HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

ImageFinder
reformierte kirche horgen

ImageFinder

Ein Google Bildersuche-Klon, der Privatsphäre bietet

Bachelorarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Frühjahrssemester 2019

Autoren:	Martin Odermatt, Tobias Saladin
Betreuer:	Prof. Oliver Augenstein
Projektpartner:	Evangelisch-reformierte Kirche Horgen
Experte:	Reto Bättig
Gegenleser:	Prof. Olaf Zimmermann

Inhaltsverzeichnis

Abstract

Die Google-Cloud unterstützt die Möglichkeit, hochgeladene Bilder nutzerfreundlich und effizient zu durchsuchen. Allerdings führt das Hochladen der Bilder auf Cloud-Dienste zu Einbussen bei der Privatsphäre, was in vielen Fällen nicht hinnehmbar ist. Ausgehend von der Studienarbeit „Bildklassifikation mit Hilfe eines neuralen Netzes“ soll in der vorliegenden Bachelorarbeit in Zusammenarbeit mit der Evangelisch-reformierten Kirche Horgen ein datenschutzfreundlicher Google Bildersuche-Klon entwickelt werden. Die Software soll bei der Kirche produktiv zum Einsatz kommen und die Mitarbeitenden beim Durchsuchen ihrer Bildersammlung unterstützen.

Die Software wird in einem Client-Server-Modell konzipiert, welches bei der Evangelisch-reformierten Kirche Horgen auf demselben System installiert ist. Mit Tensorflow/Python als weit verbreitete und bewährte Kombination für Machine Learning Anwendungen in Kombination mit einer modernen Weboberfläche, ist die Software zukünftig modular erweiterbar und erlaubt verschiedene Deployment-Varianten. Lokale Bilder werden auf der Weboberfläche selektiert und über die Schnittstelle der lokal installierten Serverkomponente übergeben. Es besteht die Möglichkeit, für jedes Bild die Metadaten Fotograf, Tags sowie Erstellungsdatum festzulegen, die als zusätzliche Filter bei der Suche nach Bildern verwendet werden können. Diese werden in einer relationalen Datenbank persistiert. Beim Übertragen der Bilder auf den Server werden Feature-Vektoren berechnet. Je ähnlicher sich zwei Vektoren sind, desto ähnlicher sind auch die dazugehörigen Bilder. Die Feature-Vektoren werden in einer Baumstruktur abgelegt, was ein rasches Auffinden von ähnlichen Bildern ermöglicht. Die Feature-Vektoren werden ausserdem dazu verwendet, um dem Benutzer bei der Startseite der Weboberfläche eine variantenreiche Auswahl anzuzeigen. Dazu werden möglichst unterschiedliche Feature-Vektoren bestimmt. Darüber hinaus werden die Feature-Vektoren für das Gruppieren sehr ähnlicher Bildern verwendet.

Eine Bildersuche erfolgt entweder über die Auswahl von lokal gespeicherten Referenzbildern oder mit einer textbasierten Suche. Damit eine Bildersuche mittels Texteingabe durchgeführt werden kann, kommuniziert der lokale Server mit der Google Custom Search API und beschafft sich von Google zum Text passende Referenzbilder aus dem Internet In beiden Fällen wird die lokale Bildersammlung mit den erhaltenen Referenzbildern nach ähnlichen Bildern durchsucht. Bei lediglich einem Referenzbild wird der Nearest Neighbor Search (NNS) Algorithmus angewendet und die nächsten Nachbarn aus der Baumstruktur geladen. Bei mehreren Referenzbildern garantiert ein Algorithmus, welcher die für jedes Referenzbild gefundenen Treffer gegenseitig vergleicht und nach dem Rang sortiert, die gleichmässige Gewichtung aller Merkmale aus den Referenzbildern.

Die entwickelte Software bietet eine mögliche Alternative zu den Diensten der gängigen Cloud-Anbieter, ohne Einbussen beim Datenschutz oder der Privatsphäre hinnehmen zu müssen. Die grösste Limitation der entwickelten Software stellt die fehlende Eignung des genutzten neuronalen Netzes für die Gesichtserkennung dar. Aufgrund der Rückmeldung des Kunden erscheint die Implementierung einer Gesichtserkennung in einem nächsten Schritt als sinnvoll.

Management Summary

Ausgangslage

Die Evangelisch-reformierte Kirche Horgen ist im Besitz einer unstrukturierten Bildersammlung, die eine Suche nach Bildern mit bestimmten Merkmalen, wie zum Beispiel „Osterfest" oder „Kirchenturm", nicht unterstützt. Die Bilder unterliegen dem Datenschutz, wodurch das Hochladen der Bilder in externe Cloud-Dienste nicht gestattet ist. Das Ziel dieser Bachelorarbeit ist die Entwicklung eines marktreifen Produktes mit einer Funktionalität, vergleichbar mit der Bildersuche von Google. Dafür wurden die Konzepte aus der vorhergehenden Studienarbeit wie das von Google frei zur Verfügung gestellte Neurale Netzwerk und die gewonnenen Erkenntnisse aus dem Siamesischen Netzwerk, in der Bachelorarbeit verwendet.

Vorgehen und Technologien

Die Software wurde mit den Technologien TensorFlow/Python als weit verbreitete und bewährte Kombination für Machine Learning Anwendungen in einem Client-Server-Modell konzipiert. Dies erlaubt einerseits die Verwendung von modernen Webtechnologien und andererseits ein modularer Aufbau der Software. Client als auch Server wurden in Form eines Windows Services an einem Arbeitsplatz der Evangelisch-reformierten Kirche Horgen installiert.

Um mögliche Anwendungsfälle zu erkennen und besser zu verstehen, wurde die Software in enger Zusammenarbeit mit der Evangelisch-reformierten Kirche Horgen entwickelt. Dies erlaubte ebenfalls die Durchführung von Tests, die eine einfache Bedienbarkeit der Weboberfläche garantieren.

Ergebnisse

Im Zuge der Bachelorarbeit ist eine marktreife Software entstanden, die zu den Diensten der gängigen Cloud-Anbieter eine gleichwertige Alternative darstellt, ohne gegen die Datenschutzrichtlinien zu verstossen.

Bilder können über eine moderne Weboberfläche importiert und lokal gespeichert werden. Um ein Bild zu suchen, besteht die Möglichkeit, ein oder mehrere angezeigte Bilder auszuwählen und so eine Suche nach möglichst ähnlichen Bildern zu starten. Die Ähnlichkeitsprüfung der Bilder wird mittels Siamesischem Netzwerk durchgeführt. Alternativ kann eine textbasierte Suche verwendet werden. Dabei werden Suchresultate von Google als Referenzbilder für die lokale Suche eingesetzt. Als Resultat werden die Bilder nach der höchsten Ähnlichkeitsübereinstimmung sortiert und entsprechend angezeigt. Sehr ähnliche Bilder werden gruppiert. Die Suchresultate lassen sich optional nach verschiedenen Kriterien wie Fotograf, Datum und Event filtern.

Das Resultat wurde von der Evangelisch-reformierten Kirche Horgen als sehr innovativ und gelungen bewertet.

Ausblick

Die grösste Limitation der entwickelten Software stellt die fehlende Eignung des genutzten Neuronalen Netzes für die Gesichtserkennung dar. Aufgrund der Rückmeldung der Mitarbeitenden der Evangelisch-reformierten Kirche Horgen erscheint die Implementierung einer Gesichtserkennung in einem nächsten Schritt als sinnvoll.

Abbildungsverzeichnis

Tabellenverzeichnis

1 Ausgangslage

Das Ziel der vorliegenden Bachelorarbeit ist die Entwicklung eines Google-Bildersuche Klon. Wie bei Google sollen Bilder mit der Eingabe von Text oder ähnliche Bilder zu ausgewählten Referenzbildern gefunden werden können. Das Produkt soll jedoch auch für Bilder, welche Datenschutzrichtlinien unterliegen, geeignet sein. Die Bilder sollen somit nicht veröffentlicht und nur lokal verwendet werden.

1.1 Zielgruppe

Die fertige Applikation für die Suche nach Bildern in einer lokalen Bildersammlung richtet sich an die Mitarbeitenden der Evangelisch-reformierten Kirche Horgen.

1.2 Studienarbeit 2018: Machbarkeitsanalyse

Die Bachelorarbeit baut auf den Erkenntnissen, der durch die Autoren verfassten Studienarbeit „Bildklassifikation mit Hilfe eines neuralen Netzes", auf [1]. In dieser wurde ein Prototyp entwickelt, welcher bei der Evangelisch-reformierten Kirche Horgen zum Einsatz kam.

1.3 Analyse des Proof of Concepts aus der Studienarbeit

Der Proof of Conepts (POC) sollte analysieren, ob eine Durchsuchung der Bildersammlung der Evangelisch-reformierten Kirch Horgen mit Schlüsselwörtern oder anhand ähnlicher Bilder möglich ist, ohne dabei gegen den Datenschutz zu verstossen. Dazu musste zuerst die Ähnlichkeit zweier Bilder bestimmt werden.

Vorgehen Bildvergleiche
Mit dem Convolutional Neural Network, Inception-v3 [1], wurde zu jedem Bild ein Feature-Vektor berechnet. Durch die Vergleiche der Feature Vektoren ist es möglich, die Ähnlichkeit der Bilder zu messen. Inception wurde auf 1000 Kategorien trainiert und gibt zu jedem Bild die prozentuale Wahrscheinlichkeit der Zugehörigkeit einer Kategorie an. Durch das Entfernen des letzten Layers des Netzwerkes erhält man statt den Wahrscheinlichkeiten die Feature-Vektoren der Bilder.
Anschliessend wurde eine Siamesische Architektur [2] mit zwei identischen Inception-v3 Netzwerken erstellt. Im Loss Layer der Architektur wurde mit der Kosinus-Metrik die Ähnlichkeit zweier Bilder bestimmt. Weiterführende Details, wie zum Beispiel die Berechnung des Schwellwertes, welcher zwei Bilder als ähnlich identifiziert, sind aus der Studienarbeit zu entnehmen.

Textbasierte Bildersuche
Um eine textbasierte Bildersuche zu ermöglichen wurde die über GitHub verfügbare Bibliothek, google-images-download [3], verwendet. Diese verwendet jedoch nicht die offizielle Google Custom Search API [4] und garantiert somit keine Lauffähigkeit. Über die Eingabe eines Suchbegriffes wurden von Google gefundene Bilder für die Suche nach ähnlichen Bildern verwendet.

Fertiges Produkt
Die zuvor beschriebene Architektur wurde durch eine mit der PyQt5 Bibliothek [5] erstellten Desktop-Applikation ergänzt. Die grafische Benutzeroberfläche ist nicht für den produktiven Einsatz gedacht, sondern für die Überprüfung der Anforderungen der Studienarbeit. Der Funktionsumfang und die Bedienung entsprachen lediglich der Qualität eines Prototypen.

Erkenntnisse
Aus der Studienarbeit ging hervor, dass sich mit dem Siamesischen Netzwerk die Bildersammlung durchsuchen lässt. Verbesserungspotenzial konnten aufgrund der gemachten Tests bei nachfolgenden Punkten festgestellt werden:

- Nicht mehr benötigte Bilder werden nicht gelöscht und verbleiben auf dem Laufwerk.
- Die Suchresultate mit mehreren Referenzbildern waren unbefriedigend und ergaben keinen Mehrwert.
- Das Arbeiten mit dem Prototypen fühlt sich nicht flüssig an. Arbeitsschritte, wie zum Beispiel das Suchen nach ähnlichen Bildern oder das Aufstarten der Software, benötigen viel Zeit und mindern das Benutzererlebnis.
- Das User Interface entspricht nicht den heutigen Standards.
- Die Google Suche benutzt nicht die offizielle Google Search API und ist daher nicht zuverlässig.

Rückmeldung der Evangelisch-reformierten Kirche Horgen
Nach der Fertigstellung des Prototyps konnte dieser den Mitarbeitenden der Evangelisch-reformierten Kirche Horgen präsentiert werden. Die Rückmeldungen zeigten, dass das geplante Programm Potenzial hat. Die Ergebnisse der Suchanfragen seien brauchbar und würden die Bildersuche unterstützen. Folgende Punkte wurden als mögliche Verbesserungen genannt:

- Bei einer Suche sollen als Resultat mehr als nur die besten 40 Treffer angezeigt werden.
- Die Aktualität der Bilder soll eingegrenzt werden können, wie zum Beispiel nur Bilder aus dem Jahr 2018.
- Bildduplikate sollen entfernt werden.
- Bildinformationen wie Fotograf, Aufnahmedatum oder Anlass sollen abrufbar sein. Zudem soll nach diesen Kriterien gefiltert werden können.
- Beim Import eines Bildes soll das Datum nach Möglichkeit erkannt werden.
- Bilder sollen mit Schlagwörtern gekennzeichnet werden können.

2 Qualitätsziele

In diesem Kapitel werden die Anforderungen, Risiken sowie Problemstellungen genauer erläutert.

2.1 Funktionale Anforderungen

Die funktionalen Anforderungen sind in Abbildung 1 dargestellt und zeigen den Umfang (Scope) des Systems. Als Primary Actor ist eine Benutzerin oder ein Benutzer der Evangelisch reformierten Kirche Horgen festgelegt. Die Anforderungen werden nach Larman [6] die Use Cases mit Precondition, Basic Flow und Postcondition genauer erläutert.

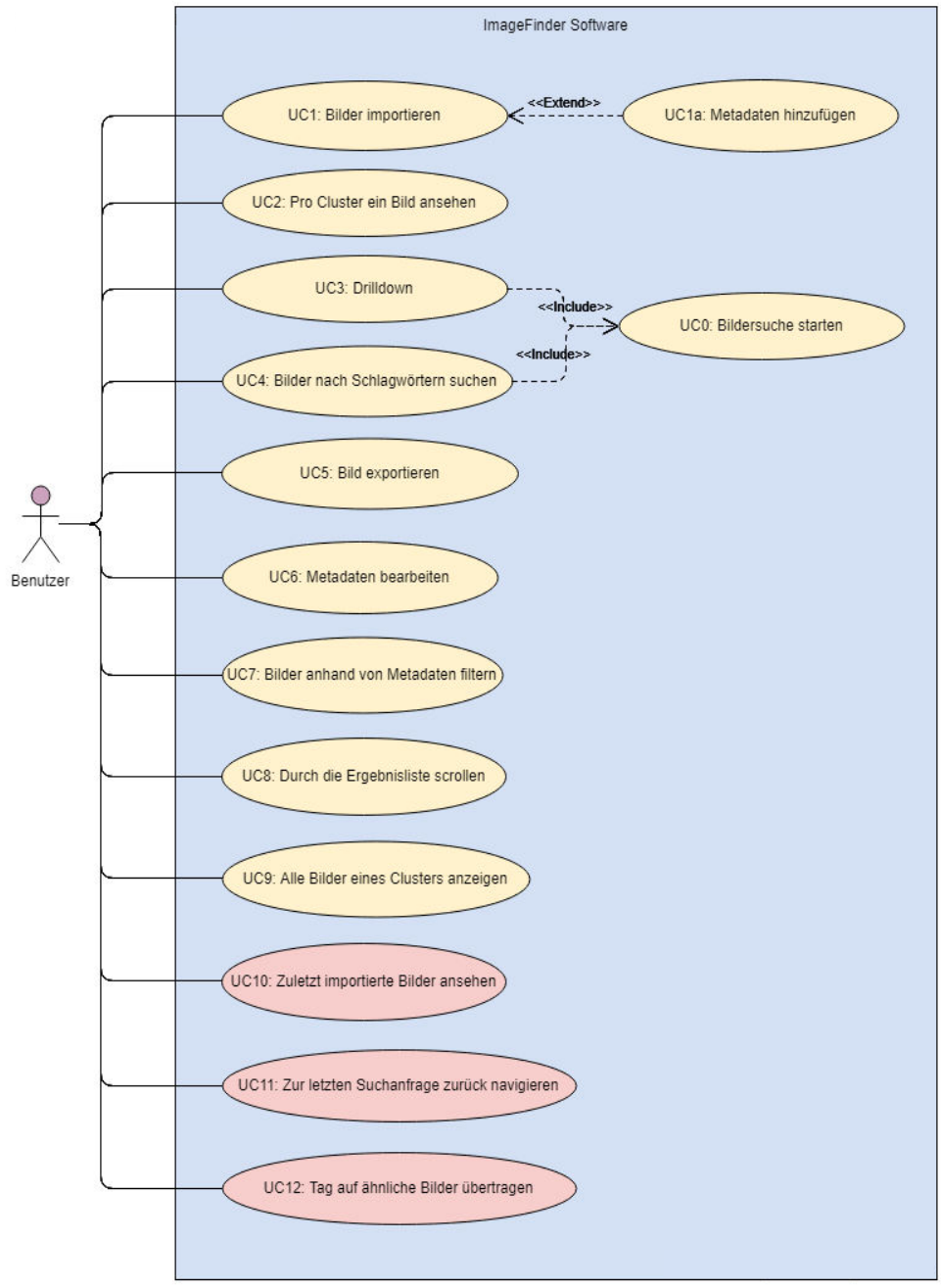


Abbildung 1: Funktionale Anforderungen

Anmerkung. Eigene Darstellung.

Die Beschreibungen in Tabelle 1 sind in „Essential Style” verfasst und somit von der Realisierung sowie einer Beschreibung einer grafischen Benutzeroberfläche losgelöst. Im Fokus stehen Ziele und Absichten.

Tabelle 1: Funktionale Anforderungen

UC	Titel	Beschreibung
UC0	Bildersuche starten	**Basic Flow**: 1. Ein Suchlauf für die ähnlichsten Bilder ausgehend von einem oder mehreren Referenzbildern wird von UC3 oder UC4 ausgelöst. 2. Die ähnlichsten Bilder werden dem Benutzer angezeigt. 3. Weitere erscheinen gruppiert in sogenannten „Clustern”.
UC1	Bilder importieren	**Basic Flow**: 1. Die zu importierenden Bilder werden ausgewählt. Das System beginnt die Bilder zu importieren. 2. Das Erstellungsdatum eines Bildes wird automatisch als Metainformation vom System persistiert. UC1a: Zu einem Bild können zusätzliche Metadaten wie Fotograf und Tags hinzugefügt werden. **Alternate Scenarios:** 2a: Es wurde vom Benutzer keine Tags und kein Fotograf angegeben. Ein Dialog zeigt eine entsprechende Meldung an. Der Import kann anschliessend abgebrochen oder fortgesetzt werden. 2b: Ein zu importierendes Bild ist beschädigt und kann nicht verarbeitet werden. Es wird übersprungen. 2c: Ein Benutzer importiert ein Bild, das bereits zu einem anderen Zeitpunkt importiert worden ist. Das System persistiert dieses nicht erneut. 3a: Das Erstellungsdatum kann vom System nicht ausgelesen werden. Es wird das aktuelle Datum hinzugefügt. **Optional Scenario:** 2d: Das System vergleicht das zu importierende Bild mit den bereits bestehenden Bildern und schlägt ein Tag vor 2e: Das System kennt den Kirchenkalender und schlägt den letzten Anlass als Tag vor. **Postconditions**: • Das Bild sowie die dazugehörigen Metadaten sind persistiert. • Die Feature-Vektoren der Bilder sind berechnet und persistiert. • Statische Mini-Cluster für die Übersicht sind neu berechnet und persistiert.
UC2	Pro Cluster ein Bild ansehen	**Precondition:** Das System verfügt bereits über Bilder.

		Basic Flow: 1. Ein Benutzer löst die Aktion „gruppieren” aus. 2. Ein Bild aus jedem Cluster wird angezeigt. D.h. ähnliche Bilder werden in einen Cluster gelegt. 3. Ein Cluster ist als solcher erkennbar. **Alternate Scenario:** 3a: Ein Bild konnte keinem Cluster zugeordnet werden. Es wird deshalb nicht als Cluster (mit nur einem Bild) angezeigt.
UC3	Drilldown	**Precondition:** Das System verfügt bereits über Bilder. **Basic Flow**: 1. Der Benutzer wählt ein Referenzbild aus. 2. Die Aktion des UC0 wird ausgelöst. **Alternate Scenario** 1a: Es werden mehrere Bilder ausgewählt.
UC4	Bilder nach Schlagwörtern suchen	**Basic Flow:** 1. Der Benutzer gibt ein oder mehrere Schlagwörter ein. 2. Die Aktion des UC0 wird ausgelöst. 3. Google Bilder erscheinen und sind als solche gekennzeichnet. **Alternate Scenario:** 2a: Der eingegebene Suchbegriff führt zu keinem Resultat. Ein Dialogfeld mit einem Hinweis erscheint.
UC5	Bilder exportieren	**Precondition:** Das System verfügt über mindestens ein Bild. **Basic Flow:** 1. Ein Benutzer selektiert ein Bild 2. Das Bild kann in Originalgrösse exportiert werden. **Alternate Scenario** 2a: Das Bild kann in den Grössen small, medium und large exportiert werden.
UC6	Metadaten bearbeiten	**Precondition:** Das System verfügt bereits über Bilder. **Basic Flow:** 1. Der Benutzer wählt ein Bild aus 2. Zum angezeigten Bild können die Metadaten: Erstellungsdatum, Tags, Fotograf bearbeitet werden. 3. Benutzer speichert die neue Eingabe. **Alternate Scenarios:** 2a: Der Benutzer kann die Änderung der Metadaten auf mehrere ausgewählten Bilder anwenden. **Postcondition**: Das System hat die Änderung der Metadaten persistiert.

UC7	Bilder anhand von Metadaten filtern	**Precondition**: Ein Suchresultat liegt vor. **Basic Flow:** 1. Ein Benutzer kann in einen Filter die Tags, Fotograf und Erstellungszeitraum eingeben. 2. Das Suchresultat wird gemäss dem Filter eingeschränkt. **Alternate Scenario:** 2a: Es wird zu einem Filter kein Suchresultat gefunden. In diesem Fall wird dem Benutzer ein Hinweis angezeigt.
UC8	Durch die Ergebnisliste scrollen	**Precondition:** Ein Suchresultat liegt vor. **Basic Flow:** 1. Ein Benutzer kann in der Ergebnisliste scrollen. 2. Weitere Suchresultate werden angezeigt. **Alternate Scenarios:** 2a: Ein Suchresultat muss erst noch geladen werden. In diesem Fall wird dies dem Benutzer signalisiert. 2b: Es gibt keine weiteren Suchresultate mehr.
UC9	Alle Bilder eines Clusters anzeigen	**Precondition:** Ein Suchresultat mit einem Cluster liegt vor. **Basic Flow:** 1. Der Benutzer wählt einen Cluster aus 2. Die Aktion „aufklappen” wird vom Benutzer ausgelöst 3. Alle Bilder, die sich in diesem Cluster befinden, werden angezeigt
UC10	Zuletzt importierte Bilder ansehen	**Precondition:** Es wurden bereits erfolgreich Bilder importiert. **Basic Flow:** 1. Benutzer wählt die Ansicht zur Import-History 2. Zuletzt importierte Bilder werden absteigend geordnet nach Datum angezeigt **Alternate Scenario:** Precondition ist nicht erfüllt. In diesem Fall erscheint ein Hinweis.
UC11	Zur letzten Suchanfrage zurück navigieren	**Precondition:** Das System verfügt bereits über Bilder. Der Benutzer hat mindestens zwei Suchen ausgelöst. **Basic Flow:** 1. Benutzer löst die Aktion „zurück” aus 2. Das Resultat der vorherigen Suche wird angezeigt. **Alternate Scenario:** Precondition wurde nicht erfüllt. Es gibt noch keine vorherige Resulate. In diesem Fall soll es für den Benutzer nicht möglich sein, die Aktion auszulösen.

UC12	Tag auf ähnliche Bilder übertragen	**Precondition:** Es existieren bereits getaggte Bilder. **Basic Flow**: 1. Benutzer wählt ein Tag aus 2. Das System zeigt alle Bilder an, die zu diesem Tag passen 3. Benutzer speichert das Tag für die angezeigten Bilder **Alternate Scenario**: 3a: Benutzer selektiert von den angezeigten Bildern, welche das Tag erhalten sollen **Postcondition**: Bilder verfügen über das neue Tag

Anmerkung. Eigene Darstellung.

2.2 Nicht-funktionale Anforderungen

Nachfolgend sind in Tabelle 2 die Nichtfunktionalen Anforderungen an die Applikation aufgelistet. Für die Klassifikation wurde das von Hewlett-Packard publizierte „FURPS+" Akronym verwendet. [7]

Tabelle 2: Nicht-funktionale Anforderungen

#	Titel	FURPS+	Unterkategorie	Beschreibung
NFR1	Global Exception-Handler	Functionality	Logging	Nicht behandelte Fehler im Python-Backend sollen in einem globalen Exception-Handler abgefangen und geloggt werden.
NFR2	OWASP TOP 10	Functionality	Security	Der Source Code soll die OWASP Top 10 2017 [8] Kriterien erfüllen, bzw. so weit wie möglich automatisiert nach diesen Kriterien getestet werden.
NFR3	UI Design	Usability	Aesthetics	Die Weboberfläche soll modern (Material Design Standards[9]) gestaltet werden.
NFR4	Benutzerführung	Usability	Understandability	Die Benutzerführung soll möglichst intuitiv sein. Hierzu sollen mit Testpersonen gängige Szenarien wie Bilder importieren (UC1), Bilder nach Schlagwörtern suchen (UC4) oder Bilder anhand von Metadaten filtern (UC7) durchgespielt werden und Schwierigkeiten bei der Verständlichkeit (Auffinden von Buttons, unklares User-Feedback) protokolliert und entsprechende Massnahmen getroffen werden.
NFR5	Response-Time „/”	Performance	Response-Time	Die Response-Time für eine Anfrage auf die Hauptseite „/” soll im besten Fall unter einer halben Sekunde, im Normalfall eine Sekunde und im Höchstfall nicht länger als vier Sekunden dauern. Dies gemessen bei einem Single-User, d.h. keine Concurrent-Requests.
NFR6	Nearest Neighbor Search	Performance	Response-Time	Das Auffinden von 10 benachbarten Bilder (Nearest Neighbors Algorithmus) zu einem Referenzbild soll im besten Fall unter drei Sekunden, im Normalfall fünf Sekunden und im Höchstfall nicht länger als acht Sekunden dauern.
NFR7	Bildimport	Performance	Throughput	Der Import der Bildersammlung der Evangelisch-reformierten Kirche Horgen, mit insgesamt 14‘800 Bildern, soll mit dem im Anhang L beschriebenen Referenzsystem nicht länger als sechs Stunden dauern. Dies gemessen bei einem Single-User, d.h. keine Concurrent-Requests.

NFR8	Win 10	Supportability	Compatibility	Die Applikation soll mit Windows 10 betrieben werden können.
NFR9	Sprache	Supportability	Localizability	Die Benutzerführung der Weboberfläche soll in deutscher Sprache erfolgen.
NFR10	Log-Level	Supportability	Configurability	Das Logging im Backend soll mehrere Stufen wie Info, Debug oder Error unterstützen.
NFR11	Datenschutz	+	Licencing	Aus datenschutzrechtlichen Gründen dürfen die Bilder aus der Bilddatenbank nicht an Dritte weitergegeben werden und sind nur lokal zu speichern.
NFR12	Persistenz	+	Design Constraint	Die Persistenz der Metadaten der Bilder soll durch eine relationale Datenbank sichergestellt werden.
NFR13	Webapplikation	+	Design Constraint	Der während der Studienarbeit entwickelte Prototyp soll in eine Webanwendung überführt werden (Design-Constraint)
NFR14	Temporäre Daten	+	Design Constraint	Temporär zwischengespeicherte Bilder auf dem Filesystem, welche lediglich für Berechnungen verwendet werden, sollen nach jeder Bildersuche komplett gelöscht oder in den Papierkorb verschoben werden.
NFR15	Inception-v3	+	Implementation Constraint	Für die Feature-Berechnung aus den Bildern soll das Inception-v3[10] Netzwerk verwendet werden.
NFR16	Baumstruktur	+	Implementation Constraint	Die Nearest Neighbors (Matrix) sollen in eine Baumstruktur abgespeichert werden.
NFR17	Best lowest search rank first	+	Implementation Constraint	Die Suche mit mehreren Referenzbildern als Ausgangslage, soll nach dem „Best lowest search rank first" Algorithmus funktionieren.
NFR18	Google-Schnittstelle	+	Interface Constraint	Für die Beschaffung der Referenzbilder soll die offizielle Google Custom Search API verwendet werden[11].
NFR19	Code Formatting	+	Software Quality	Das Frontend sowie Backend sollen einem einheitlichen Code-Styleguide folgen. Ausgeschlossen sind Konfigurationsdateien.
NFR20	Testcoverage (Frontend)	+	Software Quality	Um eine marktreife Software zu entwickeln, soll das Frontend eine Testabdeckung von mind. 35% erreichen.
NFR21	Testcoverage (Backend)	+	Software Quality	Um eine marktreife Software zu entwickeln, soll das Backend eine Testabdeckung von mind. 85% erreichen.
NFR22	Agile Vorgehensweise	+	Project Management	Für das Projektmanagement soll eine agile Methode gewählt werden.

Anmerkung. Eigene Darstellung.

3 Projektmanagement

Eine vollständig (ideale) agile Vorgehensweise wie Scrum [12] kennt keine Requirements ausser die User Stories und keine geplante Architektur sowie Projektplan. Dies ist durch den knappen, strikt einzuhaltenden Zeitraum von 17 Wochen nicht optimal. Die Autoren haben sich deshalb für diese Bachelorarbeit für die Rational Unified Process (RUP) Methode entschieden. Dies ist eine semi-agile Arbeitsweise mit iterativen Teilen. Es wurden Meilensteine definiert, welche die Fertigstellung des Produktes garantieren sollen. Innerhalb der Construction-Phase wird agil gearbeitet. Die einzelnen, umzusetzenden Komponenten werden in Pakete gegliedert und priorisiert. Die Aufwandschätzung wird mittels Story-Points gemacht.

3.1 Meilensteine

Für die zeitliche Planung wurden Meilensteine definiert. Diese werden während des Projekts laufend geprüft und aktualisiert, um Verzögerungen frühzeitig erkennen und entsprechend reagieren zu können. In Tabelle 3 ist die Planung der Meilensteine aufgelistet.

Tabelle 3: Meilensteine

Phase	End-Datum	Nr.	Meilenstein	Beschreibung
Inception	27.02.2019	M1	End of Inception	Ziele, Vision definiert
				Prioritäten gesetzt
				Risiken definiert
				Entwicklungsprozess
				Vollständiger Projektplan (einzelne Arbeitspakete inkl. Aufwandschätzung)
Elaboration	06.03.2019	M2	Review Projektplan	Abnahme des Projektplanes
	06.03.2019	M3	Review Anforderungen	Abnahme der Anforderungen
	11.03.2019	M4	End of Elaboration	Use Cases im Format Brief
				Funktionale und nichtfunktionale Anforderungen
				Wireframes
				Domain Model
				Qualitätsmassnahmen festgelegt
				Prozesse für Entwicklung und Integration aufgesetzt
				Risiken minimiert
				Buildserver, Continuous Integration
				Richtlinien für Code und Dokumentation
Construction	13.03.2019	M5	Review End of Elaboration	Architekturprototyp, Durchstich
	18.03.2019	M6	Architektur	Architektur (Schichten, Server Logik, Persistenz, Deployment) erstellt
	20.03.2019	M7	Review Architektur	Abnahme der Architektur
	01.05.2019	M8	Review Qualität	Abnahme der Softwarequalität
	01.05.2019	M9	Feature Freeze	Es sollen keine weiteren Funktionalitäten mehr umgesetzt werden
	15.05.2019	M10	Anforderungen implementiert	Die Applikation soll fertig gestellt sein. Weitere Tests sollen bis zum Rollout durchgeführt werden
Transition	22.05.2019	M11	Installation beim Kunden	Überführung der Applikation (inkl. Anleitung) in den laufenden Betrieb
	05.06.2019	M12	Dokumentation	Fertigstellung der Dokumentation
	12.06.2019	M13	Abgabe Bachelorarbeit	Einreichung der Dokumentation, des Sourcecodes sowie dem Poster
	14.06.2019	M14	Poster	Ausstellung der Bachelorarbeit

Anmerkung. Eigene Darstellung.

3.2 Monitoring und Steuerung

Um während der Erarbeitungsphase den Zeitplan überwachen zu können, werden die Meilensteine in einem Diagramm laufend überwacht. Die Planung zu Arbeitsbeginn ist in Abbildung 2 ersichtlich. Wöchentlich werden in dieser Abbildung die erreichten Meilensteine eingetragen. Zeitliche Verschiebungen der Meilensteine werden so visualisiert. In der X-Achse sind die Intervalle eingetragen, in welchen die Abbildung aktualisiert wird. In der Y-Achse sind die Zeiten aufgelistet, an welchen ein Meilenstein erreicht werden soll. Wird beispielsweise ein Meilenstein später erreicht, verschiebt sich dieser in der Y-Achse nach oben. Die Grafik dient dazu auf unvorhergesehenes rechtzeitig reagieren und im Auge behalten zu können. Die fertige Auswertung kann aus Anhang C entnommen werden.

