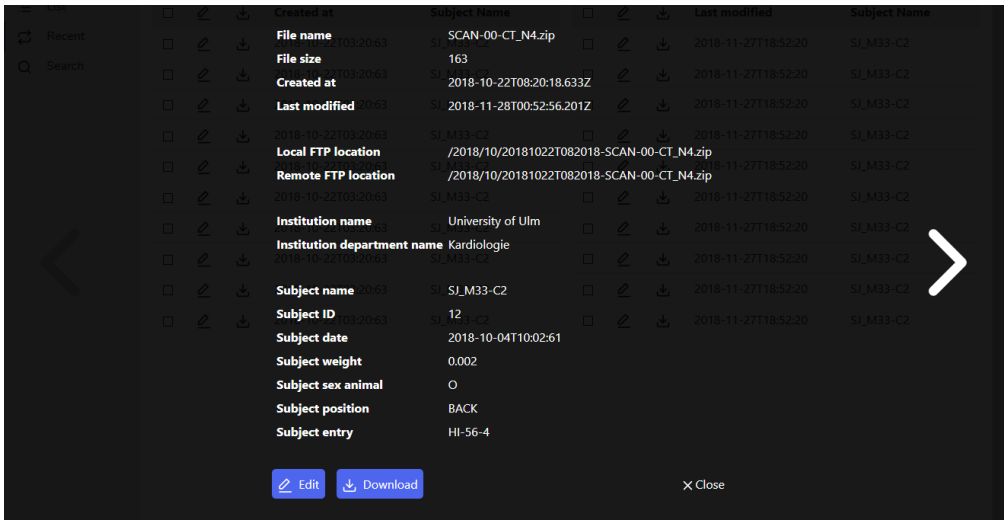


Abbildung 6.6: In dieser Abbildung ist ein Ausschnitt der Detailansicht der Webanwendung. Der Pfeil am rechten Rand indiziert dem Benutzer, dass in diesem Fall noch weitere Datensätze zur Verfügung stehen.

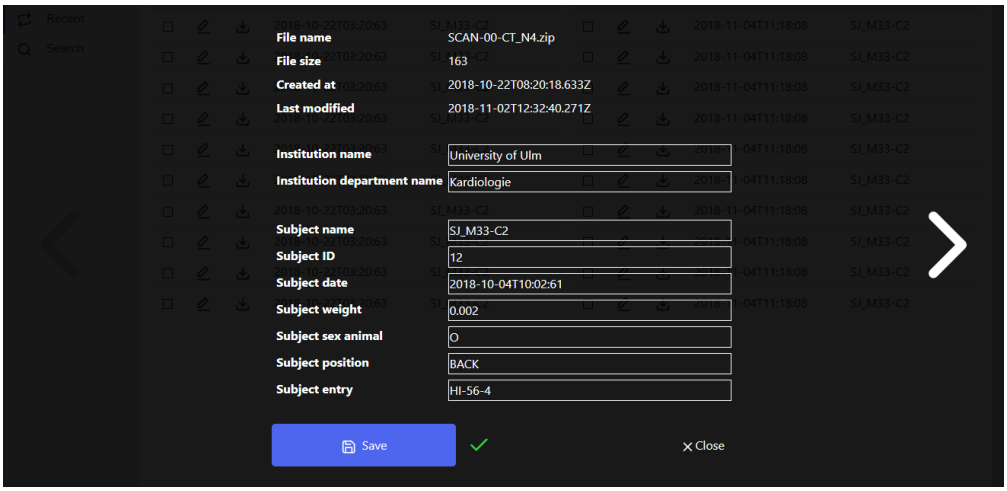


Abbildung 6.7: Diese Abbildung zeigt einen Ausschnitt der Bearbeitungsansicht der Webanwendung. Der Pfeil am rechten Rand zeigt dem Benutzer wie auch in Abbildung 6.6, dass weitere Datensätze zur Bearbeitung bereit stehen. In diesem Fall wurde der Datensatz bereits bearbeitet und erfolgreich gespeichert, was anhand des grünen Hakens neben dem Speichern-Button erkennbar ist.

7 Fazit und Ausblick

Diese Arbeit befasste sich mit der Konzeption und Implementierung einer Benutzerschnittstelle zur Archivierung medizinischer Forschungsdaten. Im Rahmen dieser Arbeit wurden eigens für die in Kapitel 1 genannte Problemstellung Konzepte zur Lösung erarbeitet und anschließend als plattformunabhängige Webanwendung implementiert. Alle in Abschnitt 4.1 genannten Anforderungen der Auftraggeber wurden vollständig in das System integriert und wie in Abschnitt 4.2 beschrieben miteinander verknüpft. Mit dem Abschluss dieser Arbeit beginnt die Verwendung des Systems in der medizinischen Forschung, nachdem bereits während der Entwicklung eine Testphase zur Evaluierung einzelner Funktionen stattgefunden hat.

Zu dem in dieser Arbeit entwickelten Konzept und der dazugehörigen Implementierung, bestehen am Ende dieser Arbeit noch Möglichkeiten zur Erweiterung des entwickelten Systems.