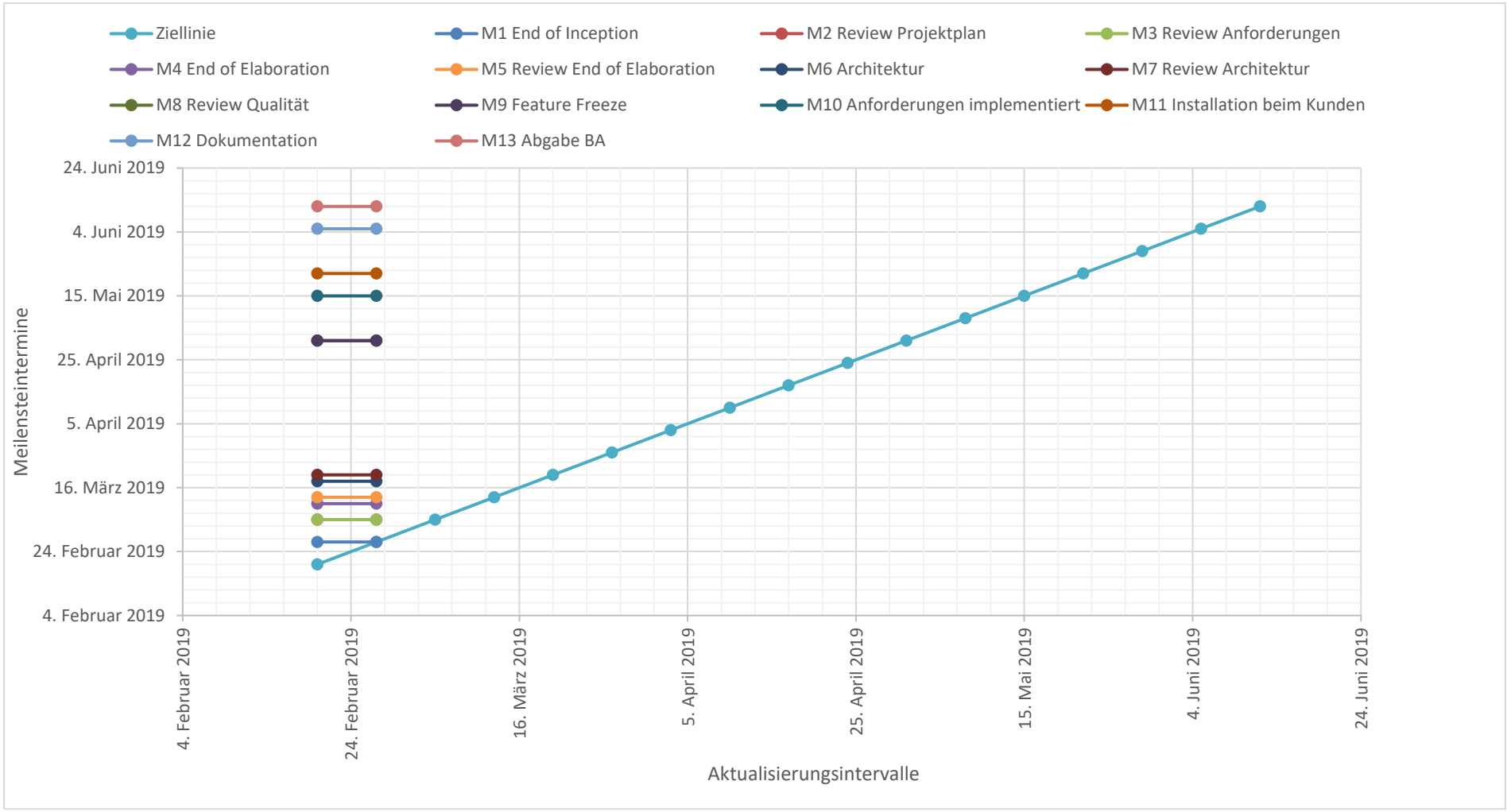


Abbildung 2: Diagramm für Zeitauswertung

Anmerkung. Eigene Darstellung.

3.3 Aufwandschätzung

Für die Erstellung der Aufwandschätzung wurde vorerst das gesamte Zeitbudget der Bachelorarbeit berechnet. Bezüglich der Ergebnisse wird auf Tabelle 4 verwiesen. Als Einheit in der Tabelle wurden Story-Points verwendet. Durch die Autoren wurde festgelegt, dass ein Story-Point einem Arbeitstag mit der Dauer von sieben Stunden entspricht.

Tabelle 4: Planung der Story-Points

Geplante Story-Points	**Story-Points**
Gesamte Anzahl zur Verfügung stehender Story-Points für beide Autoren	102
Fertigstellung der Dokumentation in den letzten zwei Wochen	6
Kickoff-Planung	2
Sitzungen mit Begleitperson	14
Sitzungen mit Kunden	3
Dokumentation während Entwicklungsphase	15
Vorbereiten der Architektur	8
Verbleibende Story-Points für die Umsetzung der Use Cases	**54**

Anmerkung. Eigene Darstellung.

Schätzung nach Storypoints der Use Cases
Die durch die Autoren definierten Use Cases wurden grob nach Aufwand geschätzt. Das Ergebnis ist Tabelle 5 zu entnehmen. Die Tabelle zeigt, dass bei einer durchschnittlichen Zeitberechnung 63 Story-Points für die Umsetzung der Use Cases benötigt werden. Dies sind neun Points mehr als den Autoren für die Arbeit zur Verfügung stehen.

Tabelle 5: Aufteilung der Use Cases nach Story-Points

UC	Titel	Best Case	Normal Case	Worst Case
UC0	Bildersuche starten (Lösung für ein und mehrere Referenzbilder)	9	12	15
UC1	Bilder importieren (Daten vorbereiten, persistieren, Duplikate entfernen)	4	7	9
UC2	Pro Cluster ein Bild ansehen (Clusterbildung)	2	4	6
UC3	Drilldown	4	6	9
UC4	Bilder nach Schlagwörtern suchen	2	3	6
UC5	Bilder exportieren	1	2	4
UC6	Metadaten bearbeiten (Metainformationen erfassen, persistieren, bearbeiten)	4	6	8
UC7	Bilder anhand von Metadaten filtern (Filteransicht, Bildergebnisse)	4	6	12
UC8	Durch die Ergebnisliste scrollen (Infinity scrolling, Paging)	3	4	8
UC9	Alle Bilder eines Clusters anzeigen (UI Lösung für Clusteranzeige)	2	3	6
UC10	Zuletzt importierte Bilder ansehen	1	2	4
UC11	Zur letzten Suchanfrage zurück navigieren (History, UI Anzeige)	3	6	8
UC12	Tag auf ähnliche Bilder übertragen	1	2	4
Summe		**44**	**63**	**98**

Anmerkung. Eigene Darstellung.

Konsequenz aus der Aufwandschätzung
Da für die Umsetzung aller Use Cases mehr Story-Points benötigt werden als den Autoren zur Verfügung stehen, wurden die Use Cases UC10, UC11 und UC12 vorerst zurückgestellt. Dadurch lassen sich rund 10 Storypoints einsparen. Somit ergibt sich ein kleiner Überschuss von einem Story-Point als zeitliche Reserve.

3.4 Risikomanagement

In Tabelle 6 wurden die relevanten Risiken zusammengetragen.

Tabelle 6: Risikomanagement

#	Kategorie	Beschreibung	Folgen	Eintrittswahrscheinlichkeit	Auswirkung	Vorbeugung	Verhalten bei Eintreten
R1	Security	Datenschutz wird nicht eingehalten	Rechtliche Konsequenzen	unwahrscheinlich	KO	Daten werden ausschliesslich lokal bearbeitet	Weiteres Vorgehen mit dem Dozenten besprechen
R2	Compatibility	Die Kombination der ausgewählten Frameworks und Libraries ist nicht kompatibel	Produkt kann nicht umgesetzt werden, bzw. es müssen Technologie wechsel vorgenommen werden	mässig	kritisch	Frühzeitig einen Durchstich erlangen	Auswahl der Komponenten neu evaluieren
R3	Usability	Das User Interface entspricht nicht den Erwartungen der nutzenden Personen	Schlechte User Experience	mässig	mässig	Regelmässiges Einholen von User-Feedback / Usability-Tests	Rücksprache mit den nutzenden Personen, um Optimierungen vornehmen zu können
R4	Scope	Scope kann in vorgegebener Zeit nicht erfüllt werden	Produkt kann nur eingeschränkt fertig gestellt werden	gross	kritisch	Regelmässige Meetings mit der Begleitperson, Überprüfung der Meilensteine, Priorisierung der Use Cases	Scope mit Begleitperson neu festlegen
R5	Scope	Neue Feature entsprechen nicht den Bedürfnissen der Evangelisch-reformierten Kirche Horgen	Unzufriedenheit seitens des Kundens	mässig	kritisch	Use Cases mit dem Kunden besprechen und Feedback einholen, ob das angedachte so gewünscht wird	Mit Kunde besprechen, welche Features als nächstes geplant sind

Anmerkung. Eigene Darstellung.

Risikomatrix

Die erfassten Risiken wurden in einer Risikomatrix aufgelistet, welche in Tabelle 7 ersichtlich ist. Es zeigt sich, dass mit den geplanten Vorbeugungen die Risiken tief gehalten werden können.

Tabelle 7: Risikomatrix

		Auswirkung			
Eintrittswahrscheinlichkeit		Gering (1)	Mässig (2)	Kritisch (3)	KO (4)
	Sicher (5)				
	Sehr gross (4)				
	Gross (3)			R4	
	Mässig (2)		R3	R2, R5	
	Unwahrscheinlich (1)				R1

Anmerkung. Eigene Darstellung.

4 Lösungskonzept

In diesem Kapitel soll auf ein Lösungskonzept eingegangen werden, bevor mit der Umsetzung begonnen wird.

4.1 Domainanalyse

In Abbildung 3 ist das konzeptionelle Modell der Domäne abgebildet.

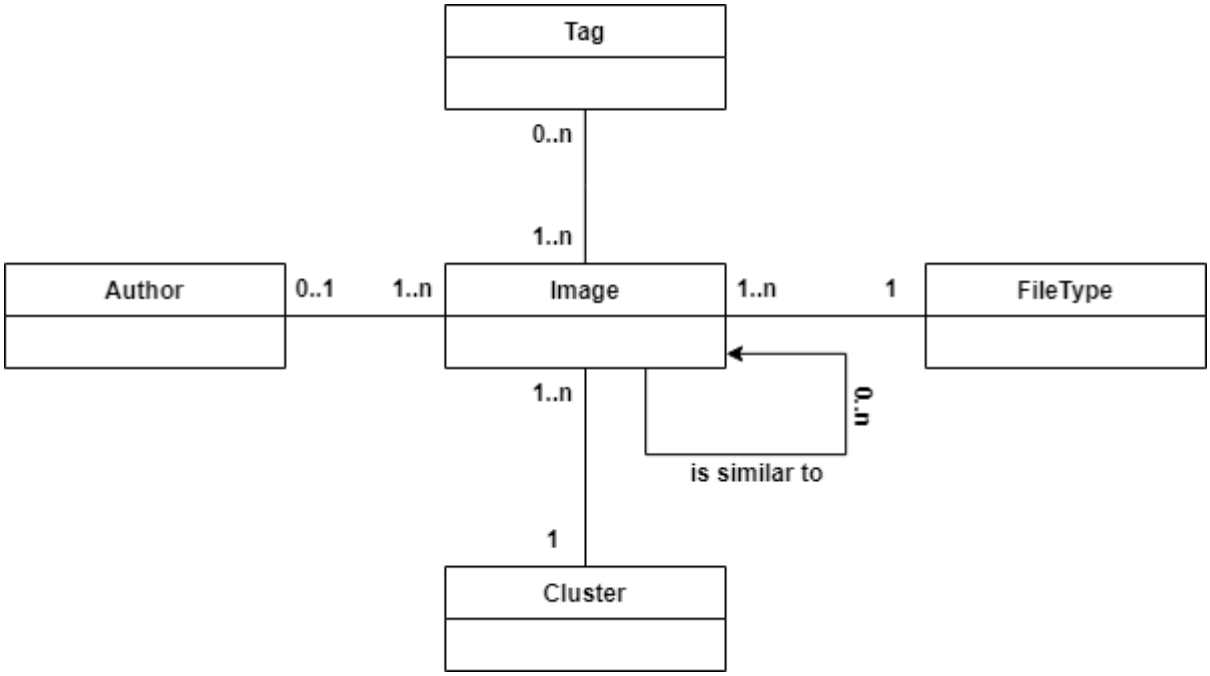


Abbildung 3: Domain Model

Anmerkung. Eigene Darstellung.

Im Mittelpunkt steht dabei ein „Image", welches keine oder mehrere Ähnlichkeiten zu einem anderen „Image" hat. Jedes „Image" hat optional einen „Author" im Sinne eines Urhebers und einen „FileType" wie zum Beispiel JPEG. Ein „Image" kann über keine oder mehrere Tags verfügen. Tags sind hierbei als Schlüsselwörter zu verstehen, die ein „Image" kennzeichnen.

4.2 Architektur-Evaluation

4.2.1 Web vs Desktop

Aus der Aufgabenstellung in Anhang A geht hervor, dass die zu entwickelnde Software in einer Webanwendung ausgeliefert werden soll. Damit TensorFlow als Machine Learning Library verwendet werden kann, soll die Architektur mit Python umsetzbar sein. Damit scheidet die Entwicklung einer Desktop-GUI-Anwendung mit Hilfe von Webtechnologien wie es zum Beispiel mit Electron [13] möglich ist, aus. Zwar wäre es grundsätzlich realisierbar, Python-Skripte aus dem JavaScript-Code auszuführen, dies jedoch zum Preis einer engen Kopplung.

Y-Template
Um die Entscheidungen zu dokumentieren, wurde im folgenden Abschnitt der Y-Ansatz gewählt, der erstmalig von Herrn Prof. Zimmermann bei der Saturn Konferenz in St. Petersburg (Florida, USA) im Jahr 2012 präsentiert wurde und an der HSR im Modul „Application Architecture“ gelehrt wird. [14]

4.2.2 Architektur-Entscheidung Y-Template

Im Kontext der Architektur für die Software ImageFinder der Evangelisch-reformierte Kirche Horgen,
um der nicht-funktionalen Anforderung „NFR15 – Inception-v3” entgegen zu treten,
haben sich die Autoren für den Einsatz einer Client-Server-Architektur entschieden und die Möglichkeit eine Desktop-GUI-Anwendung mit Webtechnologien zu erstellen, verworfen,
damit die Machine Learning Library TensorFlow verwendet werden kann und die grafische Benutzeroberfläche von dem in Python zu schreibenden Code entkoppelt ist sowie die Applikation später über die Möglichkeit verfügt, verteilt skaliert zu werden,
nehmen dafür aber in Kauf, dass die Generierung einer Ausführbaren Datei unter Windows (.exe) sowie deren Installation komplexer wird.

4.2.3 Webframework-Entscheidung Y-Template

Im Kontext des Webframworks für die Software ImageFinder der Evangelisch-reformierte Kirche Horgen,
um den nicht-funktionale Anforderung „NFR13 - Webapplikation” und „NFR8 - Win 10” sowie NFR5, NFR6 und NFR7 aus der Kategorie Performance entgegen zu treten,
haben sich die Autoren für den Einsatz von Sanic entschieden und andere Webframeworks wie Tornado, FastAPI, CherryPy und Responder verworfen,
damit die Webapplikation auf Windows 10 installiert werden kann und auf eine einfache Art und Weise mehrere Prozesse für die Skalierung gestartet werden können,
nehmen dafür aber in Kauf, dass das Alter des Frameworks noch nicht über drei Jahre hinausgekommen ist und deshalb Enterprise-Funktionalitäten wie zum Beispiel das Verschlüsseln von Cookies nicht vorhanden sind [15].

4.2.4 Datenbank-Entscheidung Y-Template

Im Kontext der Datenbank für die Software zur Bildersuche der Reformierten Kirche Horgen,
um der nicht-funktionalen Anforderung „NFR12 - Persistenz” entgegen zu treten,
haben sich die Autoren für den Einsatz von SQLite entschieden und andere RDBMS wie Postgres und MySQL verworfen,
damit eine einfache Inbetriebnahme ohne die Installation eines Datenbanksystems gewährleistet werden kann und eine einfache Datenbank für Testzwecke out-of-the-box bereitsteht,
nehmen dafür aber in Kauf, dass zur gleichen Zeit immer nur eine Schreiboperation ausgeführt werden kann.

4.2.5 Client-Server-Kommunikation

Der Server soll das REST Maturity Level 2 wie es Fowler beschreibt, erreichen [16]. Die dazu notwendigen Schritte zeigt Tabelle 8.

Tabelle 8: REST Maturity

Level	Beschreibung
Level 0: The Swamp of POX	Client und Server sollen über das HTTP-Protokoll kommunizieren. Als Austauschformat zwischen Client und Server soll JSON verwendet werden.
Level 1: Resources	Durch eine saubere Strukturierung der URI's sollen verschiedene Endpoints verwendet werden.
Level 2: HTTP Verbs	Die Schnittstelle soll die HTTP-Verben GET/POST/PUT unterstützen.

Anmerkung. Eigene Darstellung.

4.2.6 Layers

Das System soll in Bezug auf die Architektur in mehrere Layers gegliedert werden. Dabei sollen sich die Schichten nach der Presentation - Domain - Data Empfehlung von Fowler [17], wie sie in Abbildung 4 ersichtlich ist, richten.

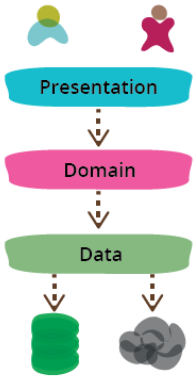


Abbildung 4: Presentation-Domain-Data Layering [18]

4.3 Clustering

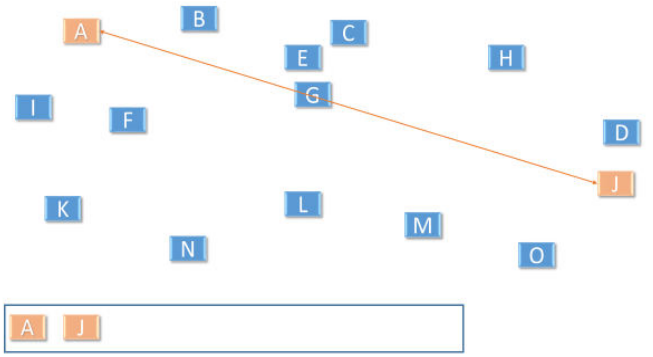
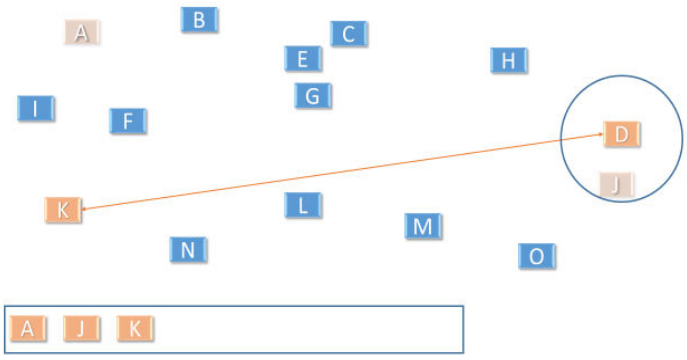
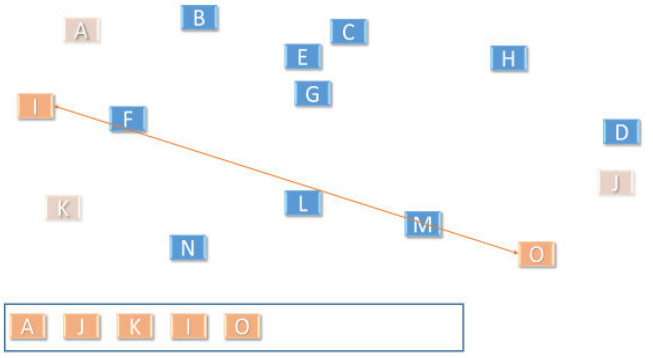
Für die Applikation wurden verschiedene Clustering-Algorithmen evaluiert, um der nutzenden Personen der Applikation eine schnellere Übersicht über die Bilder zu geben. Einerseits sollen in einer Übersicht möglichst unterschiedliche Bilder aus der Sammlung angezeigt werden, andererseits sehr ähnliche Bilder zu einem Cluster gruppiert werden. Die Clustering-Verfahren werden im Anschluss an das Hochladen von weiteren Bildern immer wieder neu ausgeführt. Dies garantiert einen stets aktuellen Stand der berechneten Cluster.

4.3.1 Clustern unterschiedlicher Bilder

Wie schon in der Studienarbeit wurde das Top-Down-Verfahren [19] in Kombination mit dem durch die Receiver Operator Characteristic (ROC) Curve berechneten Schwellwert verwendet, um möglichst unterschiedliche Bilder aus der Datenbank zu erhalten. Neu sollten jedoch 80 anstatt 40 Bilder angezeigt werden. Mit dem in der Studienarbeit verwendeten Algorithmus dauert dies zu lange, da die Operation von Numpy, matrix.min(), verhältnismässig viel Zeit in Anspruch nimmt. Aus diesem Grund wurden zwei neue Ansätze getestet, um das Clustering zu beschleunigen:

Variante 1: Umsetzung aus der Studienarbeit
In der Studienarbeit wurde, wie in Tabelle 9 ersichtlich, mit der matrix.min() Methode iterativ das nächste Bildpaar gesucht, welches am wenigsten Ähnlichkeitsmerkmale aufweist.

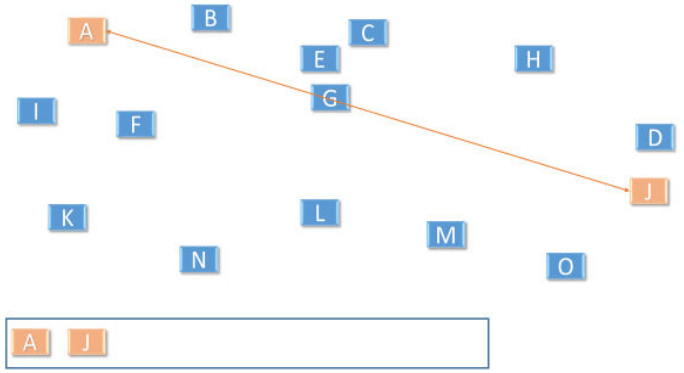
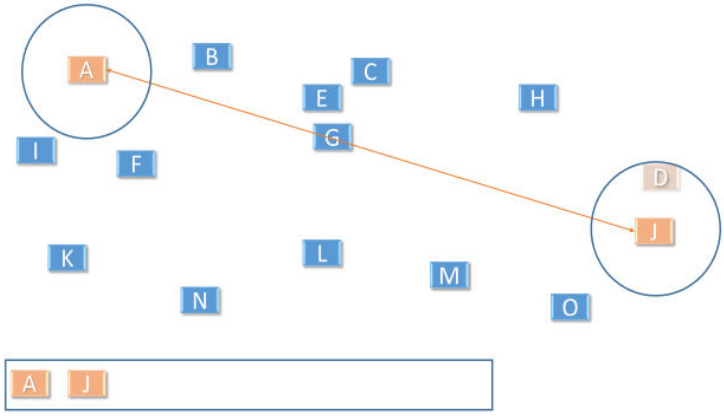
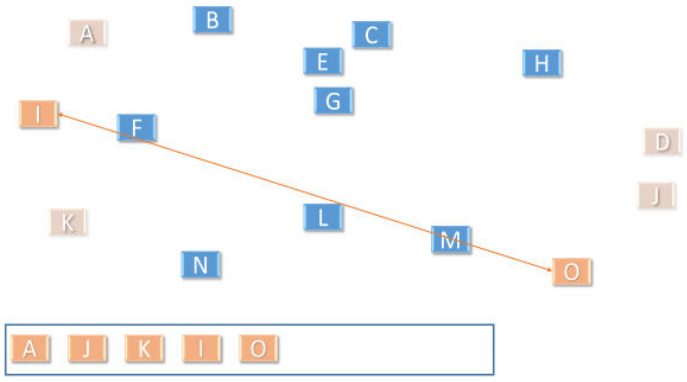
Tabelle 9: Clustering Übersichtsbilder Prototyp

	Mit matrix.min() die unterschiedlichsten Bilder finden und in die Liste der Übersichtsbilder eintragen.
	Mit matrix.min() die nächsten unterschiedlichsten Bilder finden und mit dem vorgängig berechneten ROC Wert prüfen, ob die Bilder in der Liste unterschiedlich sind.
	Danach wird das Prozedere mit der nächsten Distanz wiederholt.

Anmerkung. Eigene Darstellung.

Variante 2: Umsetzung mit Wipen der nächsten Bilder
Wie Tabelle 10 zeigt, wurde ein anderer Ansatz getestet, um die Anzahl der Verwendungen von matrix.min() zu minimieren.

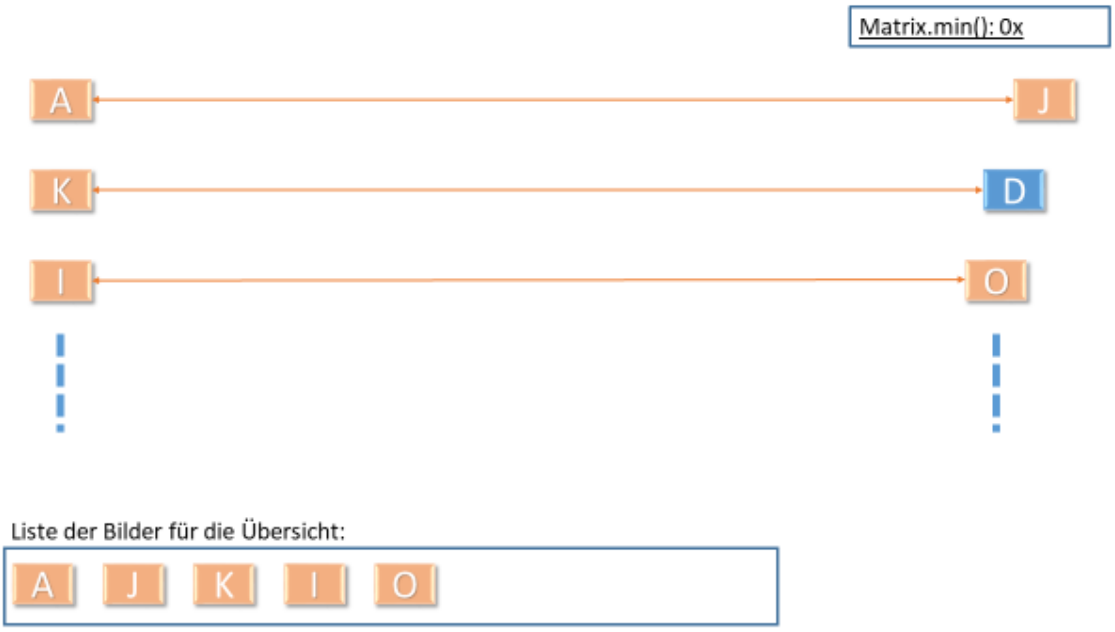
Tabelle 10: Clustering Übersichtsbilder Wipen

	Bei der Wipe Variante wird bei dem ersten Bildpaar gleich gestartet wie bei der Umsetzung in der Studienarbeit.
	Für die ersten beiden gefundenen Bilder werden ähnliche Bilder im Umkreis aus der Gleichheits-Matrix entfernt und folglich nicht mehr beachtet.
	Durch das Wipen der ähnlichsten Bilder wird matrix.min() genau so viel Mal aufgerufen, wie Bilder zur Übersicht hinzugefügt werden.

Anmerkung. Eigene Darstellung.

Variante 3: Umsetzung mit sortieren der Gleichheits-Matrix
Als dritte Variante wurde mit sortierter Gleichheitsmatrix versucht, komplett auf matrix.min() zu verzichten (siehe dazu Tabelle 11).

Tabelle 11: Clustering Übersichtsbilder mit sortierer Matrix



Die Gleichheits-Matrix wird nach den Distanzen aufsteigend sortiert. Dieser Vorgang benötigt für 11‘734 Feature-Vektoren von zehn Durchläufen im Durchschnitt 20.64 Sekunden. Die Methode matrix.min() wird so nicht mehr benötigt.

Anmerkung. Eigene Darstellung.

Wahl des Algorithmus
Zu den verschiedenen Verfahren wurden Tests durchgeführt. Diesbezüglich wird auf Abschnitt 6.5.4 verwiesen. Aufgrund der Tests soll schlussendlich Variante zwei umgesetzt werden.

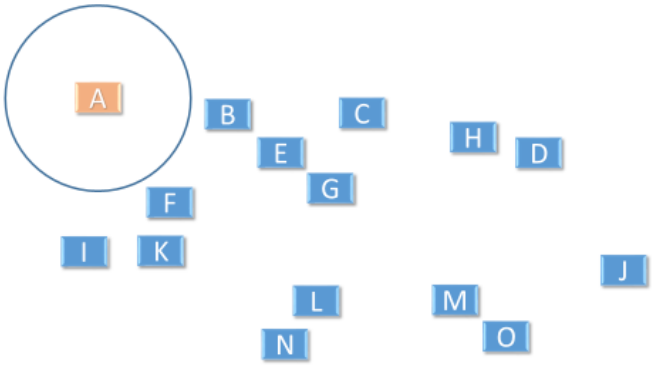
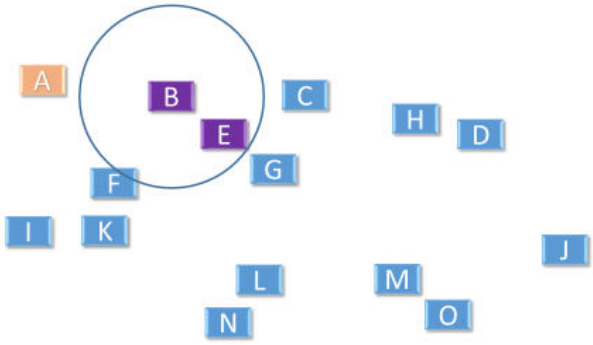
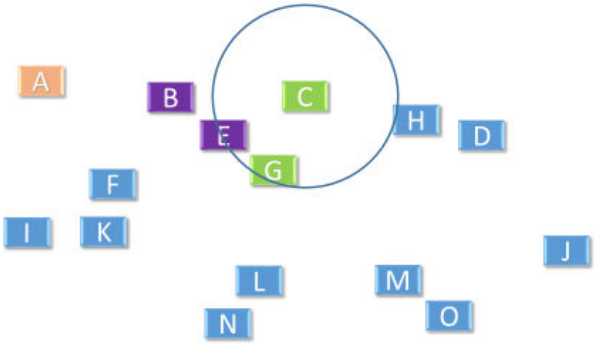
4.3.2 Clustern ähnlicher Bilder

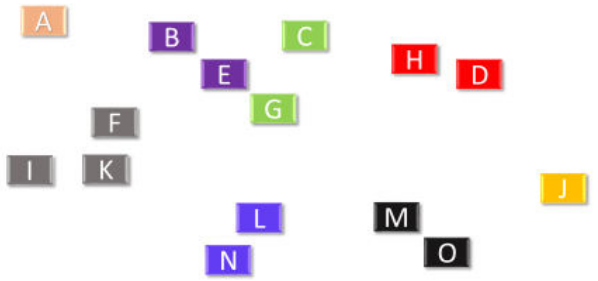
In der Studienarbeit zeigte sich, dass bei einer Bildersuche nach beispielsweise „Kirchenorgel" oder „Kirchenturm" die Ergebnisse sehr ähnlich ausfielen. Gewisse Bildmerkmale sind sehr häufig in der Bildersammlung vertreten. Durch Gruppieren, bzw. Clustern der ähnlichen Bilder, kann eine grössere Variabilität der angezeigten Bilder erreicht werden.
Beim Clustern der Bilder war entscheidend, ob sich die verwendete Clustering-Methode für die Bildersammlung der Evangelisch-reformierten Kirche Horgen eignet. Es sollen nicht zu grosse Cluster entstehen, da sonst zu unterschiedliche Bilder gruppiert werden. Bei zu kleinen Clustern wäre genau das Gegenteil der Fall. Folglich ist die Berechnung des Schwellwertes von entscheidender Wichtigkeit.

Schwellwert der Suche nach ähnlichen Bildern
Als Entscheidungsschwelle, ab welcher Ähnlichkeit Bilder gruppiert werden sollen, wurde für jedes Bild die zwischen null und eins normalisierte Kosinus-Distanz zu seinem nächsten Nachbarn berechnet. Die erhaltenen Distanzen wurden addiert und durch die Anzahl Bilder geteilt. Für die insgesamt 11‘734 Bilder der Evangelisch-reformierten Kirche Horgen wurde so eine Kosinus-Distanz von **0.8376** als Schwellwert berechnet.

Clusterbildung
Mit dem erhaltenen Schwellwert wurde iterativ für jedes Bild die innerhalb des Schwellwerts liegenden Nachbarn eruiert. Der Vorgang wurde in Tabelle 12 veranschaulicht. Der Kreis stellt dabei den mittels dem Schwellwert berechneten Distanzradius dar. Die buchstabierten Vierecke illustrieren die Bilder anhand ihrer Ähnlichkeit.

Tabelle 12: Clustering ähnlicher Bilder

	In der ersten Iteration werden zu Bild A die nächsten Nachbarn ermittelt. Bild A hat keine nächsten Nachbarn und bildet somit einen Cluster mit nur einem Bild.
	In der zweiten Iteration werden zu Bild B die nächsten Nachbarn ermittelt. Bild E liegt innerhalb des Schwellwerts und wird zum Cluster hinzugefügt. Dieser Cluster beinhaltet folglich zwei Bilder.
	Der Vorgang wird mit dem nächsten Bild wiederholt. Bereits geclusterte Bilder werden nicht mehr berücksichtigt. Jedes Bild ist somit nur in einem Cluster.

	In diesem Beispiel ergaben sich insgesamt acht Cluster, welche in der Datenbank persistiert werden.

Anmerkung. Eigene Darstellung.

Ergebnisse der Clusterbildung

Um zu analysieren, ob sich die Berechnung des Schwellwertes eignet wurden die Clustergrössen analysiert. Dies ist in Tabelle 13 ersichtlich.

Tabelle 13: Auswertung Clustergrössen

Clustergrösse [G]	Anzahl Cluster mit Grösse [G]
1	6683
2	1066
3	294
4	116
5	44
6	29
7	17
8	7
9	6
10	9
11	8
12	1
13	2
14	2
15	2
16	2
17	1
19	1
24	2
32	1
34	1
43	1
51	1

Anmerkung. Eigene Darstellung.

Für Clustergrössen ab 16 Bildern wurde visuell geprüft, welche Bildmerkmale zu erkennen waren und wie viele der Bilder das Merkmal enthalten. Wie aus Tabelle 14 zu entnehmen ist, enthalten die Cluster keine Ausreisser. Sämtliche Bilder der Cluster enthalten das aufgeführte Merkmal.

Tabelle 14: Bildmerkmale der grösseren Cluster

Clustergrösse	Treffer	Bildmerkmale
16	16	Kirchenkonzert
17	17	Kirchenchor
19	19	Kirchenfest
24	24	Gruppenfoto im Freien,
24	24	Gruppenfoto mit Taufbrunnen
32	32	Kirchenorgeln
34	34	Kirchentürme
43	43	Kirchenversammlung
51	51	Personengruppe im Freien

Anmerkung. Eigene Darstellung.

Anwendungszweck der gebildeten Cluster
Mit den ermittelten Clustern ist es nun möglich nach einer Bildersuche die Resultate zu gruppieren. Hierbei wurden zwei verschiedene Verfahren getestet.

Verfahren 1: Bestes Bild aus Cluster
Wird eine Suche gestartet, werden aus der Baumstruktur die ähnlichsten Bilder zurückgegeben. Grundsätzlich werden diese Bilder der Reihe nach, sortiert nach absteigender Ähnlichkeit, angezeigt. Jedoch erscheint von jedem Cluster nur der beste Treffer. Die restlichen Bilder werden in der Übersicht nicht beachtet werden. Dies ist in Abbildung 5 visualisiert. Wiederum steht jedes Rechteck für ein Bild. Die Zahl im Rechteck beschreibt die Kosinus-Distanz zum Referenzbild, bzw. den Referenzbildern.

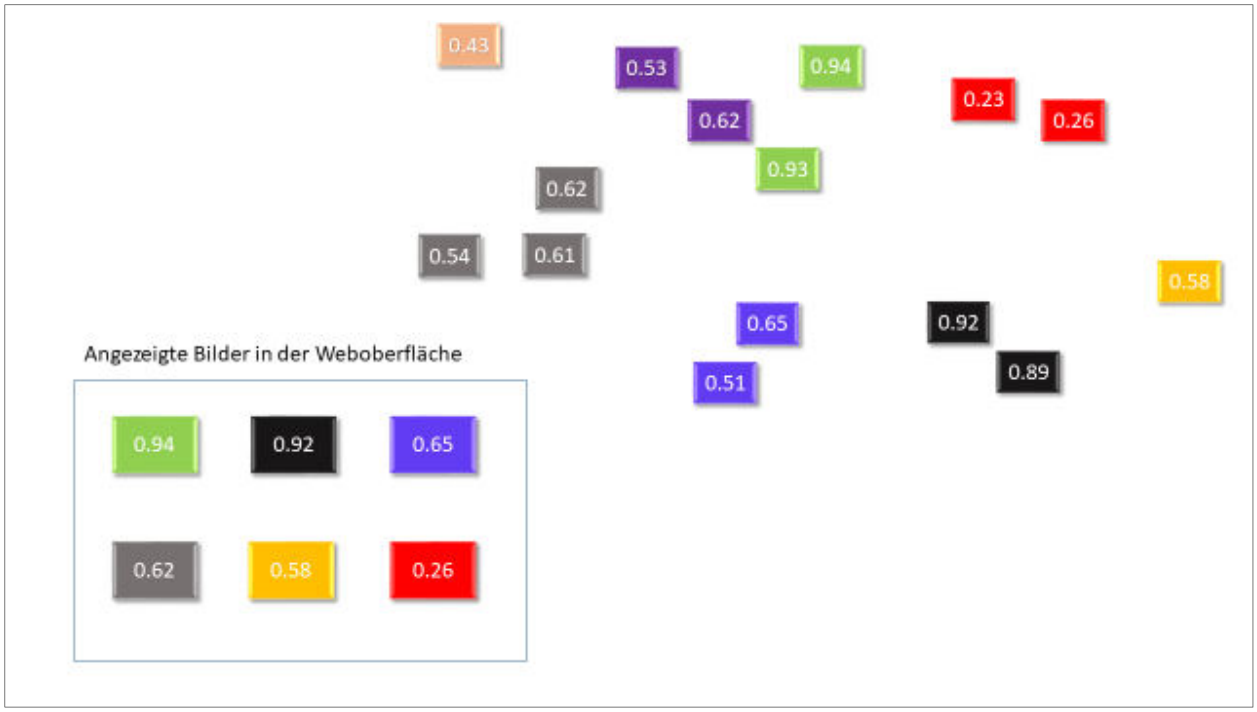


Abbildung 5: Clusteranzeige, bestes Bild aus Cluster

Anmerkung. Eigene Darstellung.

Verfahren 2: Beste Treffer aufsplitten
Beim zweiten Verfahren werden in der ersten Zeile die Cluster nicht berücksichtigt. Das heisst, es ist möglich, dass mehrere Bilder aus einem Cluster in der Übersicht vorkommen. Ansonsten gleicht diese Variante dem Verfahren 1. In Abbildung 6 ist zu sehen, dass zwei Bilder aus dem grünen Cluster ausgewählt wurden.

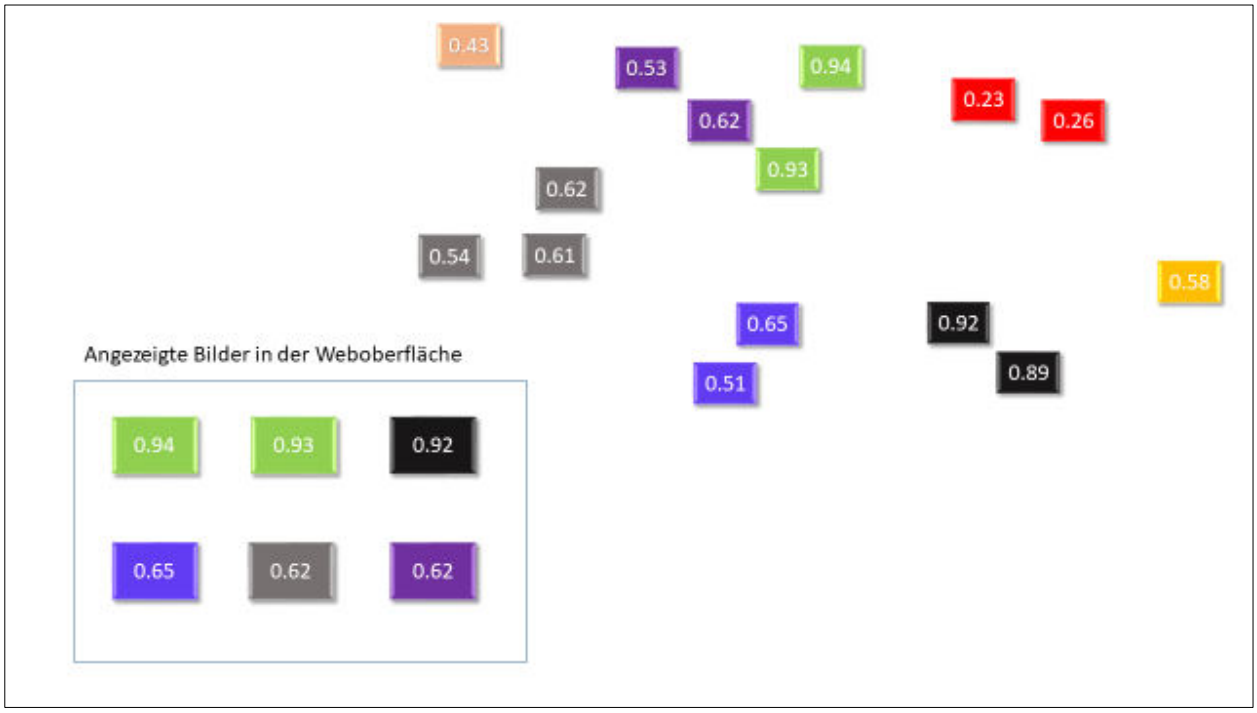


Abbildung 6: Clusteranzeige, beste Treffer aufsplitten

Anmerkung. Eigene Darstellung.

Erkenntnis des Clusterings in der Anwendung
Das Clustering mit dem verwendeten Schwellwert liefert sehr gute Ergebnisse. Die Bilder in einem Cluster sind sich sehr ähnlich. Durch den hohen Schwellwert werden jedoch Bilder, welche visuell einem Cluster zugeordnet werden könnten, nicht berücksichtigt. Da zu grosse Cluster jedoch vermieden werden sollen, eignet sich dieses Verfahren für unsere Applikation.

Das Aufsplitten der Cluster in der ersten Zeile erwies sich als Optimierung der Ergebnisse. Bei den in Anhang J beschriebenen Usability-Tests waren die Anwender verwirrt, wenn zu einem gesuchten Bild nur ein oder zwei passende Bilder erscheinen. Durch das zweite Verfahren werden in jedem Fall die vier besten Treffer angezeigt. Folglich wurde dieses in der Software umgesetzt.

4.4 Annoy

Beim Ausführen einer Suche nach ähnlichen Bildern müssen aus der gesamten Bildersammlung die ähnlichsten Nachbarn gefunden werden. Im Prototyp aus der Studienarbeit wurde dies mit den Feature Vektoren jedes Mal neu berechnet. Mit der C++ Bibliothek „Annoy“ [20] ist es möglich, sämtliche Distanzberechnungen in einer Baumstruktur zu speichern. Distanzberechnungen sind somit nach dem Hochladen der Bilder nicht mehr nötig, was sich signifikant auf die Performanz auswirkt. Bezüglich der Testergebnisse wird auf Abschnitt 3 verwiesen.

Datenspeicherung
Die mit der Annoy-Bibliothek generierte Baumstruktur wird in einem lokalen File gespeichert. Das Mapping der Daten erfolgt mit dem POSIX-compliant Unix system call „mmap“ [21]. Dies erlaubt einen performanten Datenzugriff durch mehrere Prozesse gleichzeitig.

Vergleich zu anderen NNS-Bibliotheken
Die Annoy-Bibliothek kann mit den gängigen Nearest Neighbor Berechnungs-Bibliotheken mithalten wie aus Abbildung 7 entnommen werden kann. Zudem bietet die Bibliothek mit den bestehenden Methoden „get_nns_by_item“ und „get_nns_by_vector“ auf einfache Art und Weise die für die Software benötigte Funktionalität.

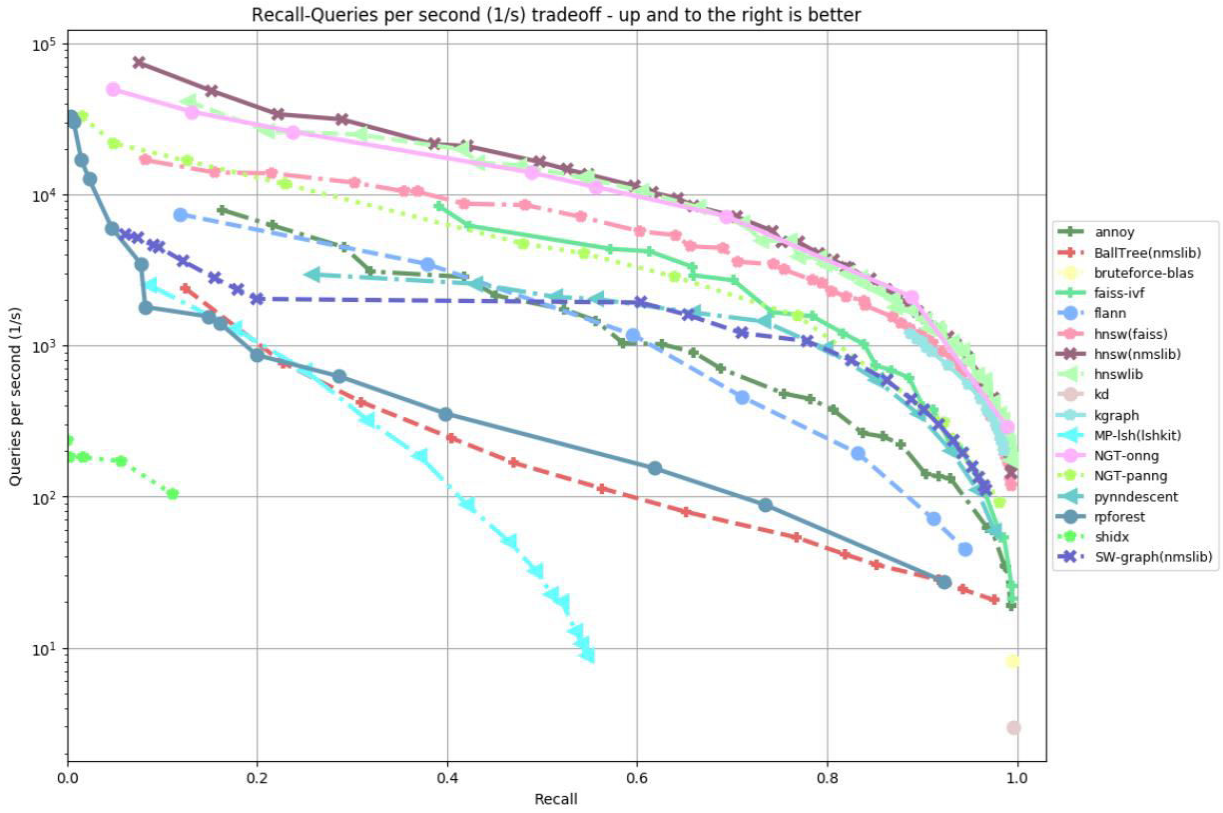


Abbildung 7: Vergleich der Nearest Neighbor Berechnungs-Bibliotheken [22]

Restriktionen mit der Annoy Bibliothek
Die Baumstruktur wird mit „random projections“ gebildet. [23] Die Granularität der Baumstruktur wird mit einer festen Konstante k vorgegeben. Wird der Wert für k erhöht, steigert sich die Genauigkeit der Ergebnisse. Die Berechnungszeit der Baumstruktur nimmt jedoch zu. Da die Baumstruktur beim Hochladen von neuen Bildern jedes Mal neu generiert werden muss, sollte ein Verhältnis zwischen Berechnungszeit und Genauigkeit eruiert werden.

Aufgrund der in Abschnitt 5.5 unter Atomic Datenbank Updates beschriebenen Einschränkungen, wurde die Anzahl der Nearest Neighbors auf 900 beschränkt. Die von der Datenbank geladenen Bilder werden nach den Nearest Neighbors gefiltert. Falls dabei das Limit der gebundenen Variabeln in einem SQL-Statement erreicht wird, kommt es zu einem Fehler. Mit der Beschränkung der Anzahl Nearest Neighbors kann dies vermieden werden.

Suche mit mehreren Referenzbildern

Die Suche mit mehreren Referenzbildern wurde in der Studienarbeit bemängelt, da der Mehrwert für den Anwender nicht spürbar ist. Deshalb wurde ein neuer Ansatz getestet. Dabei sollte nebst besseren Suchergebnissen auch die Performanz beibehalten werden. Von der Begleitperson wurde der Algorithmus „Best-lowest-rank-first“ vorgeschlagen.

In der Studienarbeit wurde über die ausgewählten Referenzbilder das arithmetische Mittel aus den Feature-Vektoren berechnet. Statt sämtliche Distanzberechnungen der einzelnen Referenzbilder über die gesamte Bilderdatenbank durchzuführen, konnte so die Performanz gesteigert werden. Von den 20 gemachten Tests, bei welchen ähnliche Bilder zu zwei Referenzbildern gesucht wurden, fielen jedoch nur fünf positiv aus.

Der „Best-lowest-rank-first“ Algorithmus sucht für jedes Referenzbild aus der durch die Annoy-Bibliothek gebildeten Baumstruktur die ähnlichsten Bilder. Tabelle 15 zeigt eine beispielhafte Darstellung der Ergebnisse.

Tabelle 15: Distanzen der Bilder aus der Bildersammlung zu Referenzbildern

		Bilder aus der Bildersammlung											
Referenz-bilder		D	E	F	G	H	I	J	K	L	M	N	O
	A	0.78	0.84	0.24	0.63	0.45	0.75	0.65	0.85	0.45	0.55	0.15	0.47
	B	0.74	0.52	0.86	0.65	0.64	0.56	0.75	0.85	0.24	0.54	0.66	0.45
	C	0.73	0.51	0.85	0.87	0.74	0.45	0.86	0.56	0.36	0.96	0.75	0.55

Anmerkung. Eigene Darstellung.

Wie in Tabelle 16 illustriert, wird für jeden Vergleich von Referenzbild und einem gefundenen ähnlichen Bild, einen Rang (aufsteigend) vergeben.

Tabelle 16: Zugeteilte Ränge der gefundenen Nachbarn

		Bilder aus der Bildersammlung											
Referenz-bilder		D	E	F	G	H	I	J	K	L	M	N	O
	A	3	2	11	6	10	4	5	1	8	7	12	9
	B	4	10	1	6	7	8	3	2	12	9	5	11
	C	7	9	4	2	6	11	3	8	12	1	5	10

Anmerkung. Eigene Darstellung.

Die Ergebnisse werden anschliessend untereinander verglichen. Dabei wird stets der schlechteste Rang für ein Bild berücksichtigt. Somit erhält zum Beispiel, wie in Tabelle 17 aufgeführt, Bild D den Rang 7.

Tabelle 17: Verteilung der Ränge für die ähnlichen Bilder

Referenzbilder	Bilder aus der Bildersammlung											
	D	E	F	G	H	I	J	K	L	M	N	O
A	3	2	11	6	10	4	5	1	8	7	12	9
B	4	10	1	6	7	8	3	2	12	9	5	11
C	7	9	4	2	6	11	3	8	12	1	5	10

Anmerkung. Eigene Darstellung.

Abbildung 8 zeigt die schlussendliche Anordnung der Bilder. Links oben ist Bild J zusehen, welches mit Rang fünf den tiefsten Rang hat. Danach wird die Anzeige aufsteigend nach Rang aufgefüllt.

Zu beachten ist, dass bei eingeschaltetem Clustering ab der zweiten Zeile Bilder in die Cluster der bereits angezeigten Bilder rutschen können. Ist beispielsweise Bild M im Cluster von Bild J, wird Bild M nicht angezeigt. Dies wird auch für alle nachfolgenden Bilder wiederholt. Befindet sich beispielsweise Bild O im Cluster von Bild H, wird auch Bild O nicht angezeigt.

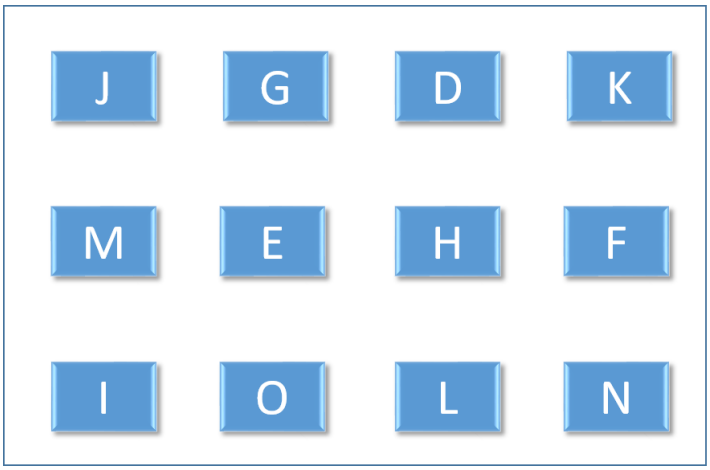


Abbildung 8: Anordnung der gefundenen Bilder bei mehreren Referenzbildern

Anmerkung. Eigene Darstellung.

4.5 Ein erster grafischer Protoyp

Um einen ersten Eindruck für eine mögliche Benutzeroberfläche zu erhalten, wurden Wireframes erstellt. So war es möglich, schnell ein Feedback einzuholen und den Funktionsumfang zu bestätigen.

Wireframes
Die erstellten Wireframes sind in Anhang M ersichtlich. Die Wireframes wurden mit der Begleitperson sowie den Mitarbeitenden der Evangelisch-reformierten Kirche Horgen besprochen. Dabei wurden folgende Verbesserungsmöglichkeiten eingebracht:

- Beim Hochladen der Bilder soll eine Fortschrittsanzeige angezeigt werden.
- Die Bilddarstellung soll ähnlich dargestellt werden wie die Google-Bildersuche
- Falls noch keine Bilder hochgeladen wurden, soll zuerst ein Fenster für den Bildupload angezeigt werden. Sind bereits Bilder in der Datenbank, dann soll direkt auf die Bilderansicht gewechselt werden.
- Die Komponenten der Weboberfläche sollen einheitlich angeordnet sein.
- Wird beim Hochladen der Bilder weder ein Tag noch der Fotograf angegeben, muss der Upload zusätzlich in einem Dialogfeld bestätigt werden.
- Ist die Google Bildersuche erfolglos, soll dies mit einer Meldung kenntlich gemacht werden.
- Können zu den Filterkriterien keine Bilder gefunden werden, soll auf ein separates Fenster weitergeleitet werden.
- Das Datum soll per Tastatur eingegeben werden können.

4.6 Analyse der Frontend Webframeworks

Diverse Frontend Webframeworks wurden verglichen. Bezüglich der detaillierten Auswertung wird auf Anhang H verwiesen.

Wahl des Frontend Frameworks
Aufgrund der getätigten Abklärungen und der Vorkenntnisse der Autoren wurde festgelegt, das Vue.js Framework zu verwenden. Der Entscheid fiel vor allem aufgrund der einfacheren Lernkurve aufgrund der Verwendung von HTML-Templates.

4.7 Erreichung der nicht-funktionalen Anforderungen

In diesem Abschnitt werden die in Tabelle 2 festgehaltenen nicht funktionalen Anforderungen adressiert. Dabei soll in Tabelle 18 festgehalten werden, wie die nicht-funktionalen Anforderungen erfüllt werden können.

Tabelle 18: Adressierung der nicht-funktionalen Anforderungen

#	Titel	Wird adressiert durch…
NFR1	Global Exception-Handler	das Einsetzen eines Loggers sowie eines globalen Exception-Handlers.
NFR2	OWASP TOP 10	eine Überprüfung der OWASP TOP 10 Bedrohungen.
NFR3	UI Design	die Verwendung einer Material Design Library.
NFR4	Benutzerführung	eine Durchführung von Usability-Tests.
NFR5	Response-Time „/"	die Evaluation eines geeigneten Webframeworks.
NFR6	Nearest Neighbor Search	die Persistierung der Ähnlichkeitsmatrix in Form einer Baumstruktur.
NFR7	Bildimport	die Evaluation eines geeigneten Webframeworks analog NFR5.
NFR8	Win 10	Verwendung keiner Abhängigkeit, die nicht Windows 10 kompatibel ist.
NFR9	Sprache	Usability-Tests.
NFR10	Log-Level	Verwendung eines Log-Frameworks, das mehrere Stufen kennt.
NFR11	Datenschutz	die Speicherung der Bilddatenbank auf eigener Hardware ohne Weitergabe an Dritte.
NFR12	Persistenz	Selektion einer relationalen Datenbank
NFR13	Webapplikation	das Einsetzen von Webtechnologien und Wahl der Architektur.
NFR14	Temporäre Daten	eine saubere Architektur mit Handling der temporären Daten
NFR15	Inception-v3	die Verwendung des Inception-v3 Netzwerks.
NFR16	Baumstruktur	die Entwicklung einer Baumstruktur oder Wahl einer Library, die dies unterstützt.
NFR17	Best lowest search rank first	die Implementation des genannten Algorithmus.
NFR18	Google-Schnittstelle	die Verwendung der offiziellen Google Custom Search API.
NFR19	Code Formatting	die Definition eines Style-Guides.
NFR20	Testcoverage (Frontend)	die Messung der Frontend-Testabdeckung.
NFR21	Testcoverage (Backend)	die Messung der Backend-Testabdeckung.
NFR22	Agile Vorgehensweise	die Wahl einer agilen Projektmanagement-Methode.

Anmerkung. Eigene Darstellung.

5 Umsetzung

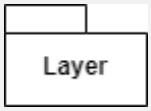
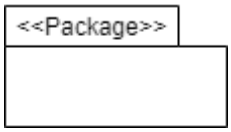
In diesem Kapitel wird auf die Umsetzung ausgewählter Komponenten und Algorithmen eingegangen. Zuerst wird die Architektur der Software und der Aufbau der Datenbank beschrieben. Anschliessend wird ein Beispiel gezeigt, wie die Kommunikation zwischen Client und Server funktioniert, gefolgt von Softwaredesign-Entscheidungen.

5.1 Architektur

Anhand einer einfachen Systemübersicht soll zuerst gezeigt werden, aus welchen Komponenten die Architektur besteht. Danach wird der Aufbau der logischen Schichten-Architektur ins Auge gefasst.

In Tabelle 19 ist die Notation für die Nachfolgenden Architekturdiagramme festgehalten.

Tabelle 19: Architektur - Übersicht über die Notation

Symbol	Beschreibung
	Ein Layer ist die höchste Einheit.
	Ein Package ist immer in einem Layer angesiedelt.
	In Python stellt ein Modul die kleinste Einheit dar, die alleine stehen kann. In einigen Fäll wird mit dem gleichen Symbol auch eine Klasse bezeichnet. Das Frontend kennt keine Module, sondern lediglich Klassen oder Components.

Anmerkung. Eigene Darstellung.

5.1.1 Systemübersicht

Die Software besteht grob aus drei Komponenten mit ihren jeweiligen Technologien. Dabei wird, wie in Abbildung 9 zu sehen ist, zwischen Front- und Backend unterschieden.

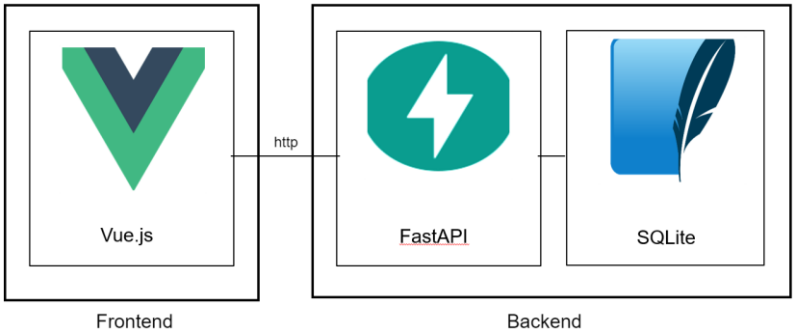


Abbildung 9: Systemübersicht

Anmerkung. Eigene Darstellung.

Die detaillierte Architektur wird in den folgenden Abschnitten näher beschrieben.

5.1.2 Logische Architektur Frontend

In Abbildung 10 ist die Schichtenarchitektur des Frontends abgebildet.

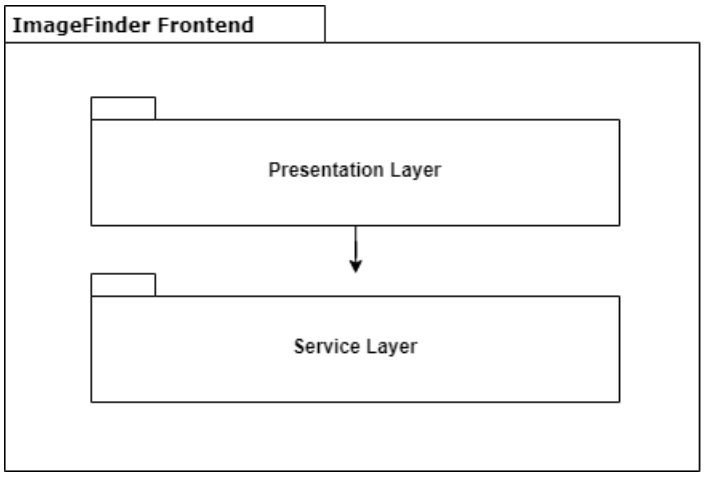


Abbildung 10: Frontendarchitektur

Anmerkung. Eigene Darstellung.

Der Presentation Layer greift auf Funktionalitäten des Service Layers zu. Nachfolgend werden die beiden Schichten genauer beschrieben.

Presentation Layer

Die View enthält die Presentation-Components. Eine Component ist im Sinne einer Single-File-Component nach Vue.js [24] zu verstehen. Sie dient zur Darstellung der Benutzeroberfläche und enthält die drei Bereiche Template, Script und Style. Im Template wird die Hypertext Markup Language (HTML) verwendet. Die Sektion Script enthält die Logik, wann welcher Abschnitt und welche Daten vom Browser gerendert werden sollen. Ebenfalls sind kleine Helper-Funktionen enthalten, die zum Beispiel ein Datum-Objekt zu einem String parsen. Im Style-Abschnitt wird mittels Cascading Style Sheets (CSS) Language beschrieben, wie die HTML-Elemente angeordnet werden sollen. In Abbildung 11 sind insgesamt 11 Components abgebildet.