Eine Möglichkeit zur Erweiterung stellt ein Dashboard für Administratoren dar, um Benutzer und Datensätze besser verwalten zu können. Zum Umfang dieser Verwaltungsmöglichkeiten, könnten das Anlegen, Bearbeiten oder Löschen von Benutzern und Datensätzen zählen. Dies ist zum Ende dieser Arbeit über das durch *Django* zur Verfügung gestellte *Administratoren Interface* möglich. Da es sich hierbei um eine generelle Lösung für ein breites Anwendungsgebiet handelt, könnte ein auf die oben genannten Anwendungsfälle spezifisch zugeschnittenes *Administratoren Dashboard* die Verwaltung des Systems erweitern.

Die im Rahmen dieser Arbeit entwickelte Anwendung, wurde in direkter Zusammenarbeit mit dem Auftraggeber entwickelt. Hierbei wurden Design und Konzept speziell auf die Wünsche und Anforderungen angepasst. Um das entwickelte Interaktionskonzept auch für neue Nutzergruppen zu evaluieren, könnte die Anwendung nach einer Nutzerstudie und dem resultierenden Feedback der Probanden in Konzept und Design für eben jene überarbeitet werden.

Abbildungsverzeichnis

Literaturverzeichnis

[1] https://docs.djangoproject.com/en/2.1/topics/templates/, abgerufen am 18.11.2018.

[2] https://docs.djangoproject.com/en/2.1/topics/http/urls/, abgerufen am 18.11.2018.

[3] https://docs.djangoproject.com/en/2.1/topics/security/, abgerufen am 18.11.2018.

[4] I. S. 1003.1-2017, "Regular Expression, The Open Group Base Specifications Issue 7, 2018 edition," Tech. Rep., 2017, abgerufen am 25.11.2018. [Online]. Available: http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap09.html

[5] M. Behlse, BitGo, R. Peon, Google inc., M. Thomson Ed., and Mozilla, "Hypertext Transfer Protocol Version 2 (HTTP/2)," Internet Requests for Comments, Internet Engineering Task Force (IETF), RFC 7540, Mai 2015, abgerufen am 26.10.2018. [Online]. Available: https://tools.ietf.org/html/rfc7540

[6] T. Berners-Lee, W3C/MIT, R. Fielding, Day Software, L. Masinter, and Adobe Systems, "Uniform Resource Identifier (URI): Generic Syntax," Internet Requests for Comments, Network Working Group, RFC 3986, Januar 2005, abgerufen am 27.10.2018. [Online]. Available: https://tools.ietf.org/html/rfc3986

[7] A. Bhushan and MIT-MAC, "THE FILE TRANSFER PROTOCOL," Internet Requests for Comments, Network Working Group, RFC 354, Juli 1972, abgerufen am 17.11.2018. [Online]. Available: https://tools.ietf.org/html/rfc354

[8] S. Bradner and Harvard University, "The Internet Standards Process – Revision 3 - Section 10," Internet Requests for Comments, Network Working Group, RFC 2026, Oktober 1996, abgerufen am 17.11.2018. [Online]. Available: https://tools.ietf.org/html/rfc2026#section-10

[9] M. Cohn, *User stories applied: For agile software development*. Addison-Wesley Professional, 2004.

[10] M. Dahm, *Grundlagen der Mensch-computer-interaktion*. Pearson Studium München, 2006.

[11] DIN Norm, “Din en iso 9241-110 2006,” *Ergonomie der Mensch-System-Interaktion-Teil*, vol. 110, pp. 9241–110, abgerufen am 22.11.2018.

[12] S. Faulkner, A. Eicholz, T. Leithead, A. Danilo, and S. Moon, “HTML 5.2 - W3C Recommendation,” Tech. Rep., Dezember 2017, abgerufen am 17.11.2018. [Online]. Available: https://www.w3.org/TR/2017/REC-html52-20171214/

[13] D. Kristol, Bell Laboratories, Lucent Technologies , L. Montulli, and Epinions.com inc., “HTTP State Management Mechanism,” Internet Requests for Comments, Network Working Group, RFC 2965, Oktober 2000, abgerufen am 20.11.2018. [Online]. Available: https://tools.ietf.org/html/rfc2965

[14] E. Rescorla and RTFM inc., “HTTP Over TLS,” Internet Requests for Comments, Network Working Group, RFC 2818, Mai 2000, abgerufen am 26.10.2018. [Online]. Available: https://tools.ietf.org/html/rfc2818

[15] A. van Kesteren, J. Aubourg, J. Song, and H. R. M. Steen, “XMLHttpRequest Level 1,” Tech. Rep., Oktober 2016, abgerufen am 17.11.2018. [Online]. Available: https://www.w3.org/TR/XMLHttpRequest/

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit mit dem Titel:

MedInA: Ein Informationssystem zur Archivierung von Daten aus bildgebenden Verfahren der Medizin

bis auf die offizielle Betreuung selbstständig und ohne fremde Hilfe angefertigt habe und die benutzten Quellen und Hilfsmittel vollständig angegeben sind. Aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind jeweils unter Angabe der Quelle als solche kenntlich gemacht. Ich erkläre außerdem, dass die vorliegende Arbeit entsprechend den Grundsätzen guten wissenschaftlichen Arbeitens gemäß der "Satzung der Universität Ulm zur Sicherung guter wissenschaftlicher Praxis" erstellt wurde.

Ulm, den 01.12.2018 .....

Robin Stüdle