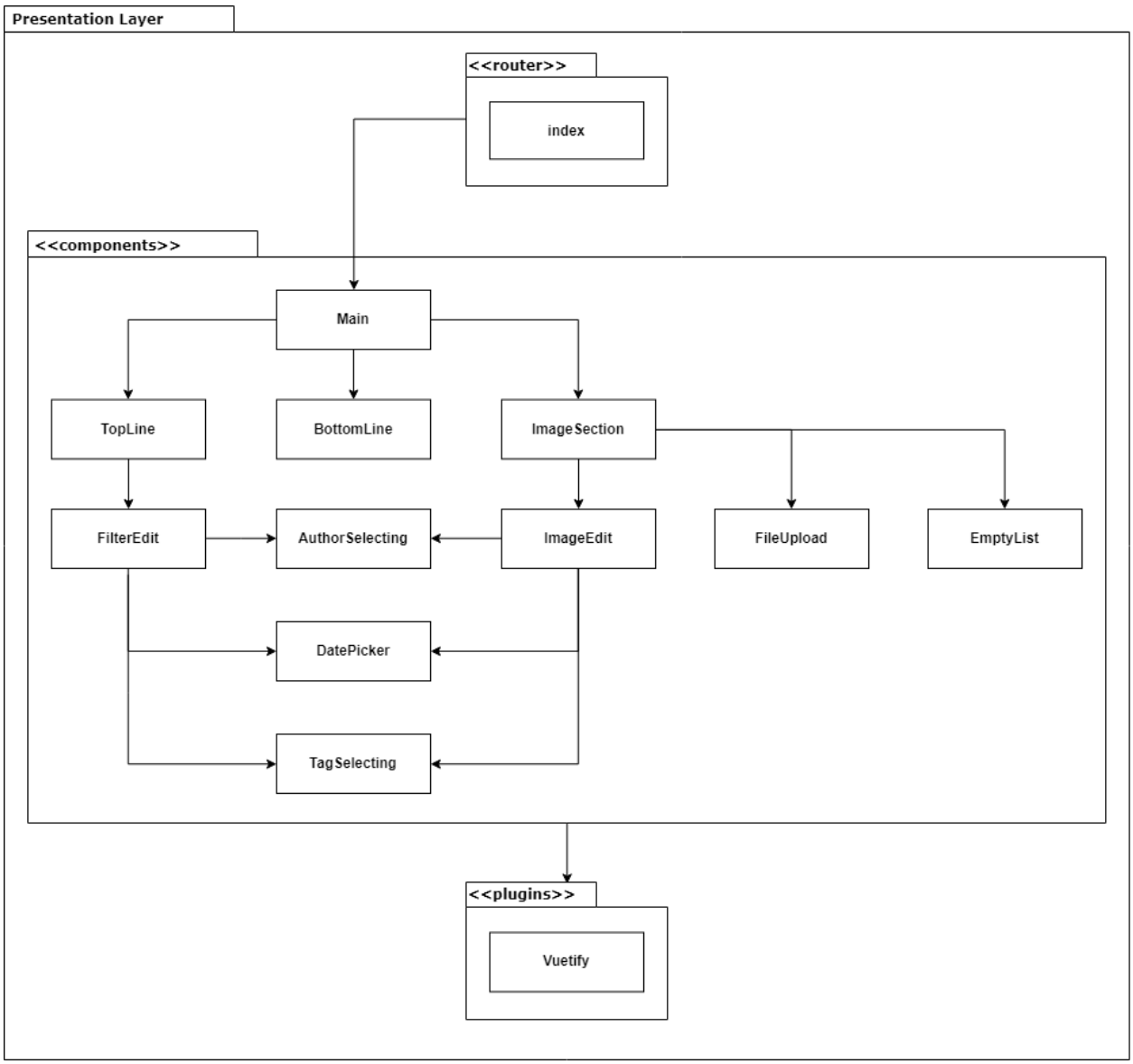


Abbildung 11: Frontendarchitektur - Presentation Layer

Anmerkung. Eigene Darstellung.

Die wichtigsten Components werden in Tabelle 20 beschrieben.

Tabelle 20: Components Frontend

Component (frontend/src/components)	Beschreibung
Main	Leitet die zur Laufzeit erhaltenden Events an die jeweiligen Components weiter.
ImageSection	Hauptoberfläche der Bilderanzeige, dient als Container für weitere Components.
FileUpload	Fenster für das Hochladen der Bilder, sogenannter „File-Chooser".
TopLine	Immer sichtbar, Navigation zu Filter, Fileupload, Bilderansicht, Darstellung der selektierten Bilder.
ImageEdit	Bearbeitung der Metadaten und Bildansicht.

Anmerkung. Eigene Darstellung.

Vom Plugin Vuetify werden vorgefertigte Komponenten verwendet. Auf die Wichtigsten wird in Tabelle 21 näher eingegangen [25].

Tabelle 21: Vuetify Komponenten

Vuetify Components	Beschreibung
v-grid	Die Grid List wurde als Grundbaustein für die Bilderansicht verwendet. Durch Angabe der URI werden die Bilder automatisch abgefüllt und bei Änderungen aktualisiert.
v-date-picker	v-date-picker erleichtert die Auswahl eines Datums und kann nach Bedarf angepasst werden.
v-dialog	Mit der Komponente v-dialog wird sichergestellt, dass die nutzende Person nur im Dialogfenster Aktionen vornehmen kann.
v-container, v-layout, v-flex	Mit v-container, v-layout und v-flex wird ein Vuetify Responsive Design. Änderungen der Bildschirmgrösse werden so erkannt und das Layout entsprechend angepasst. Für die Software wurden jedoch keine Bildschirme in Smartphone Grösse berücksichtigt.

Anmerkung. Eigene Darstellung.

Im Package Router ist ein einfacher Vue.js-Router [26] enthalten, der den Einstieg in die Single Page Application (SPA) sicherstellt.

Service Layer

Der Service Layer dient um mit der HTTP-Schnittstelle des Backends zu kommunizieren und wurde mittels dem OpenAPI Generator [27] automatisch generiert. Abbildung 12 zeigt die zwei Packages „model" und „api" sowie deren wichtigsten Klassen.

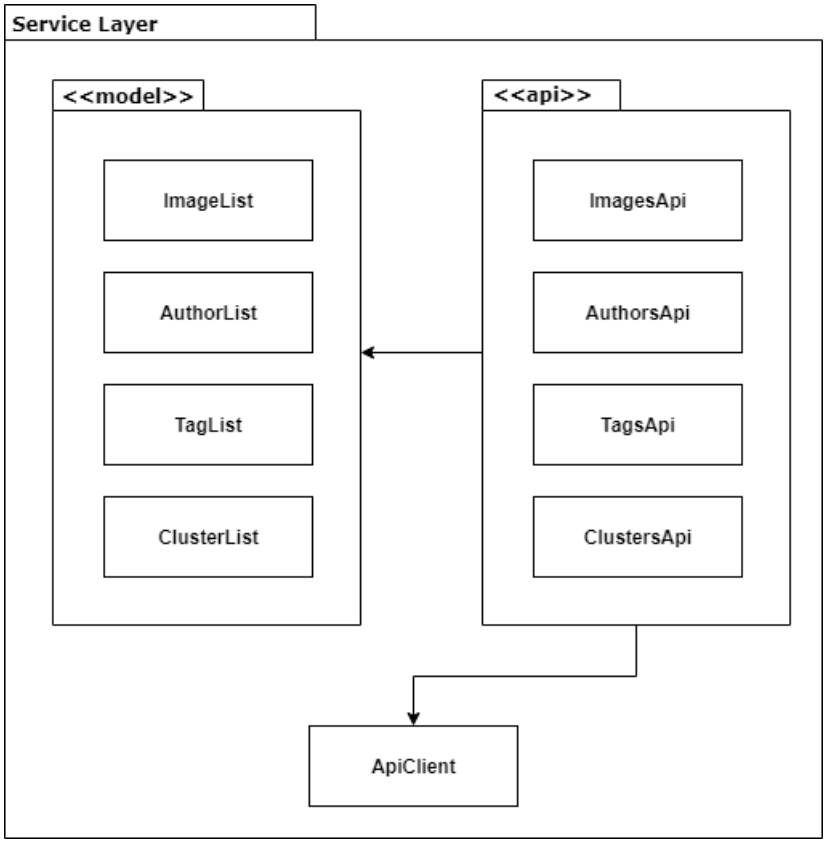


Abbildung 12: Frontendarchitektur - Service Layer

Anmerkung. Eigene Darstellung.

Die Models dienen lediglich der Datenhaltung und werden von den Api-Klassen zum Versenden und Empfangen von Daten über HTTP verwendet. Dazu verwenden die API-Klassen einen Api-Client, der die Daten serialisiert.

Der Service Layer befindet sich physisch in frontend/src/api-services und enthält den ApiClient im Stammverzeichnis, die Models in /models/* sowie Apis in /api/*.

5.1.3 Logische Architektur Backend

Entsprechend dem Anwendungsfall wurden wie in Abbildung 13 zu sehen ist, die Terminologien „Web Layer" und „Business Logic Layer" statt „Presentation Layer", bzw. „Domain" von Fowler eingeführt. [28] Diese beiden Layer und der Data Access Layer greifen auf einen Common Layer zu.

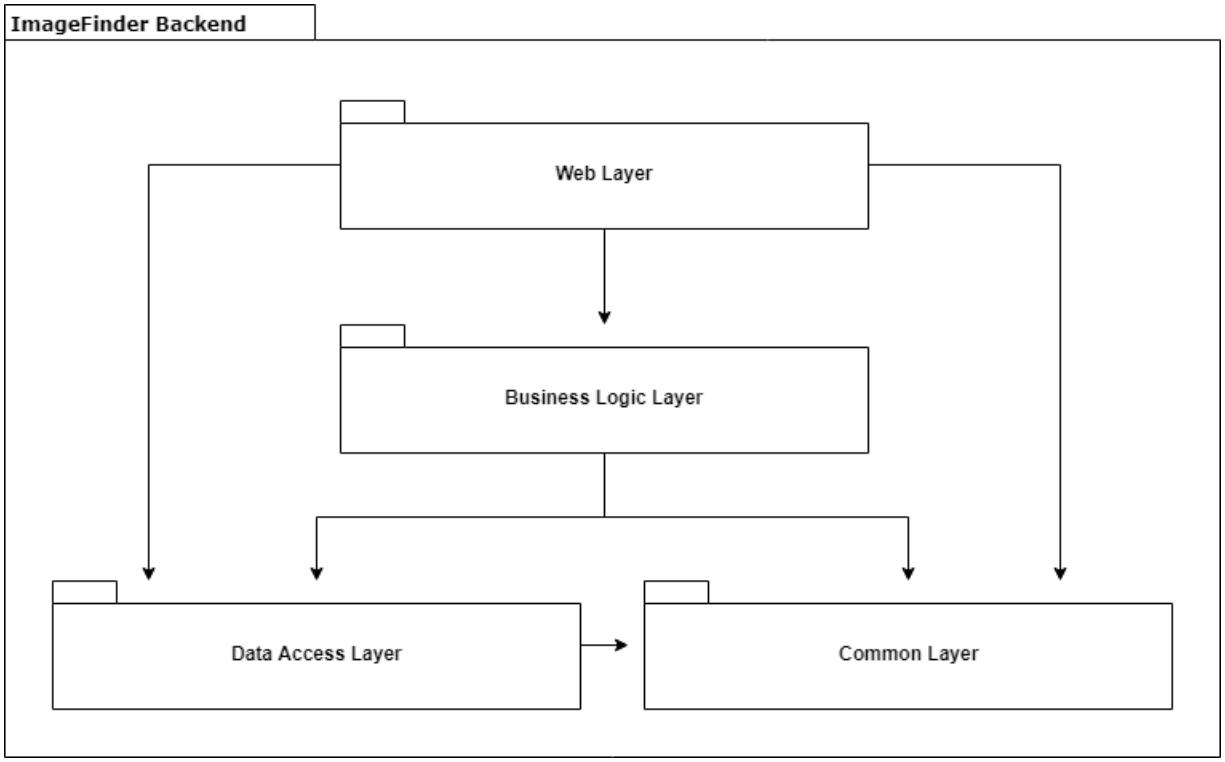


Abbildung 13: Backendarchitektur

Anmerkung. Eigene Darstellung.

Auffällig ist der Zugriff des Web Layers direkt auf den Data Access Layer. Dies wurde aufgrund der überschaubaren Grösse der Applikation entschieden und ist nur in sehr einfachen Fällen zulässig, wenn lediglich von der Datenbank gelesen wird. Abhängigkeiten von einer unten liegenden Schicht zu einer weiter oben liegenden Schicht sind nicht vorhanden.

Web Layer
Der Web Layer nimmt die HTTP-Requests des Frontends (Clients) entgegen und ist für das Rendern von HTML sowie ausliefern von statischen Objekten wie Bilder zuständig. Dies ist in Abbildung 14 ersichtlich.

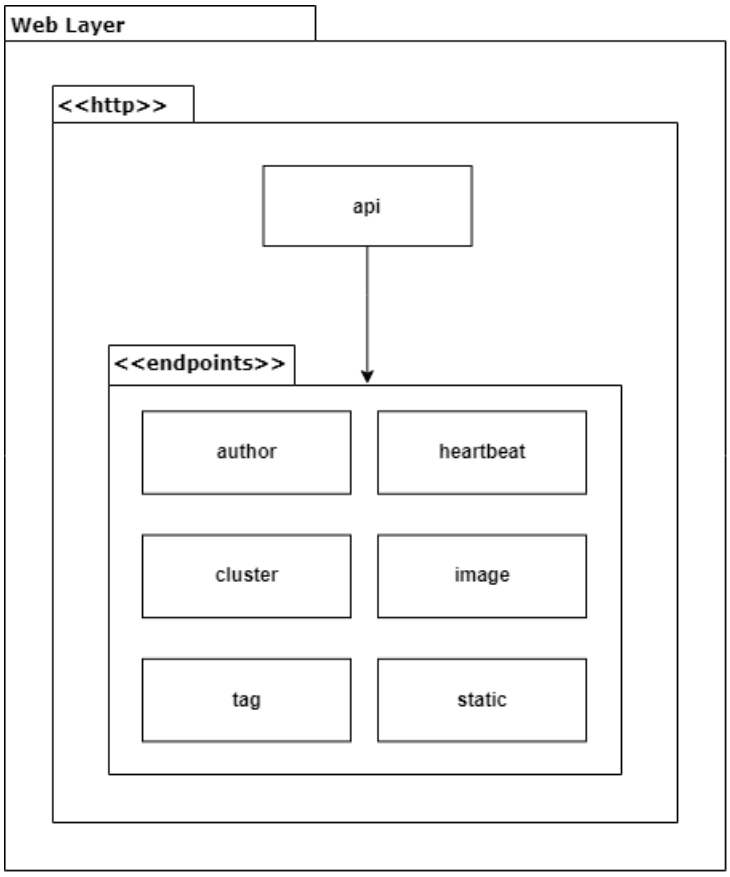


Abbildung 14: Backendarchitektur - Web Layer

Anmerkung. Eigene Darstellung.

Im Web Layer sind die beiden Packages „http" und „endpoints" zu finden. Das „api"-Modul dient lediglich als Hilfe in Form eines Routers, damit Code in die anderen sechs Modulen ausgelagert werden kann. In Tabelle 22 sind die Endpoints genauer beschrieben.

Tabelle 22: Backendarchitektur: Web Layer - endpoints

Endpoint	Beschreibung
author	Abrufen der Fotografen
heartbeat	Überprüfung der Verbindung
cluster	Erstellen eines Clusters
image	Bilder hochladen, aktualisieren
tag	Abrufen der Tags
static	Bilder laden

Anmerkung. Eigene Darstellung.

Common Layer
Wie in Abbildung 15 zu sehen ist, enthält der Common Layer das Package „models" mit sechs Klassen. Diese Klassen dienen der puren Datenhaltung und werden letztendlich über die Endpoints zum Client gesendet. Fowler bezeichnet diese Klassen als Data Transfer Objects [29].

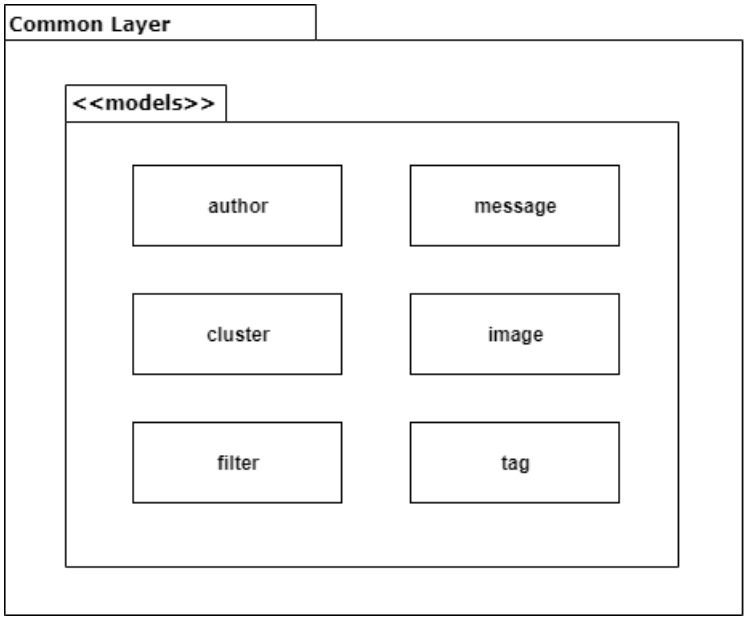


Abbildung 15: Backendarchitektur - Common Layer

Anmerkung. Eigene Darstellung.

Abbildung 16 zeigt die Packages „services", „utils" und „core", die dem Business Logic Layer zugeordnet sind.

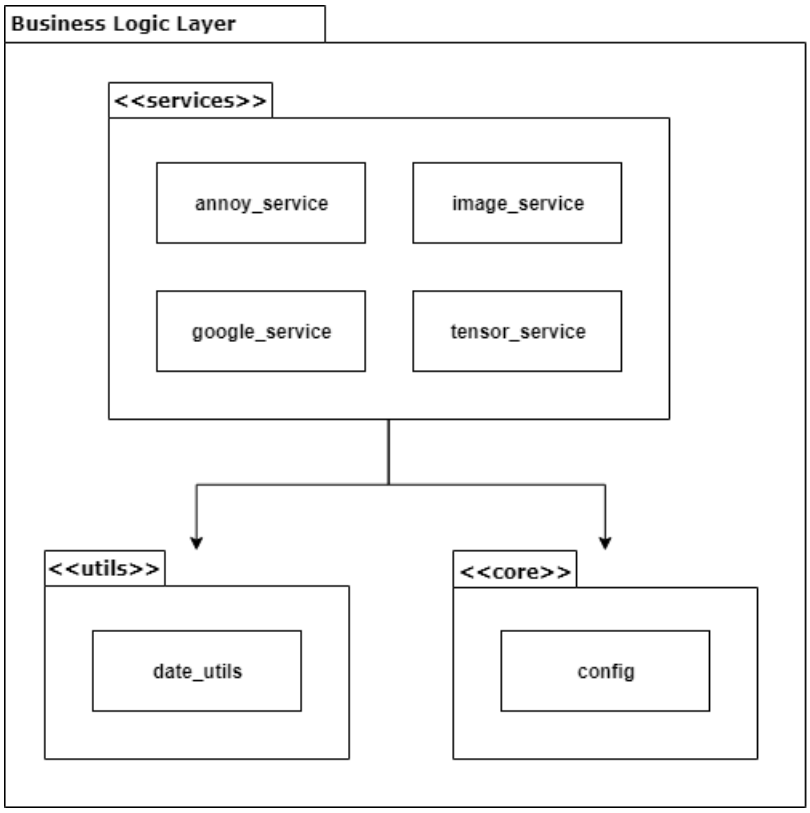


Abbildung 16: Backendarchitektur - Business Logic Layer

Anmerkung. Eigene Darstellung.

Tabelle 23 beschreibt die Grundfunktionalitäten der vier Services.

Tabelle 23: Backendarchitektur: Business Logic Layer - Services

Service	Beschreibung
annoy_service	Baumstruktur erstellen, Clusterbildung, Nearest Neighbors laden
image_service	Verarbeiten, laden, löschen von Bildern
google_service	Kommunikation zur Google-Schnittstelle, die für die textbasierte Suche verwendet wird
tensor_service	Erstellen und Laden der Feature-Vektoren

Anmerkung. Eigene Darstellung.

Data Access Layer

Der Data Access Layer (DAL) dient dazu, um mit der Datenbank zu kommunizieren, bzw. Daten zu persistieren und abzufragen. Wie in Abbildung 17 zu sehen ist, befinden sich die Packages „crud" und „entities" im DAL.

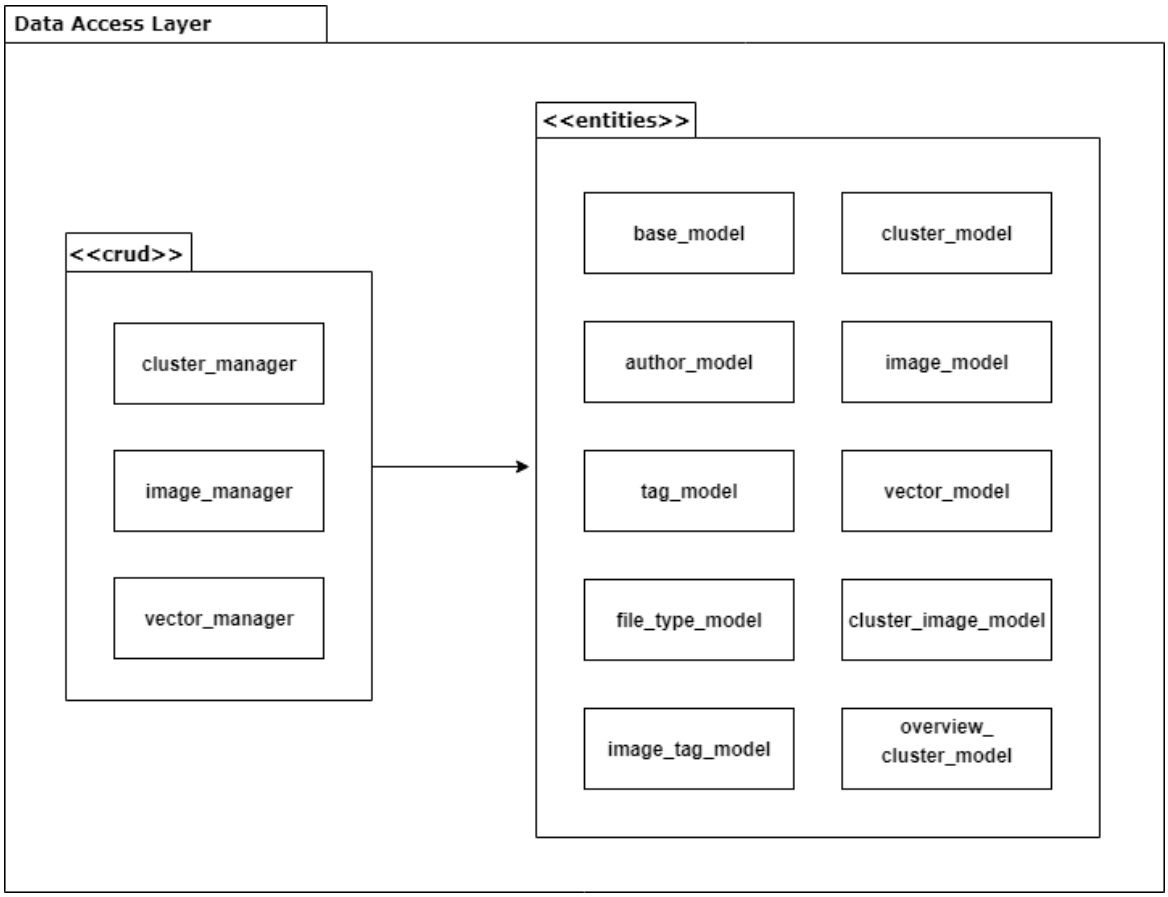


Abbildung 17: Backendarchitektur - Data Access Layer

Anmerkung. Eigene Darstellung.

Crud ist in diesem Sinne als Acronym Create/Read/Update/Delete zu verstehen. Dieses Package beinhaltet die drei Module „cluster_manager", „image_manager" und „vector_manager". Die sogenannten „Manager" sind dafür zuständig, die Entitäten aus dem Package „entities" zu persistieren. Dies geschieht hinter den Kulissen mit dem Object Relational Mapper Peewee. Insgesamt sind zehn Klassen im Package „entities" vorzufinden. In Tabelle 24**Fehler! Verweisquelle konnte nicht gefunden werden.** werden die Zuständigkeiten der drei Managers beschrieben.

Tabelle 24: Backendarchitektur: Data Access Layer – crud

Manager	Beschreibung
cluster_manager	Generieren, speichern, aktualisieren und laden der Cluster von der Datenbank
image_manager	Speichern, aktualisieren und laden der Bilder und Metadaten von der Datenbank
vector_manager	Generieren und laden der Feature-Vektoren von der Datenbank

Anmerkung. Eigene Darstellung.

5.2 Deploymentdiagramm

Die Client-Server-Anwendung wird bei der Evangelisch-reformierten Kirche Horgen auf demselben System installiert. Aus dem Python-Webserver wird ein exe-Artefakt generiert, welches als Windows Services eingebunden werden kann. Dies wird in Abbildung 18 veranschaulicht.

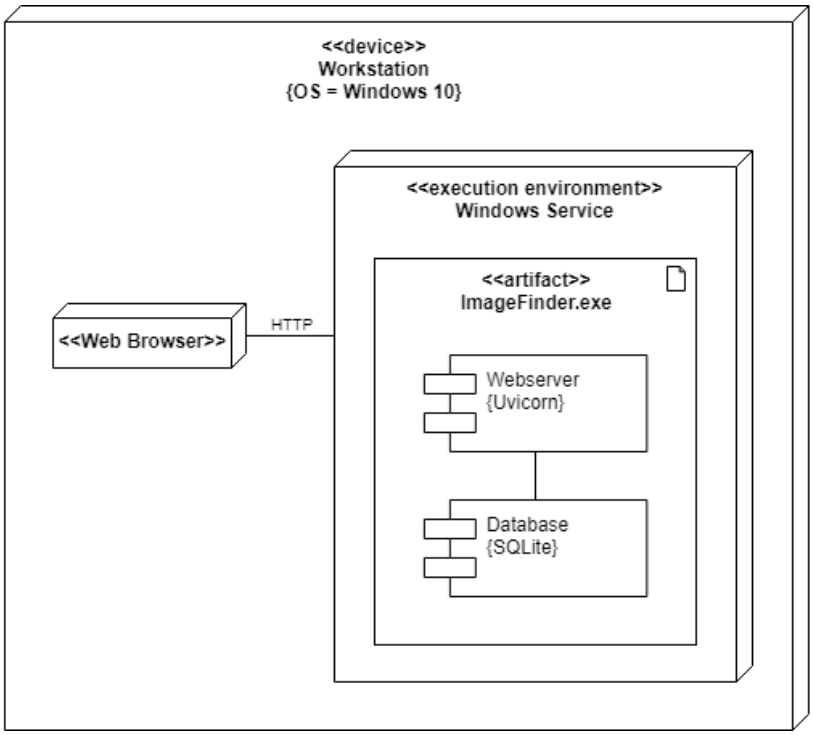


Abbildung 18: Deployment Diagram

Anmerkung. Eigene Darstellung.

5.3 Datenbankmodell

Abbildung 19 zeigt die Entitäten und Beziehungen des Datenmodells. Das Datenbanksystem generiert automatisch die Ids der Entitäten. Im Zentrum steht die “image”-Entität. Boolean-Werte werden von SQLite als Integer behandelt. Die generierten Feature-Vektoren werden als Text in die Entität “vector” persistiert.

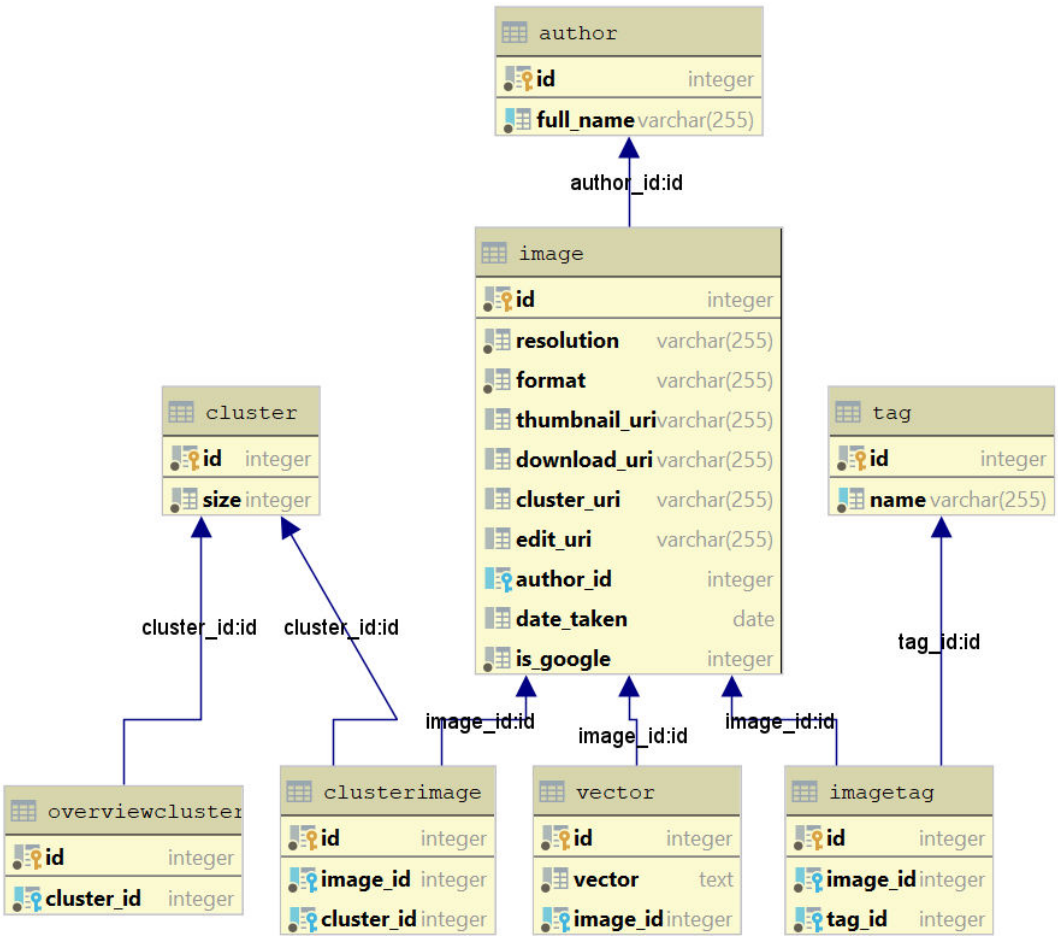


Abbildung 19: Datenmodell aus der Datenbank

Anmerkung. Eigene Darstellung.

5.4 Umsetzung der Benutzeroberfläche

Das Frontend soll für die Benutzerin oder den Benutzer ansprechend sein. Aus diesem Grund wurde zum Webframework Vue.js zusätzlich das Material Design Component Framework Vuetify [25] verwendet. Nicht nur erfüllt Vuetify heutige Design-Standards, die Gestaltung der Screens nahm durch vorgefertigte Komponenten markant weniger Zeit in Anspruch. In Anhang N sind Ausschnitte der fertigen Benutzeroberfläche ersichtlich. Die einzelnen Funktionalitäten sind kurz beschrieben.

5.5 Softwaredesign-Entscheidungen

Nachfolgend sind relevante Softwaredesign-Entscheidungen aufgelistet.

Upload von Bildern
Die im Browser zum Hochladen ausgewählten Bilder, werden einzeln und nicht als Batch-Upload zum Backend übertragen. Das verwendete Backend-Framework FastAPI hat diese Funktionalität erst seit dem 14. April 2019 implementiert [30]. Eine Umstellung scheint sich nicht zu lohnen, da einerseits die Möglichkeit verloren geht, der benutzenden Person ein granulares Feedback zu geben und andererseits der Round-Trip für einen HTTP-Request mit dem Deployment bei der Evangelisch-reformierten Kirche Horgen vernachlässigt werden kann. Ausserdem wäre dies in einem verteilten Deployment aus Security-Sicht eher bedenklich, wenn eine beliebig grosse Payload gesendet werden kann. Attacken wie Distributed-Denial-of-Service (DDoS) wären so leichter durchführbar.

Atomic Datenbank Updates
Eine Datenbank-Transaktion gilt de facto als teure Operation. Deshalb werden, wie in Abbildung 20 zu sehen ist, Bulk-Inserts verwendet, welche von einer Transaktion ummantelt sind. Dies wird in der Peewee-Dokumentation als „good practice" [31] bezeichnet.

```
with database.atomic():
    for batch in chunked(batches, config.DB_CHUNK_SIZE):
        Overview_Cluster_Table.insert_many(
            batch, fields=[Overview_Cluster_Table.cluster_id]
        ).on_conflict_replace().execute()
```

Abbildung 20: Atomic Updates - Ausschnitt aus backend/app/crud/cluster_manager.py

Anmerkung. Eigene Darstellung.

Für SQLite ist zu beachten, dass maximal 999 gebundene Variablen in einer SQL-Abfrage zulässig sind [32].

Bilder und Thumbnails als Static Files
Alle Bilder sowie Thumbnails werden beim Webserver als Static Files eingebunden. Dies ist damit zu begründen, dass diese nicht serverseitig generiert werden müssen, bzw. ihr Inhalt sich nicht ändert. Die Performanz wird leicht gesteigert, da kein Request Objekt [33] erstellt werden muss. Darüber hinaus können die Static Files in einem komplexeren Deployment auch von einem leistungsfähigeren Proxy wie Nginx ausgeliefert werden.

Thumbnails
Ein Bild der Evangelisch-reformierte Kirche Horgen ist im Schnitt rund 3.6 Megabyte gross. Bei einer Anzeige in der Weboberfläche von Beispielsweise zehn Bildern würden somit vom Server zum Client mindestens 36 Megabyte Payload übertragen werden. Um dem entgegen zu wirken, wird für jedes Bild ein „Thumbnail", bzw. eine verkleinerte Version erstellt. Ein Client erhält somit lediglich beim expliziten Download die nicht minimierte Version. Somit kann die Grösse eines übertragenen Bildes um einen Faktor von über 112 auf 0.032 Megabyte verkleinert und insgesamt die Payload bei zehn Bildern auf 0.32 Megabyte reduziert werden.

Lazy Loading

Die HTTP-Schnittstelle der Bilderabfrage wird so definiert, dass bei einem Request lediglich die URI's der Thumbnails in die Response geschrieben werden. Ein Client kann diese URI's in der Weboberfläche in einem HTML-Tag einbinden, wie dies in Abbildung 21 dargestellt wird.

```
<v-img
  :src="item.image.thumbnail_uri"
  …
>
```

Abbildung 21: HTML Client - Request von Thumbnails

Anmerkung. Eigene Darstellung.

Dies ermöglicht ein einfaches Lazy-Loading, ohne dass der gesamte Payload auf einmal übertragen werden muss und überwiegt den Nachteil, dass jeweils pro Thumbnail ein weiterer HTTP-Request notwendig ist.

URI-Schema und Persistierung

Die verwendete SQLite Datenbank wurde auf Abfrage-Geschwindigkeit optimiert. Sämtliche URI's werden wie in Abbildung 22 dargestellt, persistiert.

thumbnail_uri	download_uri
http://127.0.0.1:8000/static/thumbnails/1.JPG	http://127.0.0.1:8000/static/images/1.JPG

cluster_uri	edit_uri
http://127.0.0.1:8000/api/cluster/1	http://127.0.0.1:8000/api/image/1

Abbildung 22: Datenbankausschnitt der Tabelle image

Anmerkung. Eigene Darstellung.

In Abbildung 23 ist zu sehen, dass eine „thumbnail_uri" aus den sechs Teilen scheme, host, port, path, id und file extension besteht.

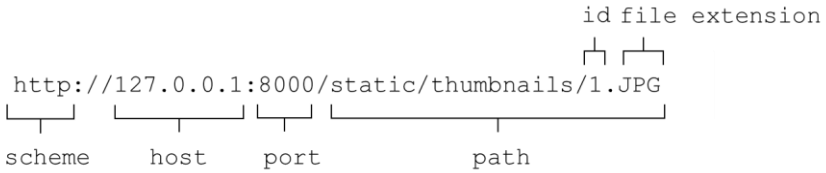


Abbildung 23: Persistierung von URI's

Anmerkung. Eigene Darstellung.

Die Teile „scheme”, „host” und „port” werden aus dem Config-File (settings.toml) gelesen und entsprechen der URL, unter der der Server erreichbar ist. Im Gegenzug wird „path” beim Upload eines Bildes generiert. Die „id” entspricht hierbei der tatsächlichen id in der Datenbank. Die Persistierung der URI's nach diesem Schema geschieht allerdings auf Kosten von Redundanz, Portierbarkeit und Skalierbarkeit. Dies ist in Tabelle 25 beschrieben.

Tabelle 25: Nachteile der Persistierung der URI's nach scheme, host, port und path

Nachteil	Beschreibung
Redundanz	Die Tabelle „image” kann das Format (in diesem Beispiel „JPG”) bereits aus dem Attribut „format” herauslesen.
Portierbarkeit	Die Datenbank lässt sich nicht auf einen neuen Host deployen, ohne dass zuerst alle bestehenden Einträge geändert wurden.
Skalierbarkeit	Die gleiche Applikation kann nicht mehrmals nutzbringend deployed und skaliert werden. Clients würden immer noch URI's mit demselben Host und Port erhalten.

Anmerkung. Eigene Darstellung.

Demgegenüber stehen folgende Vorteile:

- eine SQL-Query braucht keine zusätzlichen Joins auf andere Tabellen
- URI's müssen bei einem HTTP-Request nicht generiert werden, sondern stehen bereits zur Verfügung
- ein HTTP-Client kann bequem die erhaltene URI aufrufen, ohne Änderungen vornehmen zu müssen

Die Vorteile überwiegen die Nachteile im spezifischen Deployment der Evangelisch-reformierten Kirche Horgen. Bei einem anderen Anwendungsszenario müssten die Vor- und Nachteile neu abgewägt werden.

Konfiguration

Parameter, die zur Konfiguration der Software dienen, sollen vom Source Code separiert werden. Dies entspricht der „strict separation of config from code” Empfehlung aus The Twelve-Factor App [34]. Parameter, wie beispielsweise der Port auf dem der Webserver startet, können später bequem geändert werden, ohne dass der Python-Code neu kompiliert werden muss.

HTTP Response Codes
Mit Hilfe von HTTP Status Codes kann ein Server mitteilen, ob die Anfrage des Clients erfolgreich war. In Tabelle 26 sind die Status Codes definiert, die hauptsächlich verwendet werden.

Tabelle 26: HTTP Response Codes

Status Code[35]	**Beschreibung**
200 Ok	Dieser Status Code gilt für alle Endpoints, auf die erfolgreich lesend (HTTP GET) zugegriffen wurde. Ausnahme: Eine bestehende Ressource wurde erfolgreich ersetzt (HTTP PUT).
201 Created	Wird zurückgegeben, wenn eine neue Ressource erfolgreich angelegt wurde (HTTP POST).
422 Unprocessable Entity	Request wurde nicht verarbeitet, da es einen Validierungsfehler gab.
429 Too Many Requests	Der Endpoint, um Bilder von Google abzufragen, wurde zu oft abgefragt. Dies ist Aufgrund der Restriktion von Google gegeben.[11]

Anmerkung. Eigene Darstellung.

5.6 Client-Server-Kommunikation

Nachfolgend wird anhand eines einfachen Beispiels illustriert, wie ein Client mit dem Backend kommuniziert. Dabei wird das Beispiel gezeigt, wie ein Client sich alle Namen der im Backend gespeicherten Autoren beschafft.

In Abbildung 24 erstellt ein Client ein Objekt des Typs „AuthorsApi". Diese Klasse gehört zu den Services und wurde automatisch aus dem Backend generiert (vgl. Abbildung 12: Frontendarchitektur - Service Layer). JavaScript kennt keine statischen Typen, deshalb wurde für die Services jeweils Flow [36] eingesetzt. Dies erlaubt ausserdem Code Completion [37].

```
loadAuthors() {
  const api: AuthorsApi = new AuthorsApi();
  const callback = (error, authors: AuthorList) => {
    if (error) {
      console.error(error);
    } else {
      authors.authors.forEach((a) => {
        this.items.push(a.name);
      });
    }
  };
  api.readAuthors(callback);
},
```

Abbildung 24: JavaScript-Frontend - frontend/src/components/AuthorSelecting.vue

Anmerkung. Eigene Darstellung.

Wurde der Request aus Abbildung 24 gesendet, wird der in Python geschriebene HTTP-Endpoint angesprochen. Dies ist in Abbildung 25 zu sehen.

```
router = APIRouter()


@router.get(
    "/api/authors",
    operation_id="readAuthors",
    response_model=AuthorList,
    tags=["authors"],
)
async def read_authors() -> AuthorList:
    authors: AuthorList = image_manager.get_authors()
    return authors
```

Abbildung 25: Python-Backend - backend/app/http/endpoints/author.py

Anmerkung. Eigene Darstellung.

Der Endpoint greift direkt auf einen Manager (vgl. Abbildung 17: Backendarchitektur - Data Access Layer, Seite 45) zu, was nur lesend erlaubt ist. Dieser liefert wie in Abbildung 26 zu sehen ist, ein Objekt vom Typ „AuthorList" (vgl. Abbildung 15: Backendarchitektur - Common Layer) zurück.

```python
@database.connection_context()
def get_authors() -> AuthorList:
    authors: List[Author] = []
    db_author_name = Author_Table.select()
    for full_name in db_author_name.iterator():
        author: Author = Author(name=full_name.full_name)
        authors.append(author)
    return AuthorList(authors=authors)
```

Abbildung 26: Python-Backend - backend/app/crud/image_manager.py

Anmerkung. Eigene Darstellung.

Die Serialisierung nach JSON wird jeweils vom Framework FastAPI übernommen. In diesem Fall werden alle „authors" wie es Abbildung 27 illustriert, an den Client als Response zurück gegeben.

```json
{
  "authors": [
    {
      "name": "string"
    }
  ]
}
```

Abbildung 27: JSON-Response aus HTTP GET api/authors

Anmerkung. Eigene Darstellung.

Weitere Schnittstellendefinitionen können aus dem Anhang O entnommen werden. Anmerkung: Das in Python geschriebene Backend ist vollständig typisiert, was seit Python 3.5 möglich ist.[38]

5.7 Entfernen von Duplikaten

Löschen von Duplikaten und nicht kompatiblen Bildern Die Bildersammlung der Evangelisch-reformierten Kirche Horgen umfasst insgesamt rund 14‘800 Bilder (Stand Mai 2019). Dateien ohne Bildformat (PDF, ZIP, MP4 etc.) wurden vorgängig entfernt. Von den Bildern werden insgesamt 11‘734 Bilder als valide eingestuft und für die Software verwendet. Insgesamt wurden 3‘051 Duplikate und fünf Bilder mit einem Bildformat, welches TensorFlow nicht unterstützt gefunden und entfernt.

Duplikate erkennen
Die Duplikate werden mit der Distanz-Matrix, vor der Erstellung der Cluster, identifiziert. Wird in der Distanzmatrix bei einem Feature-Vektor-Paar der Wert 1 berechnet, handelt es sich um ein Duplikat. Selbst kleinste Anpassungen an einem Bild, wie zum Beispiel ein kleiner, gezeichneter Strich, verändern den Wert in der Matrix.

6 Testing

Dieses Kapitel widmet sich dem Testen der Software. Dabei wurden verschiedene Verfahren eingesetzt. Zuerst wird das Programm in einem Systemtest als Black Box behandelt. Danach folgen White-Box-Tests mit Blick auf den Source Code. Die anschliessenden Usability-Tests widmen sich der Benutzeroberfläche. Manuelle Tests werden durchgeführt, die das implementiere Clustering-Verfahren untersuchen. Ein Performanz-Profiling soll die zugrunde liegenden Funktionen analysieren und Auskunft über die Maximallast geben. Letztendlich wird noch die 12 Factor App Methode angewendet.

6.1 Systemtest

Nachfolgend werden Systemtests im Black Box Verfahren durchgeführt. Die Tests wurden anhand den im Kapitel 2.1 definierten Use Cases festgelegt. Um die Systemtests beliebig wiederholen zu können, werden in Tabelle 27 die Testdaten definiert und in Tabelle 28 ausgewertet.

Tabelle 27: Definition der Testdaten für die funktionalen Anforderungen

#	Testdaten
TD1	Ordner „Jugend” der Evangelisch-reformierten Kirche Horgen
TD2	Leeres Textfile als .png abgespeichert
TD3	Bild „spanien.jpg” aus dem Ordner „Jugend/12 Veloreise Andalusien”
TD4	Bild „DNSCN4010.jpg” aus dem Ordner „Katzen 2014”
TD5	Datum von: {heute}
TD6	Schlagwort: Kirchenglocke
TD7	Schlagwort: „aslkdfhasdkljghasdj”
TD8	Tag: Landschaft, Fotograf: HSR, Erstellungsdatum 1.2.2002
TD9	Bild „Blume.jpg” aus dem Ordner “Natur/Blumen“>
TD10	Datum von: 20.03.2019
TD11	Datum von: 01.01.1902, Datum bis: 01.01.1903

Anmerkung. Eigene Darstellung.

Tabelle 28: Systemtest - Auswertung

Durchführung: 28. Mai 2019, Version: Release v2019-05-28						
#	**Use Case**	**Scenario**	**Testdaten**	**Erwartetes Verhalten**	**Ergebnis**	**Implementiert?**
T1	UC1	Basic Flow	TD1	Die Bilder, die dazugehörigen Metadaten und Feature-Vektoren werden korrekt abgespeichert.	ok	ja
T2		2a	TD1	Dialog wird angezeigt.	ok	ja
T3		2b	TD2	File wird ignoriert.	ok	ja
T4		2c	TD3	Bild wird ignoriert. Anzahl der Feature-Vektoren und Bilder hat sich nicht verändert.	ok	ja
T5		3a	TD4, TD5	In der Ansicht erscheint das Bild mit heutigem Datum.	ok	ja
T6		2d	-	Tag wird vorgeschlagen.	-	nein
T7		2e	-	Anlass wird vorgeschlagen.	-	nein
T8	UC2	Basic Flow	TD1	Cluster sind als solche gekennzeichnet.	ok	ja
T9		3a	TD3	Ein einziges Bild wird nicht als Cluster angezeigt.	ok	ja
T10	UC3	Basic Flow	TD1	Die dem Bild am ähnlichsten Objekte werden angezeigt.	ok	ja
T11		1a	TD1, die ersten 4 Bilder selektiert	Die den Bildern am ähnlichsten Objekte werden angezeigt.	ok	ja
T12	UC4	Basic Flow	TD6	Google Bilder erscheinen gekennzeichnet.	ok	ja
T13		2a	-	Dialog mit Hinweis erscheint.	ok	ja
T14	UC5	Basic Flow	TD3	Bild wird in der Originalgrösse auf der Festplatte gespeichert.	ok	ja
T15		2a	-	Option „Small", „Medium", „Large" erscheint zum Download.		nein
T16	UC6	Basic Flow	TD3, TD8	Die eingegebenen Metadaten werden korrekt zum Bild abgespeichert.	ok	ja
		1a	TD3, TD8, TD9	Die eingegebenen Metadaten werden korrekt zu den Bildern abgespeichert.	ok	ja
T17	UC7	Basic Flow	TD1, TD10	TD3 erscheint angezeigt.	ok	ja
T18		2a	TD1, TD11	Ein Hinweis wird angezeigt, dass zu diesem Filter kein Suchresultat gefunden werden konnte.	ok	ja

T19	UC8	Basic Flow	TD1	Weitere Suchresultate werden angezeigt.	ok	ja
T20		2a	TD1	Es wird signalisiert, dass noch weitere Bilder geladen werden.	ok	ja
T21		2b	TD1	Es erscheint kein Hinweis, dass noch weitere Bilder geladen werden.	ok	ja
T22	UC9	Basic Flow	TD1	Alle Bilder im Cluster werden angezeigt	ok	ja
T23	UC10	Basic Flow	-	Die heutig importierten Bilder werden angezeigt.	-	nein
T24	UC11	Basic Flow	-	Resultat der vorherigen Suche wird angezeigt.	-	nein
T24	UC12	Basis Flow	-	Angezeigte Bilder verfügen über das neue Tag.	-	nein

Anmerkung. Eigene Darstellung.

6.2 Unit- und Integrationtest

Um die Software zu testen, wurde eine Kombination aus Unit- und Integrationtests gewählt. Die Tests wurden jeweils nach dem Arrange, Act, Assert (AAA) Pattern [39] geschrieben.

Backend
Eine Pytest-Fixture [40] stellt sicher, dass vor jedem Test die SQLite-Datenbank neu erstellt wird. Diese Fixture ist in Abbildung 28 dargestellt.

```
@pytest.fixture()
def use_test_database() -> Generator:
    models: Any = get_db_models()
    with database.atomic():
        database.create_tables(models)
    try:
        yield
    finally:
        with database.atomic():
            database.drop_tables(models)
```

Abbildung 28: Pytest fixture - backend/app/tests/conftest.py

Anmerkung. Eigene Darstellung.

Weitere Tests, wie in Abbildung 29 ersichtlich, definieren eine Abhängigkeit auf die Fixture. Pytest wird dann zur Laufzeit per Dependency Injection die Abhängigkeit einfügen.

```
def test_save_image(use_test_database: Generator) -> None:
    # arrange
    file_extension: str = ".jpg"
    resolution: str = "200x200"
    image: Image = Image(format=file_extension, resolution=resolution)
    vector: ndarray = ndarray([3])
    author: str = "Scrooge McDuck"
    tags: Set[str] = {"Money", "Cash"}

    # act
    image_id: int = save_image(image, vector, author, tags)
    db_image: Image = read_image_by_id(image_id)

    # assert
    assert db_image.format == file_extension
    assert db_image.resolution == resolution
    assert db_image.author.full_name == author
    assert db_image.download_uri
    assert db_image.thumbnail_uri
    assert db_image.edit_uri
    assert not db_image.is_google
```

Abbildung 29: Ausschnitt aus backend/app/tests/crud/test_image_manager.py

Anmerkung. Eigene Darstellung.

Frontend

Abbildung 30 zeigt einen Unittest der Component FilterEdit.

```
test('test if correct filter is emitted after button clicked', () => {
  // arrange
  const wrapper = mount(FilterEdit);
  const filters = {
    dateFrom: '2019-02-01',
    dateTo: '2019-03-01',
    author: 'Donald Duck',
    tags: ['Duck'],
    color: 'red',
  };
  wrapper.setData({ filters });

  const applyFilterButton = wrapper.find('.v-btn:nth-child(3) > .v-btn__content');
  const applyFilterButtonExists = applyFilterButton.exists();

  // act
  applyFilterButton.trigger('click');

  // assert
  expect(applyFilterButtonExists).toBeTruthy();
  expect(wrapper.emitted().reloadAfterFilterChanged[0]).toEqual([filters]);
});
```

Abbildung 30: Ausschnitt aus frontend/tests/unit/FilterEdit.spec.js

Anmerkung. Eigene Darstellung.

6.3 Usability-Tests

Es wurden insgesamt drei Usability-Tests durchgeführt. Bezüglich der Protokolle wird auf Anhang J verwiesen. Bei den Tests wurden folgende Mängel festgestellt:

1. Wenn eine Suche durchgeführt und anschliessend ein Filter verwendet wird, werden nicht alle Bilder mit dieser Eigenschaft angezeigt. Es erscheinen nur diejenigen Bilder, welche ähnlich zum Referenzbild sind. Möchte man trotzdem alle Bilder anzeigen, muss der Browser mit der Taste F5 aktualisiert und so sämtliche Referenzbilder zurückgesetzt werden.
2. Das Format, mit welchem das Datum eingegeben werden sollte, war nicht klar. Auch war nicht klar, dass wenn nach einem bestimmten Datum gefiltert werden möchte, sowohl das von als auch das bis Datum eingegeben werden muss (Beispiel: Suche nach Bildern vom 10. September 2018, Eingabe Datum von: 10.09.2018 und Eingabe Datum bis: 10.09.2018, notwendig).
3. Obwohl die nach einer Suche angezeigten, besten Treffer das suchende Merkmal nicht enthalten, werden die Bilder geclustert.

Konsequenzen

1. Um alle Referenzbilder zurückzusetzen, soll mit einem Klick auf das Logo der Browser aktualisiert werden können. Danach ist es möglich mit dem Filter alle Treffer aus der Bildersammlung zu finden.
2. Das Format des Datums sowie eine allfällige Falscheingabe wird in der Software angezeigt. Diesbezüglich wurden keine Änderungen vorgenommen. Beim Titel zum Datum wurde zusätzlich „Von – Bis” ergänzt.
3. Die erste Zeile der Bilderansicht soll nicht geclustert werden.

6.4 Nearest Neighbor Search Tests

Um die Qualität der Ergebnisse zu überprüfen wurde das Verfahren aus der Studienarbeit übernommen. Mit Bildern von der Google Bildersuche wurde mit der lokalen Bildersammlung eine Suche nach ähnlichen Bildern gestartet. Standardmässig verwendet die Software vier Google Bilder für eine Bildersuche. Da häufig von Google nicht alle Bilder das gesuchte Merkmal enthalten, wurde der Suchbegriff angepasst, bis ein passendes Bild von Google gefunden werden konnte. Aufgrund dessen und aufgrund der ständig wechselnden, zurückgegebenen Bilder von Google, kann das Ergebnis nicht eins zu eins in der Software quergeprüft werden. Im direkten Vergleich wurde getestet, ob der Prototyp und der ImageFinder das gleiche Ergebnis lieferten. Verwendet wurde die Bildersammlung mit Stand Mai 2019.

6.4.1 Testbedingungen

Geprüft wurden lediglich die zehn besten Treffer. In der Y-Achse ist die Anzahl Bilder, welche das gesuchte Merkmal enthielten zu sehen. In der X-Achse wurden die einzelnen Tests nach Merkmal gegliedert. Duplikate, welche im Prototypen nicht gelöscht werden, wurden für den Test nicht zu den Ergebnissen dazugezählt.

Für den Annoy Algorithmus wurden folgende Parameter-Werte verwendet:

- n_trees: 300
- metric: euclidean
- dimensions: 2048

Bezüglich der Bestimmung des Parameters n_trees wird auf Abschnitt 6.5.3 verwiesen.

6.4.2 Test mit einem Referenzbild

Aus Abbildung 31 können die Resultate mit einem Referenzbild entnommen werden.

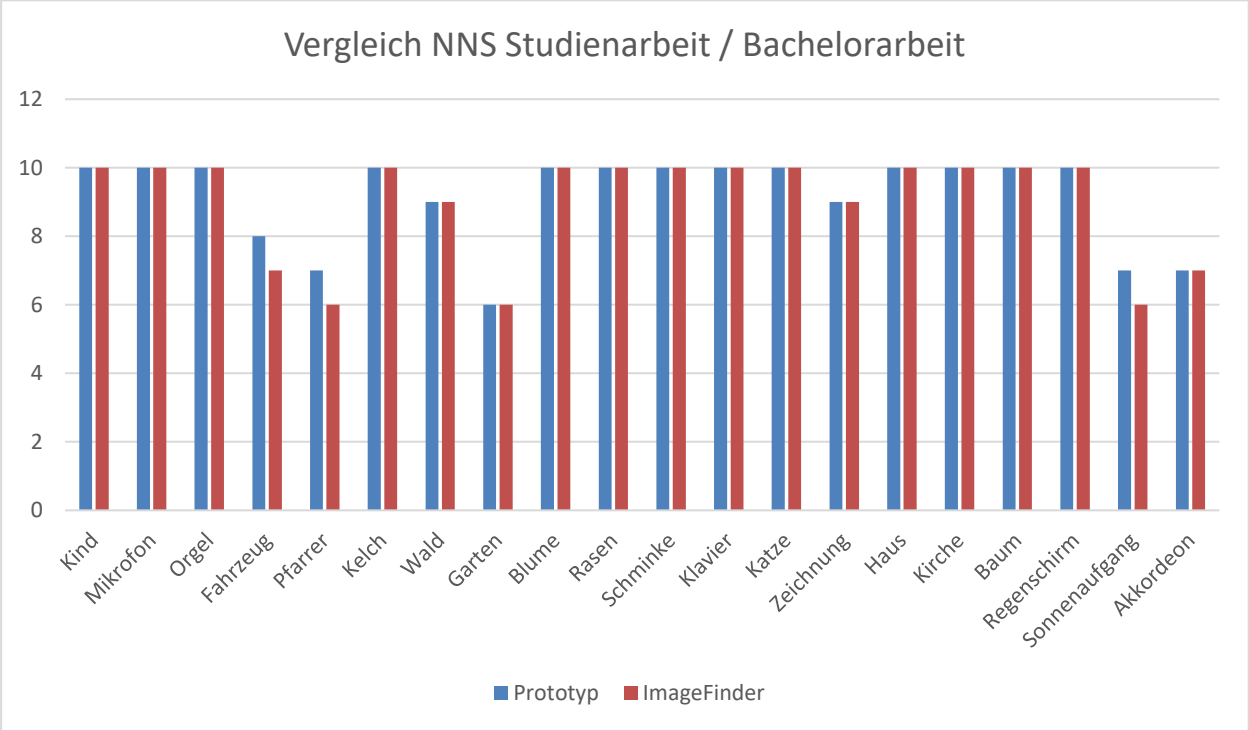


Abbildung 31: Vergleich NNS Studienarbeit / Bachelorarbeit

Anmerkung. Eigene Darstellung.

6.4.3 Erkenntnis mit einem Referenzbild

Der direkte Vergleich zeigte, dass die Ergebnisse grösstenteils identisch sind. Teilweise ergaben sich kleine Abweichungen der angezeigten Bilder.

6.4.4 Tests mit mehreren Referenzbildern

Das Testverfahren wurde aus der Studienarbeit übernommen, jedoch für die Suche mit mehreren Referenzbildern leicht angepasst. Aus den Usability-Tests und aus den Gesprächen mit den Mitarbeitenden der Evangelisch-reformierten Kirche Horgen, hat sich gezeigt, dass mehrere Referenzbilder dazu verwendet werden, um eine Suche zu verfeinern. In der Studienarbeit wurde getestet, ob Merkmale von verschiedenen Referenzbildern auf einem Bild gefunden werden können.
Folglich wurde die Bildersammlung nach einem Referenzbild durchsucht, bei welchem die ersten zehn Bilder der Suchergebnisse nicht das gesuchte Merkmal enthalten.

Als Beispiel wird ein Referenzbild mit einer Kutsche als Merkmal ausgewählt und eine Suche gestartet. In den ersten zehn Bilder der Suchergebnisse erscheinen fünf Bilder mit einer Kutsche. Die anderen Bilder enthalten das Merkmal nicht. Nun wird zum ersten Referenzbild ein zweites mit dem Merkmal Kutsche angewählt. Mit beiden Bildern wird wiederum eine Suche gestartet. Auf den ersten zehn Bildern der Suchergebnisse ist nun auf sieben eine Kutsche zu sehen. Somit konnten durch die Verwendung eines weiteren Referenzbildes zwei weitere Treffer gefunden werden. In Abbildung 32 ist in der Y-Achse die bei den Tests gefundenen, zusätzlichen Treffer zu sehen. Werden bei dem vorherigen Beispiel weniger als fünf Treffer angezeigt, hat sich das Resultat verschlechtert und der Wert wird entsprechend negativ angezeigt.

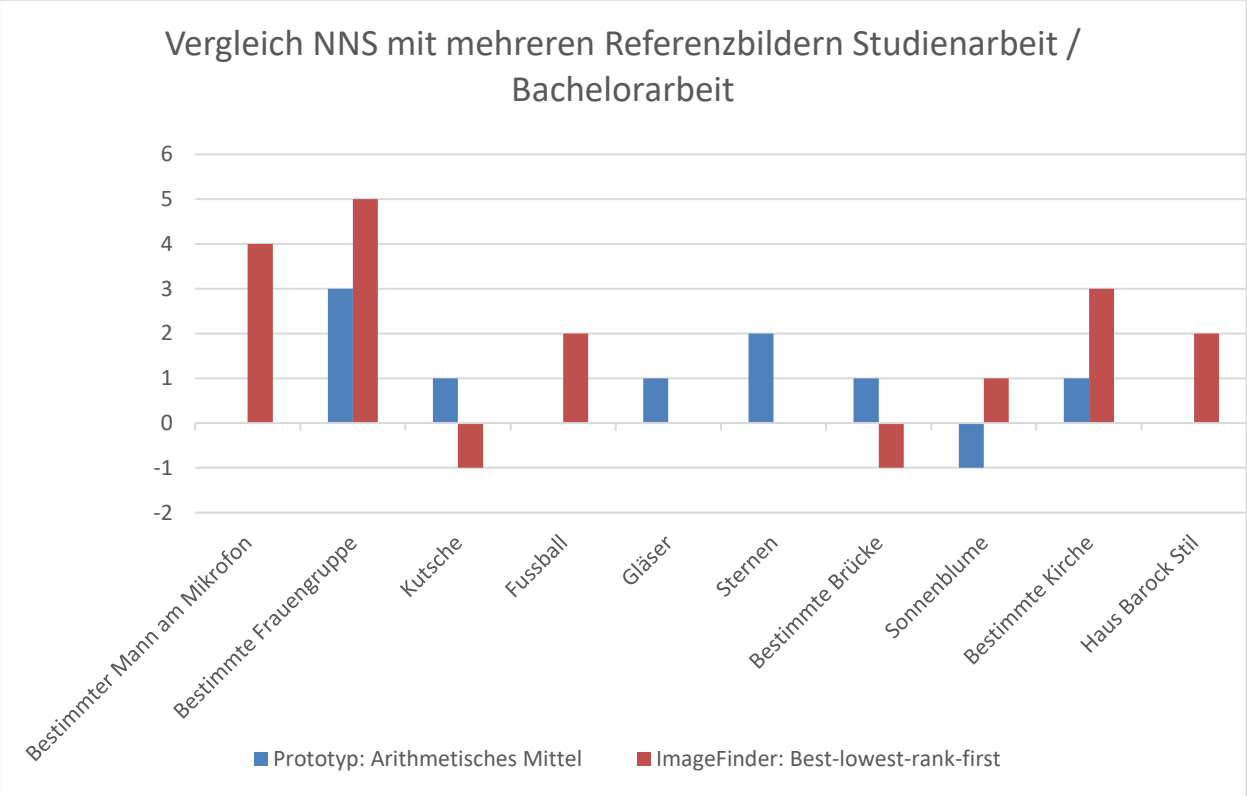


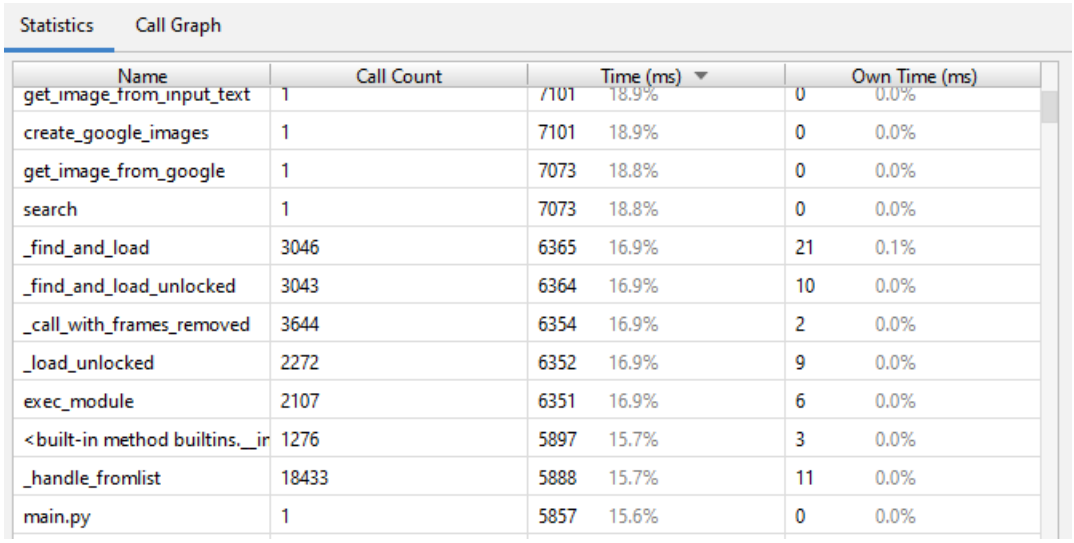
Abbildung 32: Vergleich NNS mit mehreren Referenzbildern

Anmerkung. Eigene Darstellung.

Auswertung
Die Ergebnisse haben sich mit dem „Best-lowest-rank-first“ Verfahren um 87.5% verbessert. Dies muss jedoch kritisch hinterfragt werden, da der Algorithmus nicht bei allen Merkmalen eine Verbesserung hervorbringt. Die im Test verwendeten Merkmale wurden zufällig ausgewählt und können bei einer Veränderung die Testergebnisse signifikant beeinflussen.

6.5 Performanztests

Beim ImageFinder sollte vor allem die Performanz bei der Suche nach ähnlichen Bildern verbessert werden. Dieser Use Case wird durch die nutzende Person am meisten durchgeführt und hat einen wesentlichen Einfluss auf die Usability der Software. Für die Performanztests wurde jeweils der Mittelwert aus zehn Versuchen genommen. Die gemessenen Zeiten wurden aus dem integrierten Profiler von PyCharm entnommen. [41] Ein Ausschnitt eines „Profilings“ ist in Abbildung 33 ersichtlich. Verwendet wurde die Bildersammlung mit Stand Mai 2019.


Statistics Call Graph

Name	Call Count	Time (ms)		Own Time (ms)	
get_image_from_input_text	1	7101	18.9%	0	0.0%
create_google_images	1	7101	18.9%	0	0.0%
get_image_from_google	1	7073	18.8%	0	0.0%
search	1	7073	18.8%	0	0.0%
_find_and_load	3046	6365	16.9%	21	0.1%
_find_and_load_unlocked	3043	6364	16.9%	10	0.0%
_call_with_frames_removed	3644	6354	16.9%	2	0.0%
_load_unlocked	2272	6352	16.9%	9	0.0%
exec_module	2107	6351	16.9%	6	0.0%
<built-in method builtins.__ir	1276	5897	15.7%	3	0.0%
_handle_fromlist	18433	5888	15.7%	11	0.0%
main.py	1	5857	15.6%	0	0.0%

Abbildung 33: Auschnitt Profiler von Pycharm

6.5.1 Nearest Neighbor Search – Lokale Referenzbilder

Mit einer aufsteigenden Anzahl lokal gespeicherter Referenzbilder wurde die Berechnungszeit des Prototypen mit der des ImageFinders verglichen. Aus Tabelle 29 ist zu entnehmen, dass die Berechnungszeit durch die Verwendung des Annoy-Algorithmus gut siebenmal schneller ist. Steigt die Anzahl der Referenzbilder, so verbessert sich dieser Wert exponentiell.

Tabelle 29: Vergleich NNS Prototyp / ImageFinder

Anzahl Referenzbilder	Berechnungszeit in Sekunden Prototyp	Berechnungszeit in Sekunden ImageFinder
1	8.16	1.05
2	8.91	1.13
5	13.32	1.28
10	20.78	1.69

Anmerkung. Eigene Darstellung.

Auswertung
Durch die Verwendung des Annoy-Algorithmus konnte eine massive Performanz Optimierung festgestellt werden. Dies ist für Benutzerinnen und Benutzer durchaus spürbar.

6.5.2 Nearest Neighbor Search – Referenzbilder von Google

Durch die Verwendung der Google Custom Search API haben sich die Antwortzeiten gegenüber dem Prototypen verändert. Diesbezüglich wurden Zeitmessungen durchgeführt, um die beiden Programme miteinander zu vergleichen. Veränderungen der Internetgeschwindigkeit wurden nicht weiter berücksichtigt. Aus Tabelle 30 ist zu entnehmen, dass die Google Custom Search API bei vier Referenzbildern rund eine Sekunde langsamer ist.

Tabelle 30: Vergleich Google Suche Prototyp / ImageFinder

Anzahl Bilder von Google	Antwortzeit in Sekunden Prototyp	Antwortzeit in Sekunden ImageFinder
4	2.13	3.19
8	4.76	6.98
10	5.95	7.07

Anmerkung. Eigene Darstellung.

Auswertung
Da im ImageFinder lediglich vier Google Bilder heruntergeladen werden, ist die Einbusse an Performanz durch die Google-API zu vernachlässigen.

6.5.3 Annoy Parameter n_trees

Der Erfinder von Annoy, Erik Bernhardsson, empfiehlt für den Parameter n_trees einen Wert von 100 [42]. Performanztests haben jedoch gezeigt, dass mit dem Wert für n_trees von 300 die Bildung der Baumstruktur nicht länger dauert. Zudem konnten, wie in
Tabelle 31 zu sehen, beim Vergleich der Ergebnisse mit dem Wert 100, 300 und dem Wert 1000 keine signifikanten Unterschiede festgestellt werden.

Tabelle 31: Annoy n_trees Berechnungen

Wert für n_trees	Zeit für Berechnung der Baumstruktur in Sekunden
1	0.081
5	0.33
50	3.74
100	11.282
200	12.52
300	8.275
500	16.54
1000	41.40

Anmerkung. Eigene Darstellung.

Auswertung
Der Parameter n_trees wurde aufgrund der ausgeführten Erkenntnisse auf 300 festgelegt.

6.5.4 Clustering – Startseite

In Tabelle 32 sind die mit den verschiedenen Verfahren gemessenen Zeiten und Iterationen aufgelistet. Dabei zeigt sich, dass bei Variante eins und drei die über zwei Millionen Abfragen der Distanz-Matrix stark ins Gewicht fallen.

Tabelle 32: Zeitberechnungen Clustering – Startseite

	Variante 1: Studienarbeit	**Variante 2: Wipen ähnlicher Bilder**	**Variante 3: Sortieren der Matrix**
Zeit für Clustering der Übersicht in Sekunden	338.49	48.78	112.16
Anzahl Iterationen	2‘389‘189	80	2‘389‘189
Anzahl Aufrufe von matrix.min()	2‘389‘189	80	0

Anmerkung. Eigene Darstellung.

Auswertung

Selbst wenn bei Variante drei die sortierte Gleichheits-Matrix auf der Datenbank gespeichert würde und so bei 11‘734 Bildern 20.64 Sekunden eingespart werden könnten, wäre Variante zwei immer noch schneller. Bei Variante eins und drei steigt die Berechnungsdauer bei zunehmenden Bildern in der Übersicht exponentiell an. Bei Variante zwei ist diese linear. Zudem fällt die kostspielige Methode matrix.min() nicht mehr ins Gewicht. Daher wurde diese Lösung für das Clustering der Übersichtsbilder verwendet.

6.6 Lasttest

Um die Leistungsfähigkeit des Backend-Servers messen zu können, wurde ein einfaches Python-Programm basierend auf dem Locust-Load-Testing-Tool [43] geschrieben. Als Vorbereitung wurde die gesamte Bildersammlung der Evangelisch-reformierten Kirche Horgen importiert. Der Test beschränkt sich auf HTTP GET Requests zu den drei Endpoints „/", „static/thumbnails/{image_id}" und „static/images/{image_id}". Dabei entspricht die {image_id} einem beliebigen Namen eines Bildes. Die Applikation läuft installiert als Windows Service, was dem Anwendungsfall der Evangelisch-reformierten Kirche Horgen entspricht.

```
class WebsiteTasks(TaskSet):

    def get_random_image(self):
        return random.choices(images)

    @task
    def index(self):
        self.client.get("/")

    @task
    def thumbnail(self):
        image_id = self.get_random_image()
        self.client.get(f"/static/thumbnails/{image_id}",
name="/static/thumbnail/[image_id]")

    @task
    def original(self):
        image_id = self.get_random_image()
        self.client.get(f"/static/images/{image_id}", name="/static/images/[image_id]")


class WebsiteUser(HttpLocust):
    task_set = WebsiteTasks
    min_wait = 5000
    max_wait = 15000
```

Abbildung 34: Ausschnitt aus dem Code für die Lasttest-Messungen

Anmerkung. Eigene Darstellung.

Wie Abbildung 34 zeigt, werden pro User mindestens 5000 Millisekunden gewartet, bis ein neuer Request ausgeführt wird. Dabei wird jeder Request, der länger als 15000 Millisekunden dauert, als „Fehlerhaft" gewertet. Die Anzahl gleichzeitiger User kann bequem über die Locust-Benutzeroberfläche simuliert, bzw. eingestellt werden. Dies entspricht der Spalte „Anzahl User" in Tabelle 33. Dabei wurde die „Hatch Rate", bzw. die Anzahl der User, die pro Sekunde neu hinzukommen, bis der Wert dem in der Spalte „Anzahl User" entspricht, auf den Default von 10 belassen. Ein User führt die Requests ungefähr gleich verteilt auf allen drei Endpoints durch. Tabelle 33 zeigt die Lasttest-Messungen mit 1, 10, 100, 1000 und 1000 gleichzeitigen Usern.

Tabelle 33: Server Lasttest

Anzahl User	Name	# Requests	# Fails	Median (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS
1	/	106	0	5	5	3.9	7	691	0.1
	/static/images/[image_id]	104	0	150	188	6	889.5	3727582	0
	/static/thumbnail/[image_id]	106	0	7	7	5	9	32839	0
	Total	**316**	**0**	**7**	**66**	**3.9**	**889.5**	**1238047**	**0.1**
10	/	570	0	5	6	3.9	684.6	691	0.2
	/static/images/[image_id]	542	0	170	208	7.0	1021.4	3789869	0.5
	/static/thumbnail/[image_id]	546	0	7	7	5.0	14.0	31982	0.3
	Total	**1658**	**0**	**7**	**73**	**3.9**	**1021.4**	**1249686**	**1**
100	/	2791	0	5	6	3.9	249.9	691	2.4
	/static/images/[image_id]	2784	0	210	268	6	2944.3	3510816	3
	/static/thumbnail/[image_id]	2862	0	8	9	4.6	38	31752	3.8
	Total	**8437**	**0**	**9**	**94**	**3.9**	**2944.3**	**1169481**	**9.2**
1000	/	12696	11970	1000	960	5	1922.1	40	11.6
	/static/images/[image_id]	12572	12130	1000	1152	19	27602.6	100239	11.9
	/static/thumbnail/[image_id]	12852	12122	1000	966	6	1969.3	1809	12.2
	Total	38120	36222	1000	1026	5	27602.6	33682	35.7
10000	/	245834	245834	2900	7614	1001.4	1708941.6	0	145.9
	/static /images/[image_id]	247116	247116	2900	7433	1001.4	1429480.4	0	147.5
	/static/thumbnail/[image_id]	245668	245668	2900	7467	1001.4	1258216.2	0	142.8
	Total	**738618**	**738618**	**2900**	**7505**	**1001.4**	**1708941.6**	**0**	**436.2**

Anmerkung. Eigene Darstellung.

Die Messwerte zeigen, dass der Server gut 9 bis 10 Requests pro Sekunde bei 100 gleichzeitig aktiven User bewältigen kann. Bei 1000 gleichzeitigen Usern und einer Request-Rate von knapp 36 Requests pro Sekunde kann der Server praktisch in keinem Fall mehr antworten.

6.7 Twelve-Factor App

Die Twelve-Factor Methode definiert ein gemeinsames Vokabular für die Entwicklung von modernen Applikationen und legt eine Reihe von Best Practices fest [34]. Nachfolgend soll in Tabelle 34 mit der genannten Methode untersucht werden, ob die zwölf Kriterien erfüllt sind.

Tabelle 34: Tests mittels der Twelve-Factor App Methode

#	Factor[44]	Erfüllt	Kommentar
1	„**Codebase**: One codebase tracked in revision control, many deploys”	✓	GitHub[45] dient als einziges Repository.
2	„**Dependencies**: Explicitly declare and isolate dependencies”	✓	Frontend: npm[46] dient als Package Manager und isoliert Dependencies in einem package-lock.json File. Backend: Pipenv[47] wird für die Deklaration verwendet und die Dependencies werden in einer Virtualenv isoliert.
3	„**Config**: Store config in the environment”	✓	Frontend kommt ohne Config aus Backend: Mit Dynaconf[48] wird sichergestellt, dass die Trennung von Code und Config besteht. Ein Konfigurationsfile kennt die drei Environments „default”, „testing” und „production”. Passwörter sind seperat in einem File Namens „secrets.toml” abgelegt. Dieses wird nicht in das Repository gespeichert.
4	„**Backing Services**: Treat backing services as attached resources”	✓	Die Applikation verfügt nicht über „echte” Backing Services, da die SQLite Datenbank nicht über ein Netzwerk kommuniziert. Dennoch kann die Datenbank ohne Code-Anpassungen ausgetauscht werden. Es muss lediglich der Connection-String im Config-File geändert werden. Die Applikation ist somit lose gekoppelt von der Datenbank.
5	„**Build, release, run**: Strictly separate build and run stages”	✗	Durch Azure DevOps sind Build und Release voneinander getrennt. Eine Run-Stage müsste zusätzlich hinzugefügt werden.
6	„**Processes**: Execute the app as one or more stateless processes”	✗	Die gewählte Datenbank SQLite ist nicht vom Server entkoppelt. Diese müsste als „Backing Service”[49] entkoppelt werden. Ebenfalls sind die Bilder, die für die textbasierte Suche von Google geladen werden, temporär auf der Festplatte persistiert. Ein künftiger Request geht davon aus, dass diese Bilder vorhanden sind. Dies wurde für den Anwendungszweck der Evangelisch-reformierten Kirche Horgen (Single User) so gewählt.

7	„**Port binding**: Export services via port binding”	✓	Der Webserver „ Uvicorn ” wurde als Library hinzugefügt. Somit ist die Applikation „Self-Contained” und benötigt keinen Webserver Container. Die Schnittstellen zum Backend werden über HTTP angeboten, indem auf Anfragen auf einen Port gehört wird.
8	„**Concurrency**: Scale out via the process model”	✓	Web-Requests werden von Worker-Prozessen verarbeitet, wodurch eine horizontale Skalierung leicht erreicht werden kann. Die Anzahl Worker können als Parameter angegeben werden: `uvicorn main:app --workers 2` Hinweis: Die Ausführung von Uvicorn mit mehreren Prozessen führt unter Windows aktuell noch zu Problemen.[50]
9	„**Disposability**: Maximize robustness with fast startup and graceful shutdown”	✗	Zwar kann das Backend jederzeit in wenigen Sekunden neu gestartet werden, allerdings sollte dies nicht geschehen, wenn gerade ein API Call nach „/api/clustering” ausgeführt wurde. Dies ist eine lang dauernde Operation, die die Cluster über alle Bilder berechnet und nur aufgerufen wird, nachdem neue Bilder hochgeladen wurden. Um den Cluster-Job zu speichern, müsste eine Distributed Task Queue wie Celery[51] eingesetzt werden. Nach dem Neustart der Applikation können so nicht abgeschlossene Tasks von der Queue geladen und neu ausgeführt werden.
10	„**Dev/Prod parity**: Keep development, staging, and production as similar as possible”	✓	Das Deployment wird ebenfalls von den Autoren durchgeführt. Die „Testing”-Umgebung ist beinahe identisch mit der „Production”-Umgebung.
11	„**Logs**: Treat logs as event streams”	✗	Alle Log-Nachrichten sollten lediglich als Event-Stream erfasst werden, der direkt nach stdout geschrieben wird. Da die Applikation letztendlich als Windows Service deployed wird und so keine Möglichkeit besteht, die Log-Nachrichten abzufangen, schreibt die Applikation zusätzlich in ein Log-File.

12	„**Admin processes**: Run admin/management tasks as one-off processes”	✓	Beim Start der Applikation werden Ordner für die flache Speicherung von Daten angelegt und (falls nicht vorhanden) die Datenbank erstellt.
			
			`if __name__ == "__main__":`
			`  backend_pre_start()`
			`  create_database()`

Anmerkung. Eigene Darstellung.

6.8 Software-Metriken

Nachfolgend werden Software-Metriken anhand des Commits mit der Id 4ec26fbb249ce34980b2cfcfad781807f0d02a83 ausgewertet. Eine Übersicht stellt Abbildung 35 dar.

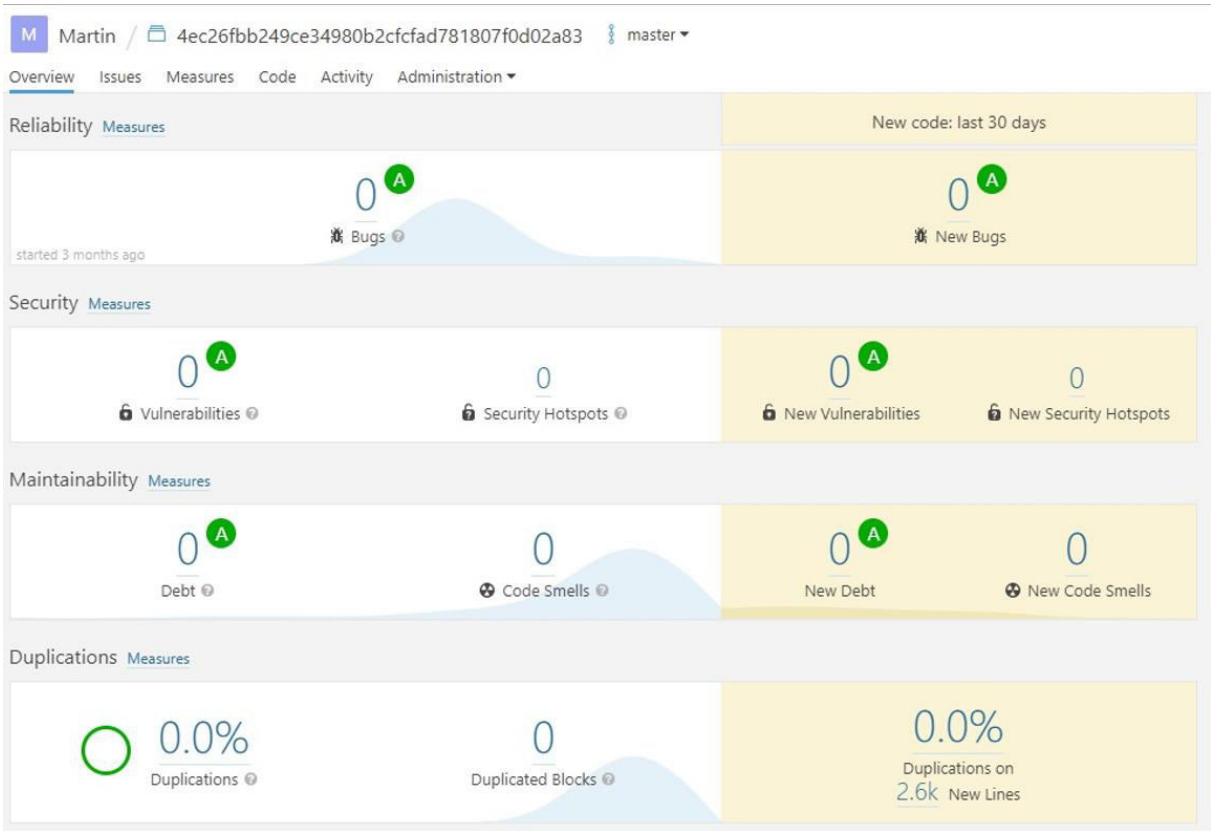


Abbildung 35: SonarCloud - Übersicht

6.9 Nicht-funktionale Anforderungen

Nachfolgend soll geprüft werden, ob die nicht-funktionalen Anforderungen aus Tabelle 35, Seite 4 eingehalten wurden.

Tabelle 35: Auswertung NFR's

#	Titel	Erfüllt	Anmerkung
NFR1	Global Exception-Handler	✓	Ist in backend/app/main.py zu finden.
NFR2	OWASP TOP 10	✓	Wurde durch SonarCloud geprüft, siehe dazu Anhang E.
NFR3	UI Design	✓	Das Vuetify Material Design Component Framework
NFR4	Benutzerführung	✓	Hierfür wurden Usability-Tests durchgeführt. Diesbezüglich wird auf Usability-TestsAnhang J verwiesen.
NFR5	Response-Time „/”	✓	Dies konnte in Kapitel 6.5 Performanztests gezeigt werden.
NFR6	Nearest Neighbor Search	✓	Dies wurde wie in Tabelle 29, Seite 65 beschrieben, in 1.69 Sekunden erreicht.
NFR7	Bildimport	✗	Dies wurde einmalig durchgeführt und erfüllt. Jedoch liegen nicht genaue Testmessungen vor.
NFR8	Win 10	✓	Die Software konnte auf einer Windows 10 Arbeitsstation der Evangelisch-reformierten Kirche Horgen installiert werden.
NFR9	Sprache	✓	Die Benutzerführung erfolgt in deutscher Sprache. Siehe dazu Anhang N.
NFR10	Log-Level	✓	Das Log-Level kann in logging.yaml angepasst werden.
NFR11	Datenschutz	✓	Die Bilder wurden zu keinem Zeitpunkt weitergegeben.
NFR12	Persistenz	✓	Als relationale Datenbank wurde SQLite eingesetzt.
NFR13	Webapplikation	✓	Die dazu notwendige Architektur ist in Kapitel 4.2.2 zu finden.
NFR14	Temporäre Daten	✓	Dies wurde in Kapitel 0 beschrieben.
NFR15	Inception-v3	✓	Wurde verwendet, um die Feature-Vektoren zu erstellen.
NFR16	Baumstruktur	✓	Die Annoy-Library wurde implementiert, siehe dazu **Fehler! Verweisquelle konnte nicht gefunden werden.** 4.6
NFR17	Best lowest search rank first	✓	Der genannte Algorithmus wurde implementiert, siehe dazu Kapitel 4.5.2.
NFR18	Google-Schnittstelle	✓	Die offizielle Google Custom Search API wurde verwendet. Der Code-Abschnitt ist in backend/app/services/google_services.py zu finden.
NFR19	Code Formatting	✓	Entsprechende Massnahmen wurden durchgeführt und eingehalten, siehe Anhang E.
NFR20	Testcoverage (Frontend)	✗	Es wurde lediglich eine Testabdeckung von 24.45% erreicht (siehe Abbildung 56).
NFR21	Testcoverage (Backend)	✓	Das Backend weist eine Testabdeckung von 88.49% auf (siehe Abbildung 56).
NFR22	Agile Vorgehensweise	✗	RUP wurde als Projektmanagement-Methode gewählt und zählt lediglich als „semi-agil”. Dies aus den in Kapitel 3 - Projektmanagement erwähnten Gründen.

Anmerkung. Eigene Darstellung.

7 Ergebnisdiskussion

Die erstellte Software konnte bei der Evangelisch-reformierten Kirche Horgen installiert werden. Eine erste Rückmeldung der Mitarbeitenden bestätigt, dass die Software den Erwartungen entspricht und eine Optimierung für das Durchsuchen der Bildersammlung darstellt. Dies zeigt ein Auszug aus der von Tiana Limberger am 07. Juni 2019 per E-Mail erhaltenen Rückmeldung:

„Vorab ein grosses Dankeschön an euch! Das neue Programm ImageFinder, dass uns eine optimierte Suche in unserer grossen Bilddatenbank ermöglicht, ist eine gelungene, intuitive und anwenderfreundliche Hilfe. Die Möglichkeit der Suche nach verschiedenen Kriterien wie Bildsujet, Autor, Datum und Anlass ist nahezu ein Garant, das optimale Bild zu finden. Gerade für die Öffentlichkeitsarbeit ist es hilfreich, wenn nicht mehr in mühseliger Detektivarbeit nach dem Speicherort der jeweiligen Anlässe gesucht werden muss. Die Ergänzung über die Suche von Google ist innovativ und lässt auch hier eine Vielzahl an Möglichkeiten zu.“

7.1 Funktionale Anforderungen

Die geplanten funktionalen Anforderungen konnten umgesetzt werden. Use Cases 10, 11 und 12 wurden bereits zu Beginn zurückgestellt. Es zeigte sich, dass die Zeit nicht mehr reichte, um diese Funktionalitäten zu implementieren. Nachfolgend werden die wichtigsten Anforderungen kurz beschrieben.

Bildimport
Der Bildimport konnte gegenüber dem Prototypen verbessert werden. Die Benutzerführung ist klar und einfach gehalten. Da bereits beim Hochladen die Metadaten erfasst werden können, wird der Aufwand für die Benutzerinnen und Benutzer reduziert.

Erfassen der Metadaten
Das Erfassen von Metadaten ermöglicht ein grösseres Spektrum an Varianten, um sich in der Bildersammlung zurechtzufinden. Es ist mit der Filterfunktion möglich, gezielt einzelne Bilder zu finden und mit diesen weiterzuarbeiten. Werden neu hochgeladene Bilder konsequent mit Tags und Fotograf gekennzeichnet, entfällt das mühsame Suchen im Windows Explorer.

Clustering
Das Clustering bietet, je nachdem wie viele und wie ähnliche Bilder sich in der Bildersammlung befinden, eine optimierte Darstellung der Ergebnisse. Je mehr Bilder in die Software geladen werden, desto mehr gewinnt das Clustering an Relevanz. Eine abschliessende Beurteilung durch die Mitarbeitenden der Evangelisch-reformierten Kirche Horgen, ob sich das Clustering für die Bildersuche eignet, war aufgrund des Zeitmangels nicht möglich. Dies wird sich in den nächsten Monaten zeigen.

7.2 Nicht-funktionale Anforderungen

Nachfolgend werden die Ergebnisse der wichtigsten Aspekte der nicht-funktionalen Anforderungen beschrieben.

Performanz
Die Steigerung der Performanz war ein wichtiger Faktor und wurde in verschiedenen nicht-funktionalen Anforderungen thematisiert, da eine Software, welche mit hohen Latenzen das Benutzererlebnis mindert, nicht verwendet wird. Diverse Verbesserungen wie Pagination, das

Speichern der Distanzen in der Baumstruktur oder das Wipen beim Identifizieren der Bilder für die Übersicht, konnten die Response Time markant verbessern.

Suchergebnisse
Die Suchergebnisse mit einem Referenzbild waren bereits im Prototypen zufriedenstellend. Diese Qualität konnte mit der Baumstruktur von Annoy beibehalten werden. Für die Suche mit mehreren Referenzbildern konnte bei den durchgeführten Tests mit dem „Best-lowest-search-rank-first“ Algorithmus eine Verbesserung der Ergebnisse erreicht werden. Jedoch waren nicht alle Tests zufriedenstellend. Teilweise waren die Ergebnisse mit dem arithmetischen Mittelwert besser.

Software-Metriken
Mit Hilfe von SonarCloud und DeepScan konnten einige Metriken automatisiert und kontinuierlich überprüft werden. Dabei wurde letztendlich in vielen Kategorien (Reliability, Security, Maintainability, Duplications) das höchste Level erreicht. Die Testabdeckung ist mit einem Gesamtwert von 71.6% zufriedenstellend.

7.3 Projektmanagement

Das semi-agile Vorgehen nach RUP brachte einen Mehrwert, da einerseits eine Elaboration-Phase für die Planung eingeführt werden konnte und andererseits in der Construction-Phase individuell nach Rücksprache des Kunden die Prioritäten neu verteilt werden konnten. Dabei lag der Fokus stets auf dem erfolgreichen Projektabschluss. Nachfolgend werden die wichtigsten Punkte des Projektmanagements ausgewertet.

Meilensteine
Die Festlegung von Meilensteinen erlaubte eine grobe Überwachung des Projektes und half den Autoren den Zeitplan, trotz agilem Vorgehen in der Construction-Phase, einzuhalten. Der Feature-Freeze wurde um eine Woche verschoben. Aufgrund der eingeplanten Reservezeit hatte dies jedoch keinen Einfluss auf das Projekt.

Risiken
Das zu Projektbeginn definierte Risiko R2, „Compatibility“, trat mehrfach während des Projektes ein. Trotz der getätigten Abklärungen zu den verwendeten Frameworks und Libraries kam es immer wieder zu Kompatibilitätsproblemen. Vor allem neue Updates der Frameworks kosteten Zeit, da diese teilweise Anpassungen an der Applikation zur Folge hatten. Auch mussten Technologien komplett ausgetauscht werden, da diese auf Windows Systemen teilweise fehlerhaftes Verhalten zeigten. Die restlichen Risiken konnten durch die Projektplanung abgefangen werden.

7.4 Architektur

Der Entscheid für eine Client-Server-Architektur in Kombination mit einer SQLite Datenbank gab den Autoren die benötigte Flexibilität. Die entstandene Software ist für die Zukunft gerüstet, da die Erweiterbarkeit gewährleistet ist. Es besteht die Möglichkeit die Software nicht nur lokal zu betreiben, sondern die Verteilung mehrerer Clients zu erlauben.

8 Zusammenfassung und Ausblick

Das Ziel dieser Bachelorarbeit war es, ausgehend von einem Prototypen und den gewonnen Erkenntnissen aus der Studienarbeit, einen marktreifen Google-Bildersuche Klon zu entwickeln.

Zu Beginn der Bachelorarbeit wurden nebst funktionalen Anforderungen auch Qualitätsziele festgelegt und adressiert. Ein semi-agiles Vorgehen nach RUP in Kombination mit festgelegten Meilensteinen half die Bachelorarbeit zu steuern und zu überwachen. Durch eine Aufwandschätzung der funktionalen Anforderungen nach Worst, Normal und Best Case konnte ungefähr ein Gefühl für den Scope entwickelt werden. Bevor mit dem Lösungskonzept begonnen wurde, haben sich die Autoren mögliche Risiken überlegt und diese mittels einer Risikomatrix illustriert. So konnten die Risiken besser eingeschätzt und geeignete Massnahmen getroffen werden, um die Risiken zu minimieren. In einem Lösungskonzept wurden für das Client-Server-Modell - unter Berücksichtigung der vorhandenen Infrastruktur bei der Evangelisch-reformierten Kirche Horgen - geeignete Frameworks evaluiert. Unglücklicherweise trat in einem späteren Verlauf der Bachelorarbeit trotzdem das Risiko ein, dass das gewählte Backendframework nicht mehr von Windows unterstützt wurde, wodurch ein rascher Technologiewechsel vorgenommen werden musste. Durch simple, in Excel klickbare Wireframes, konnte von der Evangelisch-reformierten Kirche Horgen ein erstes Feedback eingeholt werden. Dies half ebenfalls bei der Überarbeitung der Aufwandschätzung der funktionalen Anforderungen, wodurch bewusst drei Use Cases weiter bearbeitet wurden. Parallel dazu wurde bereits vor der Umsetzung überlegt, wie die nicht funktionalen Anforderungen umgesetzt werden können. Somit konnte der im Projektplan festgelegte Meilenstein „End of Elaboration” mit den Bestandteilen: Anforderungen verstanden, Entwürfe des User Interfaces vorhanden, Software Architektur-Durchstich erlangt, Entwicklungs-Werkzeuge aufgesetzt, Aufwandschätzung durchgeführt, erreicht werden. Bei der Umsetzung der Architektur der Software wurde darauf Wert gelegt, dass keine zirkulären Abhängigkeiten bestehen und eine hohe Kohäsion der Klassen sowie der Module erreicht wird. Um einem User eine grössere Vielfalt auf der Startseite der Weboberfläche zu bieten und damit verbunden die Bilder zu clustern, wurde ein geeigneter Schwellwert ermittelt. Mit diesem Schwellwert sind aus der Bildersammlung der Evangelisch-reformierten Kirche Horgen viele Cluster mit der Grösse eins entstanden. Daraus wurde die Erkenntnis gewonnen, dass das Clustering bei grossen Datenmengen geeignet ist. In der Umsetzungsphase konnte eine erhebliche Beschleunigung der Nearest Neighbor Suche, die zu einem Referenzbild ähnliche Bilder sucht und bereits aus der Studienarbeit bekannt war, erreicht werden. Ebenfalls konnte die Qualität der Suchresultate bei einer Suche ausgehend von mehreren Referenzbildern durch die Implementierung des „Best-lowest-rank-first” Algorithmus leicht verbessert werden. Durch die automatisiert eingerichteten Qualitätskontrollen konnte die Qualität des Source Codes durchgängig hochgehalten werden. Nach einer durchgeführten Installation und Schulung bei der Evangelisch-reformierten Kirche Horgen wurde die Software erfolgreich abgenommen.

Das Feedback der Evangelisch-reformierten Kirche Horgen war sehr erfreulich und zufriedenstellend. Die entwickelte Software bietet eine gleichwertige Alternative zu den Diensten der gängigen Cloud-Anbieter, ohne gegen die Datenschutzrichtlinien zu verstossen. Weiterführende Features, wie zum Beispiel die Implementierung einer Gesichtserkennung, wären durchaus denkbar.

9 Abkürzungsverzeichnis

ASGI – Asynchronous Server Gateway Interface
DDOS – Distributed-Denial-of-Service
IDE – Integrated Development Environment
JSON – JavaScript Object Notation
NNS – Nearest Neighbor Search
PyQt5 – Python Cross-Platform GUI Toolkit
ROC – Receiver Operator Characteristic Curve
RUP – Rational Unified Process
SPA – Single Page Application
URI – Unified Resource Identifier
WSGI – Web Server Gateway Interface

10 Literaturverzeichnis

[1] Martin Odermatt, TS (2018): Bildklassifikation mit Hilfe eines neuralen Netzes, 2018.

[2] Melekhov, I, Kannala, J, Rahtu: Siamese network features for image matching. 2016 23rd International Conference on Pattern Recognition (ICPR). IEEE. https://www.semanticscholar.org/paper/Siamese-network-features-for-image-matching-Melekhov-Kannala/5df4b92f86698a8270f9fba73b87c9b97e0d1016.

[3] hardikvasa/google-images-download. https://github.com/hardikvasa/google-images-download. Abgerufen am 11.06.2019.

[4] (16.05.2019): Custom Search | Google Developers. https://developers.google.com/custom-search/. Abgerufen am 11.06.2019.

[5] (20.08.2016): PyQt5 Reference Guide - PyQt 5.7 Reference Guide. https://doc.bccnsoft.com/docs/PyQt5/#. Abgerufen am 14.06.2019.

[6] Larman, C (Hrsg) (2009): Applying UML and patterns. An introduction to object-oriented analysis and design and iterative development. 3. Auflage. Prentice Hall, Upper Saddle River, NJ.

[7] Hewlett-Packard (1989): Journal. https://www.hpl.hp.com/hpjournal/pdfs/IssuePDFs/1989-04.pdf.

[8] OWASP. https://www.owasp.org/index.php/Main_Page. Abgerufen am 10.06.2019.

[9] (28.05.2019): Design - Material Design. https://material.io/design/. Abgerufen am 10.06.2019.

[10] Szegedy, C, Vanhoucke, V, Ioffe, S, Shlens, J, Wojna, Z (2015): Rethinking the Inception Architecture for Computer Vision. 08866236.

[11] (25.01.2019): Custom Search JSON API | Custom Search | Google Developers. https://developers.google.com/custom-search/v1/overview. Abgerufen am 10.06.2019.

[12] (13.06.2019): What is Scrum? https://www.scrum.org/resources/what-is-scrum. Abgerufen am 14.06.2019.

[13] Electron | Build cross platform desktop apps with JavaScript, HTML, and CSS. https://electronjs.org/. Abgerufen am 13.06.2019.

[14] Prof. Olaf Zimmermann (2012): Making Architectural Knowledge Sustainable–Industrial Practice Report and Outlook. https://resources.sei.cmu.edu/asset_files/Presentation/2012_017_001_31349.pdf.

[15] (06.06.2019): Cookies - Sanic 19.6.0 documentation. https://sanic.readthedocs.io/en/latest/sanic/cookies.html. Abgerufen am 10.06.2019.

[16] (13.06.2019): Richardson Maturity Model. https://martinfowler.com/articles/richardsonMaturityModel.html. Abgerufen am 13.06.2019.

[17] (13.06.2019): bliki: PresentationDomainDataLayering. https://martinfowler.com/bliki/PresentationDomainDataLayering.html. Abgerufen am 13.06.2019.

[18] (13.06.2019): all_basic.png (PNG-Grafik, 540 × 384 Pixel). https://martinfowler.com/bliki/images/presentationDomainDataLayering/all_basic.png. Abgerufen am 13.06.2019.

[19] Wikipedia (27.05.2019): Hierarchische Clusteranalyse. https://de.wikipedia.org/w/index.php?oldid=184718283. Abgerufen am 10.06.2019.

[20] spotify/annoy. https://github.com/spotify/annoy. Abgerufen am 10.06.2019.

[21] (10.06.2019): mmap(2) - Linux manual page. http://man7.org/linux/man-pages/man2/mmap.2.html. Abgerufen am 10.06.2019.

[22] erikbern/ann-benchmarks. https://github.com/erikbern/ann-benchmarks. Abgerufen am 10.06.2019.

[23] Wikipedia (06.06.2019): Locality-sensitive hashing - Wikipedia. https://en.wikipedia.org/w/index.php?oldid=894277348. Abgerufen am 10.06.2019.

[24] Single File Components - Vue.js. https://vuejs.org/v2/guide/single-file-components.html. Abgerufen am 10.06.2019.

[25] Vue.js Material Component Framework - Vuetify.js. https://vuetifyjs.com/en/. Abgerufen am 10.06.2019.
[26] Introduction | Vue Router. https://router.vuejs.org/. Abgerufen am 13.06.2019.
[27] (09.06.2019): OpenAPI Generator · Generate clients, servers, and documentation from OpenAPI 2.0/3.x documents. https://openapi-generator.tech/. Abgerufen am 10.06.2019.
[28] bliki: PresentationDomainDataLayering. https://martinfowler.com/bliki/PresentationDomainDataLayering.html. Abgerufen am 10.06.2019.
[29] P of EAA: Data Transfer Object. https://martinfowler.com/eaaCatalog/dataTransferObject.html. Abgerufen am 10.06.2019.
[30] Add support for multi-file uploads by tiangolo · Pull Request #158 · tiangolo/fastapi. https://github.com/tiangolo/fastapi/pull/158. Abgerufen am 10.06.2019.
[31] (10.06.2019): Querying - peewee 3.9.6 documentation. http://docs.peewee-orm.com/en/latest/peewee/querying.html. Abgerufen am 10.06.2019.
[32] (24.05.2019): Implementation Limits For SQLite. https://www.sqlite.org/limits.html. Abgerufen am 10.06.2019.
[33] Using the Request Directly - FastAPI. https://fastapi.tiangolo.com/tutorial/using-request-directly/. Abgerufen am 10.06.2019.
[34] Wiggins, A: The Twelve-Factor App. https://12factor.net/config. Abgerufen am 10.06.2019.
[35] HTTP response status codes. https://developer.mozilla.org/en-US/docs/Web/HTTP/Status. Abgerufen am 13.06.2019.
[36] (12.06.2019): Flow: A Static Type Checker for JavaScript. https://flow.org/. Abgerufen am 13.06.2019.
[37] Code completion - Help | PyCharm. https://www.jetbrains.com/help/pycharm/auto-completing-code.html. Abgerufen am 13.06.2019.
[38] (13.06.2019): typing - Support for type hints - Python 3.7.3 documentation. https://docs.python.org/3/library/typing.html. Abgerufen am 13.06.2019.
[39] gewarren: Unit testing fundamentals - Visual Studio. https://docs.microsoft.com/en-us/visualstudio/test/unit-test-basics?view=vs-2019. Abgerufen am 14.06.2019.
[40] (13.06.2019): pytest fixtures: explicit, modular, scalable - pytest documentation. https://docs.pytest.org/en/latest/fixture.html. Abgerufen am 14.06.2019.
[41] Optimize your code using profilers - Help | PyCharm. https://www.jetbrains.com/help/pycharm/profiler.html. Abgerufen am 12.06.2019.
[42] Increasing accuracy of search · Issue #148 · spotify/annoy. https://github.com/spotify/annoy/issues/148. Abgerufen am 11.06.2019.
[43] Locust - A modern load testing framework. https://locust.io/. Abgerufen am 13.06.2019.
[44] Wiggins, A: The Twelve-Factor App. https://12factor.net/. Abgerufen am 12.06.2019.
[45] Build software better, together. https://github.com/. Abgerufen am 10.06.2019.
[46] npm | get npm. https://www.npmjs.com/get-npm. Abgerufen am 12.06.2019.
[47] pypa/pipenv. https://github.com/pypa/pipenv. Abgerufen am 12.06.2019.
[48] rochacbruno/dynaconf. https://github.com/rochacbruno/dynaconf. Abgerufen am 12.06.2019.
[49] Wiggins, A: The Twelve-Factor App. https://12factor.net/backing-services. Abgerufen am 12.06.2019.
[50] Run uvicorn with multiple workers on Windows - WinError 87 · Issue #342 · encode/uvicorn. https://github.com/encode/uvicorn/issues/342. Abgerufen am 12.06.2019.
[51] Homepage | Celery: Distributed Task Queue. http://www.celeryproject.org/. Abgerufen am 12.06.2019.
[52] MetricsReloaded - Plugins | JetBrains. https://plugins.jetbrains.com/plugin/93-metricsreloaded. Abgerufen am 13.06.2019.
[53] Vue CLI. https://cli.vuejs.org/. Abgerufen am 12.06.2019.

[54] (21.02.2019): PyInstaller Quickstart - PyInstaller bundles Python applications. https://www.pyinstaller.org/. Abgerufen am 12.06.2019.
[55] (20.03.2019): Security Reports | SonarQube Docs. https://docs.sonarqube.org/latest/user-guide/security-reports/. Abgerufen am 12.06.2019.
[56] Impacts and Grades | DeepScan. https://deepscan.io/docs/guides/get-started/grades. Abgerufen am 13.06.2019.
[57] (18.03.2019): Code Coverage Done Right | Codecov. https://codecov.io/. Abgerufen am 10.06.2019.
[58] Known issue: Azure Pipelines Linux images missing sqlite3 for Python - Developer Community. https://developercommunity.visualstudio.com/content/problem/598264/known-issue-azure-pipelines-images-missing-sqlite3.html. Abgerufen am 12.06.2019.
[59] Graphs. https://docs.codecov.io/docs/graphs. Abgerufen am 12.06.2019.
[60] Azure DevOps Services | Microsoft Azure. https://azure.microsoft.com/en-us/services/devops/. Abgerufen am 10.06.2019.
[61] eslint/eslint. https://github.com/eslint/eslint. Abgerufen am 10.06.2019.
[62] airbnb/javascript. https://github.com/airbnb/javascript. Abgerufen am 10.06.2019.
[63] (02.01.2019): Available rules | eslint-plugin-vue. https://vuejs.github.io/eslint-plugin-vue/rules/. Abgerufen am 12.06.2019.
[64] python/black. https://github.com/python/black. Abgerufen am 10.06.2019.
[65] PEP 8 - Style Guide for Python Code. https://www.python.org/dev/peps/pep-0008/. Abgerufen am 10.06.2019.
[66] SonarCloud | Clean Code, Rockstar Status. https://sonarcloud.io/about. Abgerufen am 10.06.2019.
[67] How to ensure JavaScript code quality | DeepScan. https://deepscan.io/. Abgerufen am 10.06.2019.
[68] Why Us - Semantic analysis for your JavaScript | DeepScan. https://deepscan.io/why-us/. Abgerufen am 10.06.2019.
[69] Rules | DeepScan. https://deepscan.io/docs/rules/#vue. Abgerufen am 10.06.2019.
[70] PyCharm: the Python IDE for Professional Developers by JetBrains. https://www.jetbrains.com/pycharm/. Abgerufen am 13.06.2019.
[71] (10.06.2019): Hitchhiker's Guide to Software Architecture and Everything Else - by Michael Stal: Architect's Toolset - CRC Cards. http://stal.blogspot.com/2006/12/architects-toolset-crc-cards.html. Abgerufen am 10.06.2019.
[72] Christie, T (04.06.2019): Home - Django REST framework. https://www.django-rest-framework.org/#installation. Abgerufen am 10.06.2019.
[73] (30.05.2019): Full Stack TurboGears: Wiki in 20 Minutes - TurboGears 2.4.0 documentation. https://turbogears.readthedocs.io/en/latest/turbogears/wiki20.html. Abgerufen am 10.06.2019.
[74] web2py. http://www.web2py.com/. Abgerufen am 10.06.2019.
[75] web2py/web2py. https://github.com/web2py/web2py/graphs/contributors. Abgerufen am 10.06.2019.
[76] web2py - Services. http://web2py.com/books/default/chapter/29/10/services#parse_as_rest-experimental-. Abgerufen am 10.06.2019.
[77] (05.05.2019): CherryPy - A Minimalist Python Web Framework - CherryPy 18.1.2.dev4+g0f8523cd.d20190505 documentation. https://docs.cherrypy.org/en/latest/. Abgerufen am 10.06.2019.
[78] pallets/flask. https://github.com/pallets/flask. Abgerufen am 10.06.2019.
[79] (12.03.2019): Flask. https://www.palletsprojects.com/p/flask/. Abgerufen am 10.06.2019.
[80] (25.05.2018): Patterns for Flask - Flask 1.0.2 documentation. http://flask.pocoo.org/docs/1.0/patterns/#patterns. Abgerufen am 10.06.2019.
[81] com, d: CherryPy vs Sanic: Which Python API Framework is Faster? https://dataweave.com/blog/cherrypy-vs-sanic-which-python-api-framework-is-faster-103fe732adc6. Abgerufen am 10.06.2019.

[82] Tornado multiple sub-processes on windows · Issue #2622 · tornadoweb/tornado. https://github.com/tornadoweb/tornado/issues/2622. Abgerufen am 10.06.2019.
[83] (03.11.2018): Pyramid Tutorials - The Pyramid Tutorials v0.1. https://docs.pylonsproject.org/projects/pyramid-tutorials/en/latest/. Abgerufen am 10.06.2019.
[84] encode/uvicorn. https://github.com/encode/uvicorn. Abgerufen am 10.06.2019.
[85] kennethreitz/responder. https://github.com/kennethreitz/responder. Abgerufen am 10.06.2019.
[86] encode/starlette. https://github.com/encode/starlette. Abgerufen am 10.06.2019.
[87] Hattingh, C (2016): 20 Python libraries you aren't using (but should). O'Reilly Media, Sebastopiol, CA.
[88] encode/starlette. https://github.com/encode/starlette. Abgerufen am 10.06.2019.
[89] CubicWeb Semantic Web Framework. https://www.cubicweb.org/. Abgerufen am 10.06.2019.
[90] (08.02.2019): 1. Installation of a CubicWeb environment - CubicWeb 3.26.7. https://cubicweb.readthedocs.io/en/latest/book/admin/setup/#windowsinstallation. Abgerufen am 10.06.2019.
[91] undefined. undefined. Abgerufen am 10.06.2019.
[92] Visual Studio IntelliCode | Visual Studio. https://visualstudio.microsoft.com/services/intellicode/. Abgerufen am 10.06.2019.
[93] vibora-io/vibora. https://github.com/vibora-io/vibora. Abgerufen am 10.06.2019.
[94] vibora-io/vibora. https://github.com/vibora-io/vibora/blob/master/README.md. Abgerufen am 10.06.2019.
[95] Wikipedia (07.06.2019): Microframework - Wikipedia. https://en.wikipedia.org/w/index.php?oldid=899349376. Abgerufen am 10.06.2019.
[96] TurboGears/tg2. https://github.com/TurboGears/tg2. Abgerufen am 10.06.2019.
[97] Group, D (02.04.2019): Welcome! - The Apache HTTP Server Project. https://httpd.apache.org/. Abgerufen am 10.06.2019.
[98] NGINX | High Performance Load Balancer, Web Server, & Reverse Proxy. https://www.nginx.com/. Abgerufen am 10.06.2019.
[99] How to deploy with WSGI | Django documentation | Django. https://docs.djangoproject.com/en/2.1/howto/deployment/wsgi/. Abgerufen am 10.06.2019.
[100] (15.03.2019): uWSGI 1.9 - uWSGI 2.0 documentation. https://uwsgi-docs.readthedocs.io/en/latest/Changelog-1.9.html. Abgerufen am 10.06.2019.
[101] (06.06.2019): Deployment - The Pyramid Community Cookbook v0.2. https://docs.pylonsproject.org/projects/pyramid-cookbook/en/latest/deployment/index.html. Abgerufen am 10.06.2019.
[102] (06.06.2019): ASGI (Asynchronous Server Gateway Interface) - The Pyramid Community Cookbook v0.2. https://docs.pylonsproject.org/projects/pyramid-cookbook/en/latest/deployment/asgi.html. Abgerufen am 10.06.2019.
[103] (06.06.2019): Windows - The Pyramid Community Cookbook v0.2. https://docs.pylonsproject.org/projects/pyramid-cookbook/en/latest/deployment/windows.html. Abgerufen am 10.06.2019.
[104] (30.05.2019): Deploying TurboGears - TurboGears 2.4.0 documentation. https://turbogears.readthedocs.io/en/latest/cookbook/deploy/. Abgerufen am 10.06.2019.
[105] circus-tent/chaussette. https://github.com/circus-tent/chaussette. Abgerufen am 10.06.2019.
[106] web2py - Deployment recipes. http://www.web2py.com/books/default/chapter/29/13/deployment-recipes#Windows. Abgerufen am 10.06.2019.
[107] kirillkovalenko/nssm. https://github.com/kirillkovalenko/nssm. Abgerufen am 10.06.2019.
[108] (25.05.2018): Deployment Options - Flask 1.0.2 documentation. http://flask.pocoo.org/docs/1.0/deploying/. Abgerufen am 10.06.2019.
[109] (25.05.2018): Standalone WSGI Containers - Flask 1.0.2 documentation. http://flask.pocoo.org/docs/1.0/deploying/wsgi-standalone/#twisted-web. Abgerufen am 10.06.2019.

[110] (25.05.2018): Standalone WSGI Containers - Flask 1.0.2 documentation. http://flask.pocoo.org/docs/1.0/deploying/wsgi-standalone/#gevent. Abgerufen am 10.06.2019.
[111] (25.05.2018): Deploy to Production - Flask 1.0.2 documentation. http://flask.pocoo.org/docs/1.0/tutorial/deploy/. Abgerufen am 10.06.2019.
[112] (10.06.2019): Deployment - Bottle 0.13-dev documentation. https://bottlepy.org/docs/dev/deployment.html. Abgerufen am 10.06.2019.
[113] (05.05.2019): Deploy - CherryPy 18.1.2.dev4+g0f8523cd.d20190505 documentation. http://docs.cherrypy.org/en/latest/deploy.html. Abgerufen am 10.06.2019.
[114] (05.05.2019): Foreword - CherryPy 18.1.2.dev4+g0f8523cd.d20190505 documentation. http://docs.cherrypy.org/en/latest/intro.html. Abgerufen am 10.06.2019.
[115] huge-success/sanic. https://github.com/huge-success/sanic. Abgerufen am 10.06.2019.
[116] (06.06.2019): Deploying - Sanic 19.6.0 documentation. https://sanic.readthedocs.io/en/latest/sanic/deploying.html. Abgerufen am 10.06.2019.
[117] (23.03.2019): Tornado Web Server - Tornado 6.0.2 documentation. https://www.tornadoweb.org/en/stable/. Abgerufen am 10.06.2019.
[118] https://www.facebook.com/WordPresscom (2017): Evolution of Push Technologies : From Regular HTTP to Long Polling to WebSocket. https://efficientcodeblog.wordpress.com/2017/12/14/evolution-of-push-technologies-from-regular-http-to-websocket/. Abgerufen am 10.06.2019.
[119] (29.03.2019): A familiar HTTP Service Framework - responder 1.3.0 documentation. https://python-responder.org/en/latest/index.html. Abgerufen am 10.06.2019.
[120] (03.05.2019): hug: Embrace the APIs of the future. http://www.hug.rest/website/quickstart. Abgerufen am 10.06.2019.
[121] (26.04.2019): Installation - Falcon 2.0.0 documentation. https://falcon.readthedocs.io/en/stable/user/install.html. Abgerufen am 10.06.2019.
[122] Deployment - FastAPI. https://fastapi.tiangolo.com/deployment/. Abgerufen am 10.06.2019.
[123] tiangolo/full-stack-fastapi-postgresql. https://github.com/tiangolo/full-stack-fastapi-postgresql. Abgerufen am 10.06.2019.
[124] tiangolo/full-stack-fastapi-couchbase. https://github.com/tiangolo/full-stack-fastapi-couchbase. Abgerufen am 10.06.2019.
[125] (26.03.2019): Uvicorn. https://www.uvicorn.org/. Abgerufen am 10.06.2019.
[126] (09.05.2019): Hypercorn documentation - Hypercorn 0.6.0 documentation. https://pgjones.gitlab.io/hypercorn/. Abgerufen am 10.06.2019.
[127] Django Software Foundation (2019): Django Documentation, 2019.
[128] The TurboGears Doc Team (2015): turbogears. https://media.readthedocs.org/pdf/turbogears/tg2.3.6/turbogears.pdf.
[129] Massimo Di Pierro (2013): web2py8.5plus3minus4plus24plus2minus2Complete Reference Manual, 5th Edition. https://mdipierro.github.io/web2py/web2py_manual_5th.pdf.
[130] Hellkamp, M (2019): Bottle Documentation, 2019.
[131] Team, C (2019): CherryPy Documentation. https://buildmedia.readthedocs.org/media/pdf/cherrypy/latest/cherrypy.pdf.
[132] (2018): pallet, 2018.
[133] contributors, S (2019): Sanic Documentation. https://media.readthedocs.org/pdf/sanic/latest/sanic.pdf.
[134] The Tornado Authors (2019): Tornado Documentation. https://media.readthedocs.org/pdf/tornado/latest/tornado.pdf.
[135] McDonough, C (2019): The Pyramid Web Framework. http://media.readthedocs.org/pdf/pyramid/1.9-branch/pyramid.pdf.
[136] Reitz, K (2019): responder Documentation. https://media.readthedocs.org/pdf/responder/latest/responder.pdf.

[137] Kurt Griffiths et al. (2019): Falcon Documentation. https://media.readthedocs.org/pdf/falcon/latest/falcon.pdf.
[138] FastAPI. https://fastapi.tiangolo.com/. Abgerufen am 10.06.2019.
[139] WinError 10022 - multiple workers · Issue #1545 · huge-success/sanic. https://github.com/huge-success/sanic/issues/1545. Abgerufen am 10.06.2019.
[140] Error with Multiple workers on Windows · Issue #1517 · huge-success/sanic. https://github.com/huge-success/sanic/issues/1517#issuecomment-478083478. Abgerufen am 10.06.2019.
[141] ASGI refactoring attempt by tomchristie · Pull Request #1475 · huge-success/sanic. https://github.com/huge-success/sanic/pull/1475. Abgerufen am 10.06.2019.
[142] dzone: reactjs-vs-vuejs-comparison-of-popular-frameworks. https://dzone.com/articles/reactjs-vs-vuejs-comparison-of-popular-frameworks. Abgerufen am 10.06.2019.
[143] NSSM - the Non-Sucking Service Manager. https://nssm.cc/. Abgerufen am 12.06.2019.

Anhang A Aufgabenstellung

Bildklassifikation mit Hilfe eines neuronalen Netzes

Bachelorarbeit von: Martin Odermatt und Tobias Saladin

Industriepartner: Reformierte Kirche Horgen

Planung

18.02.2019	Beginn der Bachelorarbeit, Ausgabe der Aufgabenstellung durch den Betreuer.
07.06.2019	Die Studierenden geben den Abstract für die Diplomarbeitsbroschüre zur Kontrolle an ihren Betreuer/Examinator frei. Die Studierenden erhalten vorgängig vom Studiengangsekretariat die Aufforderung mit den Zugangsdaten zur Online-Erfassung des Abstracts im DAB-Tool. Die Studierenden senden per Email das A0-Poster zur Prüfung an ihren Examinator/Betreuer. Vorlagen sowie eine ausführliche Anleitung betreffend Dokumentation stehen auf dem Skripteserver zur Verfügung.
12.06.2019	Der Betreuer/Examinator gibt das Dokument mit dem korrekten und vollständigen Abstract der Broschüre zur Weiterverarbeitung an das Studiengangsekretariat frei. Für die Ausstellung der Bachelorarbeiten das A0 Posters per Email bis 10.00 Uhr an das Studiengangsekretariat senden.
14.06.2019	Hochladen aller verlangten Dokumente auf archiv-i.hsr.ch Abgabe des Berichts an den Betreuer bis 12.00 Uhr
14.06.2019	Präsentation und Ausstellung der Bachelorarbeiten, 16 bis 20 Uhr
05. - 23.08.2019	Mündliche BA-Prüfung
27.09.2019	Bachelorfeier

Regelmässig: **Wöchentliche Treffen: Mi 13:00 – ca. 15:00 / max. bis 16:00 Uhr**

- **Vorbereitung**: Agenda (Themen, Fragen):
  jeweils bis Mo durch Studierende
- **(kurzes) Protokoll**: jeweils bis Montag nach dem Meeting

Informationen zur Dokumentation und dem Ablauf der SA insbesondere dem Bericht finden Sie hier:

\\hsr.ch\root\alg\skripte\Informatik\Fachbereich\Bachelor-Arbeit_Informatik\BAI14

Der Bericht soll inhaltlich der dort hinterlegten „Anleitung Dokumentation BA_SA_181120" folgen. Der Bericht soll in gedruckter Form ohne Sourcen abgegeben werden. Bei den elektronischen Dokumenten (Sourcen, „Executable", usw.) reicht die Abgabe im Archivserver.

Achtung: Die Bilddatenbank der reformierten Kirche Horgen soll nicht auf den Archivserver geladen werden. Die Dateien unterliegen dem Datenschutz. Bilder, die zur Illustration im Bericht hinterlegt werden, müssen aus Datenschutzgesichtspunkten genehmigt werden (im Zweifel private Fotos oder Fotos des Betreuers verwenden).

Achtung: Die abgegebene Bachelorarbeit wird mit einer automatischen Plagiatserkennungssoftware überprüft. Bitte unbedingt darauf achten, dass korrekt zitiert wird, d.h. mit Angabe der Quelle. Bei wörtlichen Zitaten sind zusätzlich zur Quellenangabe auch Anführungszeichen erforderlich.

Quellenangaben sind auch erforderlich, wenn es sich um Online-Quellen handelt.

Massives Abschreiben ist keine eigenständige Leistung und führt zur Abwertung der Arbeit, selbst wenn korrekt zitiert wird.

Aufgabenstellung

Ausgangslage

Die reformierte Kirche Horgen verfügt über eine Bilddatenbank mit ca. 18000 Bildern, auf der Fotos von den verschiedensten Anlässen hinterlegt sind. Die Bilder der Datenbank sind gegenwärtig schlecht geordnet und nicht verschlagwortet.

In der Studienarbeit wurde im Rahmen eines POC ein Prototyp entwickelt, durch den die Bilddatenbank für den Endbenutzer durchsuchbar wird, ohne Datenschutzrechtliche Probleme einzugehen.

Ziel der Bachelorarbeit ist es, den "Prototyp" in ein "Produkt" zu überführen.

Lösungsansatz/Vorgehen

Im Rahmen eines agil angelegten Softwareentewicklungsprojekts soll der bestehende Prototyp in eine performante, leicht zu benutzende Webanwendung in Produktqualität überführt werden. Die Anwendung soll dabei einen Teil der folgenden Funktionen implementieren. Die Auswahl der Features ist Aufgabe der Studierenden und soll im Rahmen einer Requirement-Analyse mit dem Kunden durchgeführt werden.

Das Projekt soll, in Zusammenarbeit mit dem Kunden, Agil entwickelt werden.

Beheben von Fehlern:

- Automatisierung des Cleanups der Datenbank
- Ist die Google-Schnittstelle, die zur Keyword-basierten Suche verwendet wird, stabil genug?

Anpassungen am User-Interface:

- Hochladen neuer Bilder in die Datenbank
  - Abspeichern von Bild-Metainformationen (Anlass, Urheber, Datum, falls das Datum des Bildes zweifelhaft ist, eventuell Auswahl von Tags), die beim Hochladen vom Benutzer eingegeben werden
- Implementierung einer zusätzlichen Bildfilterfunktion, die es erlaubt, Bilder zu unterdrücken, welche den gewünschten Bild-Metainformationen nicht entsprechen
- Überarbeitung des Suchpannels:
  - Im Hauptpanel sollen neben den Bildern aus der Datenbank auch die Google-Bilder angezeigt werden. (Keyword-basierte Suche via Google soll erhalten bleiben)
  - Bei jedem Bild soll ein Indikator zeigen, ob das Bild für den vom Benutzer gewünschten Verwendungszweck geeignet ist, oder ob es lediglich zur Verfeinerung der Suche verwendet werden kann (Bilder aus der Google-Suche und Bilder, die nicht die gewünschten Metainformationen besitzen, sollen entsprechend gekennzeichnet sein)

- Als "ungeeignet gekennzeichnete" Bilder sollen optional ausgeblendet werden können
- Vereinfachung der Drill-Down-Suche. Doppelklick auf ein Bild sucht nach zu diesem Bild ähnlichen Objekten.
- Alternativ zum Doppelklick kann ein Bild ausgewählt werden und auf einen "Suchknopf" gedrückt werden. Auswahl mehrerer Bilder mit gedrückter Ctrl-Taste soll eine "Clustersuche" ermöglichen.

Neuer Algorithmus zur Clustersuche:

- Verbesserung des gegenwärtigen Suchalgorithmus, welcher nach zu mehreren Bildern ähnlichen Objekten sucht ("Clustersuche").
- Experimentieren mit möglichen Algorithmen:
  - Best lowest search rank first
  - Covariance-based approach (QDA)

Neue Funktionen:

- Implementierung einer Tag-basierten Suche:
  - Aktivierung von Meta-Informationsfiltern als zusätzliche Suchebene
  - Erweiterung dieser Funktionalität auf Bilder, bei denen Meta-Informationen fehlen: "Suche Bilder, die Ähnlichkeit mit Bildern haben, welche mit dem entsprechenden Tag versehen sind"
- Grouping-Funktionalität:
  - Entfernung von Duplikaten aus der Bilddatenbank
  - Gruppierung von sehr ähnlichen Bildern, so dass eine Suche eine grössere Variabilität ermöglicht
- Clusters-of-search-Anzeige
  - Nach Auswahl eines Bildes, soll ein möglichst variantenreicher Querschnitt über alle Matches gezeigt werden und nicht nur die ähnlichsten Bilder des ausgewählten Bildes.
  - Dazu muss ein Konzept für das "vollständige Ergebnis" einer Suchanfrage erarbeitet werden. Innerhalb dieses Ergebnisses können dann Cluster gebildet werden, von denen jeweils ein Repräsentant gezeigt wird.
  - Alternativ könnte auch die gesamte Datenbank in einer Baum-Struktur organisiert werden, in der "Cluster von Clustern" gebildet werden (funktioniert nur für Einzelbildsuche) oder bereits beim Upload statische "Mini-Cluster" aus sehr ähnlichen Bildern ermittelt werden (siehe Grouping).
- Scrolling:
  - Beim Scrollen werden weitere Bilder angezeigt (Vergrösserung des "vollständigen Ergebnisses")

Performance:

- Verwendung von Sparse-Matrices, Feature-Vector-Quantization und Duplikat-Elimination (ev. Bilden von Gruppen und einem Cluster-Tree während des Datenbankaufbaus).

Anhang B Persönliche Berichte

Tobias Saladin
In der Studienarbeit konnte ich einiges über die Konzepte von Machine Learning und Künstlicher Intelligenz lernen. In der Bachelorarbeit hatte ich vor allem im Bereich Software Engineering eine steile Lernkurve. Für mich als Quereinsteiger war es eine Bereicherung, ein komplettes Produkt von Anfang an bis zum Schluss entwickeln zu können. Mir war es wichtig, dass auch in der Bachelorarbeit Machine Learning Themen aufgegriffen werden. Glücklicherweise gab es diesbezüglich einige Herausforderungen zu bewältigen.

Ich bewerte unsere Bachelorarbeit als ein erfolgreiches Projekt. Das Feedback der Mitarbeitenden der Evangelisch-reformierten Kirche Horgen bestätigte mir dies zusätzlich. Das Erfolgsrezept lag vor allem in der Teamarbeit. Wie auch schon in der Studienarbeit ergänzen sich Martin und ich hervorragend. Ich konnte viel von Martin profitieren. Aufgetretene Herausforderungen konnten gemeinsam bewältigt werden, was sich in der erreichten Softwarequalität wiederspiegelt.

Die Zusammenarbeit mit den Mitarbeitenden der Evangelisch-reformierten Kirche Horgen empfand ich als sehr angenehm. Es war stets ein Interesse für unsere Arbeit zu spüren und das Ergebnis wurde durch konstruktive Inputs massgeblich verbessert.

Zu guter Letzt sei noch erwähnt, dass der Erfolg der Bachelorarbeit nur durch die Unterstützung von Prof. Oliver Augenstein möglich war. Ihm ist es gelungen uns die Konzepte verständlich zu erklären und uns stets den richtigen Weg zu zeigen.

Martin Odermatt
Nach der vorangehenden Studienarbeit, in der Konzepte und Algorithmen in Bereich Machine Learning in Bezug auf die Bildklassifikation erarbeitet wurden, war es schade, die gewonnenen Erkenntnisse nicht zu einem marktreifen Produkt umzuwandeln. Die Begleitperson Prof. Oliver Augenstein hatte uns dies nun in der vorgelegten Bachelorarbeit ermöglicht.

Die zu entwickelnde Software galt als besondere Herausforderung. Einerseits da sie letztendlich auf der beschränkten Infrastruktur der Evangelisch-reformierten Kirche Horgen lauffähig sein musste und nicht auf ein Webframework wie Django gesetzt werden konnte, wo schon sehr viele Anleitungen und Best Practices zur Verfügung stehen. Andererseits auch durch die gewählte Client-Server-Architektur, wodurch ein Frontend sowie ein Backend zu entwickeln waren, was einer Tätigkeit eines Fullstack-Entwicklers entspricht. Dies kostete enorm viel Zeit für Aufgaben in Bereich DevOps und ständiges Umdenken zwischen den Technologien.
Die Arbeit mit Open Source Libraries erwies sich teilweise als mühsam, da diese zueinander nicht immer kompatibel waren und man beinahe bei jedem Update einer Library wieder befürchten musste, dass etwas nicht funktioniert.

Insgesamt bin ich sehr zufrieden mit dem Resultat. Ich denke, wir konnten viel aus dem Studium Erlerntes anwenden und ein Projekt professionell führen. Letztendlich hat die Zusammenarbeit mit meinem Studienkollegen Tobias viel Spass gemacht und ich konnte mich stets auf ihn verlassen.

Anhang C Zeitabrechnung

Die Auswertung ergab zwei Abweichungen in der Zeitplanung. Der Feature Freeze sowie das Erstellen der Dokumentation wurden um eine Woche verschoben. In Abbildung 36 ist dies an den oben verschobenen Linien zu erkennen.

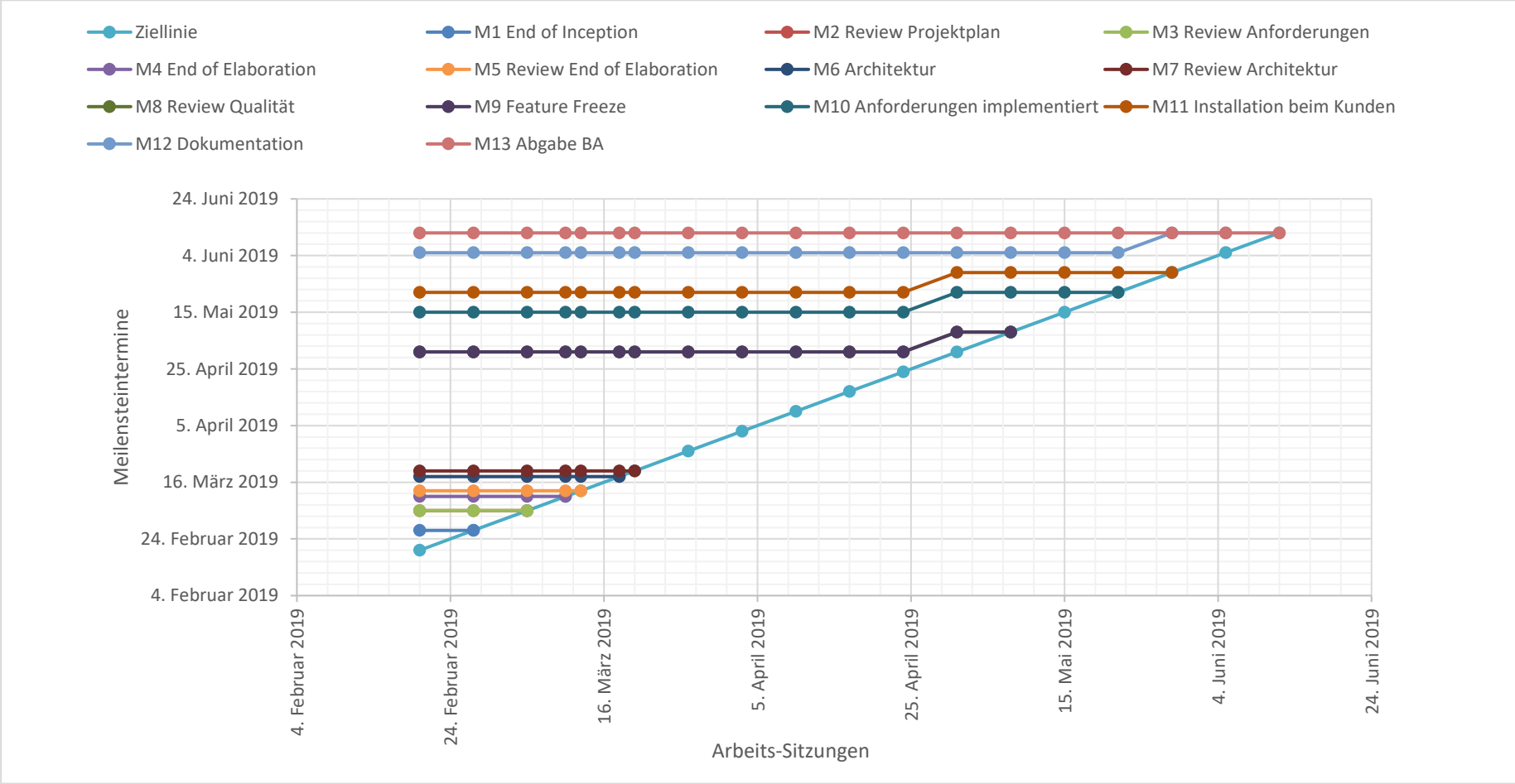


Abbildung 36: Zeitauswertungs-Grafik

Anmerkung. Eigene Darstellung.

Tabelle 36: Zeitauswertung der Stories

Kalenderwoche	Titel	Zugewiesen an	Story-Points Martin	Story-Points Tobias
KW 08	Projektplan erstellen	Tobias Saladin		1
KW 08	Evaluation Deployment	Tobias Saladin		1.5
KW 08	Projektplan erstellen	Martin Odermatt	1	
KW 08	Evaluation Deployment	Martin Odermatt	1.5	
KW 08	Sitzung mit Oliver und Dokumentation	Martin Odermatt	0.5	
KW 08	Sitzung mit Oliver und Dokumentation	Tobias Saladin		0.5
KW 09	Aufsetzen Sanic	Martin Odermatt	2.5	
KW 09	Einrichten Vue	Tobias Saladin		1
KW 09	Einrichten Vuetify	Tobias Saladin		0.5
KW 09	Erstellen Wireframes	Tobias Saladin		1
KW 09	Sitzung mit Oliver und Dokumentation	Tobias Saladin		0.5
KW 09	Sitzung mit Oliver und Dokumentation	Martin Odermatt	0.5	
KW 10	Implementation DB	Martin Odermatt	2.5	
KW 10	Testumgebung einrichten	Martin Odermatt	1	
KW 10	Evaluation Nearest Neighbors Search	Tobias Saladin		1
KW 10	Tests mit Vue.js	Tobias Saladin		1.5
KW 10	Sitzung mit Oliver und Dokumentation	Tobias Saladin		0.5
KW 10	Sitzung mit Oliver und Dokumentation	Martin Odermatt	0.5	
KW 11	Sitzungen Kunde, 12.03.2019	Tobias Saladin		0.5
KW 11	Integration TensorFlow	Tobias Saladin		2
KW 11	Dokumentation Schnittstellen	Martin Odermatt	0.5	
KW 11	Tests mit Annoy	Tobias Saladin		0.5
KW 11	Sitzung mit Oliver und Dokumentation	Tobias Saladin		0.5
KW 11	Sitzung mit Oliver und Dokumentation	Martin Odermatt	0.5	
KW 11	Sitzung mit Kunde, 12.03.2019	Martin Odermatt	0.5	
KW 12	Integration SA Prototyp Image Parts	Tobias Saladin		1
KW 12	Refactoring Sanic -> Multiple Workers	Martin Odermatt	0.5	
KW 12	Umsetzung Bildimport	Tobias Saladin		0.5

KW 12	Integration Google API	Tobias Saladin		1
KW 12	Umsetzung API	Martin Odermatt	1	
KW 12	Continous Integration	Martin Odermatt	1	
KW 12	Sitzung mit Oliver und Dokumentation	Martin Odermatt	0.5	
KW 12	Sitzung mit Oliver und Dokumentation	Tobias Saladin		0.5
KW 13	Test Bilder von Backend -> Frontend	Tobias Saladin		0.5
KW 13	Implementation lowest rank first	Tobias Saladin		0.5
KW 13	tags Funktionalität	Tobias Saladin		0.5
KW 13	Sitzung mit Oliver und Dokumentation	Tobias Saladin		0.5
KW 13	Sitzung mit Oliver und Dokumentation	Martin Odermatt	0.5	
KW 13	Integration Clustering	Tobias Saladin		1
KW 13	Umsetzung statische Files	Martin Odermatt	0.5	
KW 13	Swagger Demo	Martin Odermatt	1	
KW 13	Evaluation Sanic Ersatz	Martin Odermatt	1	
KW 14	Sitzung mit Oliver und Dokumentation	Martin Odermatt	0.5	
KW 14	Sitzung mit Oliver und Dokumentation	Tobias Saladin		0.5
KW 14	Evaluation Windows-Service	Martin Odermatt	0.5	
KW 14	Frontend: Image Section	Tobias Saladin		1
KW 14	Frontend: Image Upload	Tobias Saladin		1
KW 14	Frontend: Main layout	Tobias Saladin		0.5
KW 14	Demo FastApi, Durchstich	Martin Odermatt	1	
KW 14	Evaluation geeigneter Code Quality Tools	Martin Odermatt	0.5	
KW 14	Optimierung Continous Integration	Martin Odermatt	1	
KW 15	Sitzung mit Oliver und Dokumentation	Martin Odermatt	0.5	
KW 15	Sitzung mit Oliver und Dokumentation	Tobias Saladin		0.5
KW 15	Integration OpenApi3.0 von Swagger	Martin Odermatt	0.5	
KW 15	Optimierung des Datenbankmodells	Martin Odermatt	0.5	
KW 15	Wechsel von Sanic zu FastApi	Martin Odermatt	0.5	
KW 15	Integration Clusteranzeige	Tobias Saladin		1.5
KW 15	Testing	Martin Odermatt	0.5	

KW 15	Einbindung Flow	Martin Odermatt	0.5	
KW 15	Umsetzung API calls für Upload	Tobias Saladin		0.5
KW 15	Refactoring Upload View	Tobias Saladin		0.5
KW 16	Sitzung mit Oliver und Dokumentation	Tobias Saladin		0.5
KW 16	Sitzung mit Oliver und Dokumentation	Tobias Saladin		0.5
KW 16	Ignore Duplicates	Tobias Saladin		1
KW 16	Integration Author and Tags	Martin Odermatt	0.5	
KW 16	Refactoring Distance Matrix	Tobias Saladin		0.5
KW 16	Evaluieren Tag-Lösung	Martin Odermatt	0.5	
KW 16	Evaluieren Tag-Lösung	Tobias Saladin		0.5
KW 16	Code Refactoring	Martin Odermatt	2	
KW 17	Sitzung mit Oliver und Dokumentation	Martin Odermatt	0.5	
KW 17	Sitzung mit Oliver und Dokumentation	Tobias Saladin		0.5
KW 17	Anpassungen API	Tobias Saladin		0.5
KW 17	Performanz Optimierung Similarity Matrix und Clustering	Tobias Saladin		0.5
KW 17	Vorbereitung Zwischenpräsentation	Tobias Saladin		1
KW 17	Vorbereitung Zwischenpräsentation	Martin Odermatt	1	
KW 17	Zwischenpräsentation	Tobias Saladin		0.5
KW 17	Zwischenpräsentation	Martin Odermatt	0.5	
KW 17	Clean Azure Board, Update Use Cases	Martin Odermatt	1	
KW 18	Sitzung mit Oliver und Dokumentation	Martin Odermatt	0.5	
KW 18	Sitzung mit Oliver und Dokumentation	Tobias Saladin		0.5
KW 18	Optimize DB performance	Tobias Saladin		1
KW 18	Filtering	Tobias Saladin		1.5
KW 18	Image Edit	Tobias Saladin		1
KW 18	Meeting Notizen hochladen	Martin Odermatt		
KW 18	Fix Author reload	Tobias Saladin		0.1
KW 18	Sitzung mit Mitarbeitern der Reformierten Kirche Horgen	Tobias Saladin		0.5
KW 18	Sitzung mit Mitarbeitern der Reformierten Kirche Horgen	Martin Odermatt	0.5	
KW 19	Toggle Button	Tobias Saladin		0.2

KW 19	Clustering V2	Tobias Saladin		1
KW 19	Sitzung mit Oliver und Dokumentation	Martin Odermatt	0.5	
KW 19	Sitzung mit Oliver und Dokumentation	Tobias Saladin		0.5
KW 19	Bugfix: FilterEdit.vue	Tobias Saladin		0.5
KW 19	fix clustering v3	Tobias Saladin		1
KW 19	Bugfix: verwirrende Tag auswahl	Tobias Saladin		0.1
KW 19	Bugfix: Fotograf auswählen	Tobias Saladin		0.1
KW 19	Feature -> Delete image in small view	Tobias Saladin		0.1
KW 19	Bugfix -> Tag splits, Frozen Set	Martin Odermatt	1	
KW 20	Fix mypy	Martin Odermatt	0.5	
KW 20	Fix mypy	Martin Odermatt	0.5	0.1
KW 20	Bugfix: Align buttons	Tobias Saladin		0.1
KW 20	Bufix: Neu hinzugefügte Tags im Filter anzeigen	Tobias Saladin		0.1
KW 20	Maybe new calender	Tobias Saladin		0.5
KW 20	Bugfix: Clustering	Tobias Saladin		1
KW 20	Bugfix google -> First search goes to nirvana	Tobias Saladin		0.2
KW 20	Feature -> new calender	Tobias Saladin		0.5
KW 20	Check performance with analyzer	Tobias Saladin		0.7
KW 20	resolve TODO's	Martin Odermatt	0.5	
KW 20	Deployment WinService	Martin Odermatt	1	
KW 20	Check Sonar Errors	Tobias Saladin		0.5
KW 20	google schnee	Tobias Saladin		0.1
KW 20	add npm run lint to CI	Martin Odermatt	0.5	
KW 21	Test coverage	Martin Odermatt	0.5	
KW 21	Add logger	Martin Odermatt	1	
KW 21	Frontend Tests	Martin Odermatt	0.5	
KW 21	bug -> reset marked image	Tobias Saladin		0.5
KW 21	Installation Applikation	Tobias Saladin		0.5
KW 21	Installation Applikation	Martin Odermatt	0.5	
KW 21	bug -> swiss calendar, arial font	Tobias Saladin		0.3

KW 22	Korrekturen aus Usability Tests	Tobias Saladin		2
KW 22	Code Refactoring	Martin Odermatt	2.5	
KW 22	Block Dokumentation	Martin Odermatt	2	
KW 22	Block Dokumentation	Tobias Saladin		2
KW 23	Block Dokumentation	Martin Odermatt	4	
KW 23	Block Dokumentation	Tobias Saladin		4
KW 24	Update Tiana	Tobias Saladin		0.5
KW 24	Update Tiana	Martin Odermatt	0.5	
KW 24	Block Dokumentation	Martin Odermatt	7	
KW 24	Block Dokumentation	Tobias Saladin		4
Summe	**Story-Points**		**54 SP**	**56.2 SP**
Summe	**Stunden**		**378 h**	**393.4 h**

Anmerkung. Eigene Darstellung.

Anhang D Protokolle der Sitzungen

Tabelle 37: Sitzungesprotokoll KW 8 - 9

Woche	**KW 8 - 9**
Datum & Zeit	27.02.2019 / 17:00 Uhr
Teilnehmer	Oliver Augenstein, Martin Odermatt, Tobias Saladin
Traktanden	• Beschreibung der Aufgabenstellung • Besprechung der Vorgehensweise
Beschlüsse	• Erstellen einer Aufwandschätzung für die in der Aufgabenstellung definierten Aufgaben • Schreiben der Ausgangslage im Bericht

Anmerkung. Eigene Darstellung.

Tabelle 38: Sitzungesprotokoll KW 9 - 10

Woche	**KW 9 - 10**
Datum & Zeit	06.03.2019 / 14:00 Uhr
Teilnehmer	Oliver Augenstein, Martin Odermatt, Tobias Saladin
Traktanden	• Backend Lösung • Präsentation der Aufwandschätzung, Definition Storypoints, ect.
Beschlüsse	• Datenbereinigung -> älter als 2 Tage als Beispiel • Als Backend Frameworks wie Tornado oder CherryPy verwenden • Sitzung mit Kunden -> Wireframes, Metadaten, Anzahl Autoren • Es wird eine Zwischenpräsentation mit einem Experten geben. Dabei soll eine kurze Demo gezeigt werden.

Anmerkung. Eigene Darstellung.

Tabelle 39: Sitzungesprotokoll KW 10 - 11

Woche	**KW 10 - 11**
Datum & Zeit	13.03.2019 / 14:00 Uhr
Teilnehmer	Oliver Augenstein, Martin Odermatt, Tobias Saladin
Traktanden	• Web Frameworks • Erste Layout Ideen
Beschlüsse	• Analyse in Anhang von Frameworks • Up-/Download mit Onedrive • Backbutton • Google Bilder in gleichen Topf • Ahnliche in Bottomleiste des Bildes • Indikator wie viel wird von DB angzeigt • Stufenweise Vorrechnen

Anmerkung. Eigene Darstellung.

Tabelle 40: Sitzungesprotokoll KW 11 - 12

Woche	**KW 11 - 12**
Datum & Zeit	20.03.2019 / 08:15 Uhr
Teilnehmer	Tiana, Jürg, Martin Odermatt, Tobias Saladin
Traktanden	• Vorstellung der Änderungen gegenüber dem Prototyp • Präsentation der Wireframes

	• Annoy Bibliothek
Beschlüsse	• Angabe von Autor wichtig für den Kunden, da bei Verwendung von Bildern der Autor klar sein muss. • • Funktion für markierte Auswahl Tag verteilen • Zurüch Funktion umsetzen • Mehrere Bilder gleichzeitig exportieren

Anmerkung. Eigene Darstellung.

Tabelle 41: Sitzungesprotokoll KW 12 - 13

Woche	**KW 12 - 13**
Datum & Zeit	27.03.2019 / 13:00 Uhr
Teilnehmer	Oliver Augenstein, Martin Odermatt, Tobias Saladin
Traktanden	• Integration des Prototypen • Umsetzung Bildimport • Integration Google API • Continous Integration
Beschlüsse	• Struktur der Präsentation • Feature Vektoren persistieren • Echt gleiche entfernen • Clustergrösse an Resultat anpassen

Anmerkung. Eigene Darstellung.

Tabelle 42: Sitzungesprotokoll KW 13 - 14

Woche	**KW 13 - 14**
Datum & Zeit	03.04.2019 / 13:00 Uhr
Teilnehmer	Martin Odermatt, Tobias Saladin, Oliver Augenstein
Traktanden	• Integration OpenApi3.0 • Datenbankmodell • FastApi • Mögliche Lösung für das Clustering • Dokumentation
Beschlüsse	• Toggle für Clustering • Datum eintippen • Doppelklick für Bildersuche • Clustergrösse

Anmerkung. Eigene Darstellung.

Tabelle 43: Sitzungesprotokoll KW 15 - 16

Woche	**KW 15 - 16**
Datum & Zeit	17.04.2019 / 13:00 Uhr
Teilnehmer	Martin Odermatt, Tobias Saladin, Oliver Augenstein
Traktanden	• Umsetzung Bilder von DB im Frontend anzeigen • Deployment Diagramm, Domain Modell • Besprechung Machbarkeitsanalyse
Beschlüsse	• Abwägen Aufwand / Nutzen • Reihenfolge der Bilder überprüfen

Anmerkung. Eigene Darstellung.

Tabelle 44: Sitzungesprotokoll KW 16 - 17

Woche	**KW 16 - 17**
Datum & Zeit	24.04.2019 / 13:00 Uhr
Teilnehmer	Martin Odermatt, Tobias Saladin, Oliver Augenstein
Traktanden	• Duplikate ignorieren • Integration von Autor, Tags in der API • Refactoring Distanz Matrix
Beschlüsse	• Umsetzung Autor, Tags gemäss Notizen

Anmerkung. Eigene Darstellung.

Tabelle 45: Sitzungesprotokoll KW 17 - 18

Woche	**KW 17 - 18**
Datum & Zeit	01.05.2019 / 13:00 Uhr
Teilnehmer	Martin Odermatt, Tobias Saladin, Oliver Augenstein
Traktanden	• Filtering • Editieren • Datenbank optimieren • Aufsplitten Cluster • Testing
Beschlüsse	• Erste Zeile beim Clustering aufsplitten • Ergebnisse des Clusterings überprüfen

Anmerkung. Eigene Darstellung.

Tabelle 46: Sitzungesprotokoll KW 18 - 19

Woche	**KW 18 - 19**
Datum & Zeit	08.05.2019 / 13:00 Uhr
Teilnehmer	Martin Odermatt, Tobias Saladin, Oliver Augenstein
Traktanden	• Anpassungen Clustering • Umsetzung neue Anforderungen Kirche Horgen • Refactoring Backend • Testcoverage
Beschlüsse	• Neues Clustering beibehalten • Sitzungen nach Bedarf

Anmerkung. Eigene Darstellung.

Tabelle 47: Sitzungesprotokoll KW 19 - 20

Woche	**KW 19 - 20**
Datum & Zeit	15.05.2019 / 13:00 Uhr
Teilnehmer	Martin Odermatt, Tobias Saladin, Oliver Augenstein
Traktanden	• Erneute Anpassungen am Clustering • Abschliessende Verbesserungen im Frontend • Diverse Fehlerbehebungen • Verbesserung Source-Code Qualität
Beschlüsse	• Letzte Anpassungen am Clustering vornehmen • Feedback beim Kunden einholen

Anmerkung. Eigene Darstellung.

Anhang E Software Qualität

In diesem Abschnitt werden einige Metriken zu Lines of Code, Statements, Functions und Classes gemessen.

Lines of Code
Nachfolgend wurde mittels des PyCharm-Plugins „MetricsReloaded" [52] die Lines of Code sowie die Non-comment Lines of Code gemessen. Dabei werden leere Zeilen nicht miteinberechnet. Dies soll ein Gefühl für den Umfang vermitteln. Die Zahlen sind jedoch mit Vorsicht zu geniessen. Es spielt neben der Programmiersprache und dem festgelegten Styleguide sowie Zeilenlänge eine grosse Rolle, wie der Code geschrieben wurde. Dies soll in Abbildung 37 und
Abbildung 38 gezeigt werden.

```
uris: Set[str] = set()
for entry in images:
    uris.add(entry)

return uris
```

Abbildung 37: Lines of Code - Python (schlecht)

Anmerkung. Eigene Darstellung.

```
return set([entry.download_uri for entry in images])
```

Abbildung 38: Lines of Code - Python (gut)

Anmerkung. Eigene Darstellung.

Der Code in Abbildung 37 würde als 4 Lines of Code gezählt werden, während in Abbildung 38 lediglich 1 Zeile gezählt werden würde.

Backend

Tabelle 48: Lines of Code - Backend

File	Lines of Code	Non-Comment Lines of Code
backend/app/core/config.py	79	76
backend/app/crud/cluster_manager.py	296	293
backend/app/crud/image_manger.py	155	155
backend/app/crud/vector_manager.py	22	22
backend/app/db/init_db.py	24	24
backend/app/db/session.py	3	3
backend/app/entities/author_model.py	5	5
backend/app/entities/base_model.py	6	6
backend/app/entities/cluster_image_model.py	7	7
backend/app/entities/cluster_model.py	4	4
backend/app/entities/file_type_model.py	4	4
backend/app/entities/image_model.py	13	13
backend/app/entities/image_tag_model.py	7	7
backend/app/entities/overview_cluster_model.py	5	5
backend/app/entities/tag_model.py	4	4
backend/app/entities/vector_model.py	6	6
backend/app/http/endpoints/author.py	13	13
backend/app/http/endpoints/cluster.py	15	15

backend/app/http/endpoints/heartbeat.py	13	13
backend/app/http/endpoints/image.py	120	120
backend/app/http/endpoints/static.py	6	6
backend/app/http/endpoints/tag.py	8	8
backend/app/http/api.py	9	9
backend/app/models/author.py	6	6
backend/app/models/cluster.py	13	13
backend/app/models/filter.py	8	8
backend/app/models/image.py	20	20
backend/app/models/message.py	3	3
backend/app/models/tag.py	8	8
backend/app/services/annoy_service.py	214	210
backend/app/services/google_service.py	55	55
backend/app/services/image_service.py	192	192
backend/app/services/tensor_service.py	55	55
backend/app/tests/api/test_heartbeat.py	9	9
backend/app/tests/api/test_image.py	25	24
backend/app/tests/api/test_static.py	16	16
backend/app/tests/crud/test_image_manager.py	142	124
backend/app/tests/environment/test_env.py	15	15
backend/app/tests/services/test_annoy_service.py	68	68
backend/app/tests/services/test_google_service.py	18	15
backend/app/tests/services/test_image_service.py	39	33
backend/app/tests/utils/utils.py	39	39
backend/app/tests/conftest.py	64	64
backend/app/utils/date_utils.py	9	9
backend/app/main.py	63	63
backend/app/tests_pre_start.py	10	10
server.py	7	7

Anmerkung. Eigene Darstellung.

Wie aus Tabelle 48 hervorgeht, sind es im Backend insgesamt 1922 Lines of code, davon 1884 Non-Comment Lines of Code.

Frontend

Tabelle 49: Lines of Code - Frontend

File	Lines of Code	Non-Comment Lines of Code
frontend/public/index.html	14	14
frontend/src/api-services[1]	2192	1008
frontend/src/components/AuthorSelecting.vue	73	73
frontend/src/components/BottomLine.vue	72	72
frontend/src/components/DatePicker.vue	130	130
frontend/src/components/EmptyList.vue	53	53
frontend/src/components/FileUpload.vue	295	295
frontend/src/components/FilterEdit.vue	141	141
frontend/src/components/ImageEdit.vue	273	273
frontend/src/components/AuthorSelecting.vue	592	592
frontend/src/components/Main.vue	69	69
frontend/src/components/TagSelecting.vue	92	92
frontend/src/components/TopLine.vue	308	308

[1] Automatisch generierter Code

frontend/src/plugins/vuetify.js	12	12
frontend/src/router/index.js	14	14
frontend/src/App.vue	23	23
frontend/src/main.js	17	14
frontend/tests/unit/BottomLine.spec.js	21	19
frontend/tests/unit/FilterEdit.spec.js	40	35
frontend/tests/unit/ImageSection.spec.js	22	21
frontend/tests/unit/TopLine.spec.js	67	54

Anmerkung. Eigene Darstellung.

Aus Tabelle 49 lässt sich herauslesen, dass das Frontend total 4520 Lines of code enthält, davon 3312 Non-Comment Lines of Code. Wenn der automatisch generierte Code noch subtrahiert wird, sind es 2304 Non-Comment Lines of Code.

Scripts

Tabelle 50: Lines of Code - Scripts

File	Lines of Code	Non-Comment Lines of Code
scripts/create_windows_console_exe.ps1	8	8
scripts/create_windows_service_exe.ps1	8	8
scripts/run_all_backend_tests.ps1	6	6
scripts/run_all_backend_tests.sh	7	7
scripts/run_all_frontend_tests.ps1	3	3
scripts/run_all_frontend_tests.sh1	3	3
scripts/run_backend_linter.ps1	6	6
scripts/run_backend_linter.sh	7	7

Anmerkung. Eigene Darstellung.

Alle Scripts, die in Tabelle 50 aufgelistet sind, enthalten zusammen 48 Lines of Code

Config Files

Tabelle 51: Lines of Code - Config Files

File	Lines of Code	Non-Comment Lines of Code
frontend/.browserlistrc	2	2
frontend/.eslintignore	5	5
frontend/.eslintrc.js	17	17
frontend/.flowconfig	0	0
frontend/.gitignore	21	21
frontend/babel.config.js	8	8
frontend/jest.config.js	35	35
frontend/package.json	63	63
frontend/postcss.config.js	5	5
frontend/vue.config.js	14	14
.editorconfig	8	8
.gitignore	106	80
.secrets	3	3
azure-pipelines.yml	78	78
codecov.yml	29	29
ImageFinderConsole.spec	38	36
ImageFinderWindowsService.spec	38	36

logging.ymal	21	21
mypy.ini	6	6
Pipfile	39	39
pytest.ini	3	3
settings.toml	32	32
sonar.project.properties	8	8

Anmerkung. Eigene Darstellung.

In Tabelle 51 sind alle Config Files aufgelistet. Diese wurden teilweise mittels Vue-CLI [53], Pyinstaller [54] sowie GitHub [45] generiert und ergänzt. Total summiert sind es 579 Lines of Code, davon 549 Non-Comment Lines of Code.

Statements
Das Backend enthält über drei mal mehr Statements (if, while, for, throw, catch etc.) als das Frontend. Dies ist aus Abbildung 39 zu entnehmen.

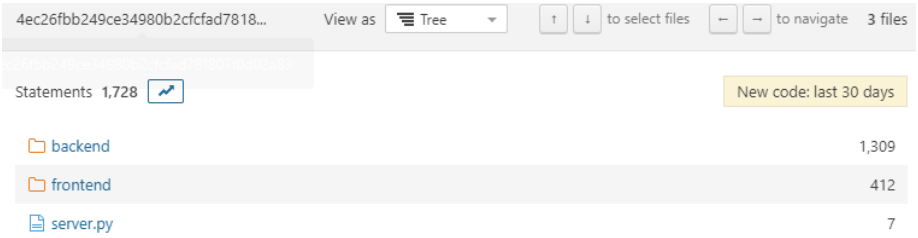


Abbildung 39: SonarCloud - Size Statements

Classes
Da in der Python-Programmiersprache die kleinste Einheit einem Modul entspricht, wurden lediglich Klassen für die Entities (vgl. Abbildung 17: Backendarchitektur - Data Access Layer, Seite 45) und Models (vgl. Abbildung 15: Backendarchitektur - Common Layer) eingesetzt.

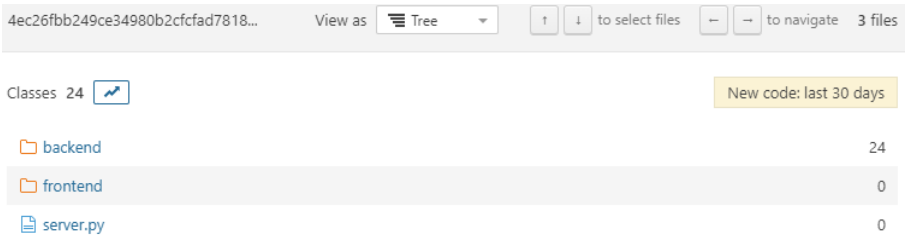


Abbildung 40: SonarCloud - Size Classes

SonarCloud zählt wie in Abbildung 40 illustriert, insgesamt 24 Klassen.

Functions
Obwohl die Funktionen im Frontend grösstenteils lediglich Logik für die der Darstellung der Weboberfläche enthalten, sind fast genau so viele wie im Backend vorhanden. Dies ist in Abbildung 41 ersichtlich.

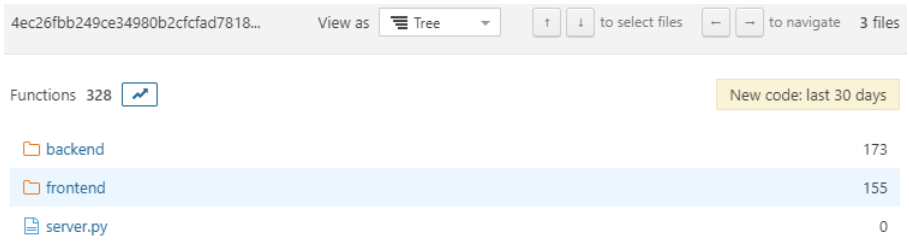


Abbildung 41: SonarCloud - Size Functions

Konformität mit Coding Guidelines
In diesem Abschnitt werden die in Anhang E definierten Guidelines überprüft.

Wie in Abbildung 42 zu sehen ist, werden keine im Frontend keine Fehler von ESLint, Airbnb-Styleguide oder Vue.js-Plugin rapportiert.

```
PS C:\Users\maod\source\ba\ImageFinder\frontend> npm run lint

> vue-template@0.1.0 lint C:\Users\maod\source\ba\ImageFinder\frontend
> vue-cli-service lint --fix

 DONE  No lint errors found!
PS C:\Users\maod\source\ba\ImageFinder\frontend>
```

Abbildung 42: Einhaltung der Styleguides von ESLint, Airbnb, Vue.js

Anmerkung. Eigene Darstellung.

Im Backend ist die Code-Formattierung nach Black eingehalten. Dies ist in Abbildung 43 zu sehen.

```
PS C:\Users\maod\source\ba\ImageFinder> pipenv run black backend/
All done! □ □□ □
62 files left unchanged.
PS C:\Users\maod\source\ba\ImageFinder>
```

Abbildung 43: Einhaltung des Styleguides nach Black

Anmerkung. Eigene Darstellung.

Potentiell gefährliche Konstrukte
Nachfolgend werden potentielle Risiken im Bereich Security mit Hilfe des Scanners SonarCloud ausgewertet.

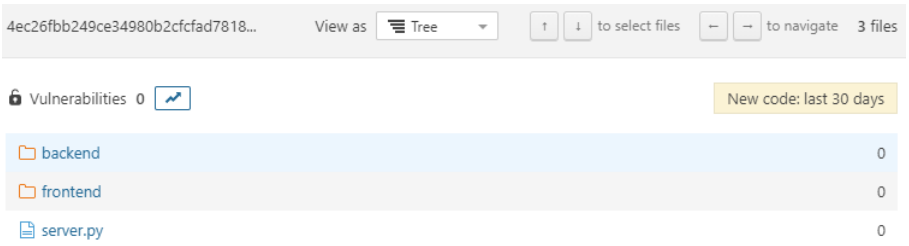


Abbildung 44: SonarCloud - Security Vulnerabilities

Aus Abbildung 44 geht hervor, dass keine Security Vulnerabilities gefunden wurden. Vulnerabilities sind in diesem Kontext als mögliche Angriffspunkte zu verstehen. [55] In Abbildung 45 ist zu sehen, dass die Software mit einem Rating der höchsten Stufe bewertet wurde.

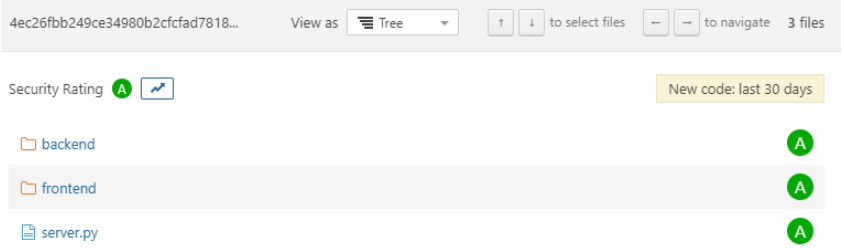


Abbildung 45: SonarCloud - Security Rating

Anmerkung. Eigene Darstellung.

Somit ist der geschätzte Auffwand, der investiert werden muss, um mögliche Sicherheitsrisiken zu beheben, gleich dem Wert 0. Dies ist aus Abbildung 46 zu entnehmen.

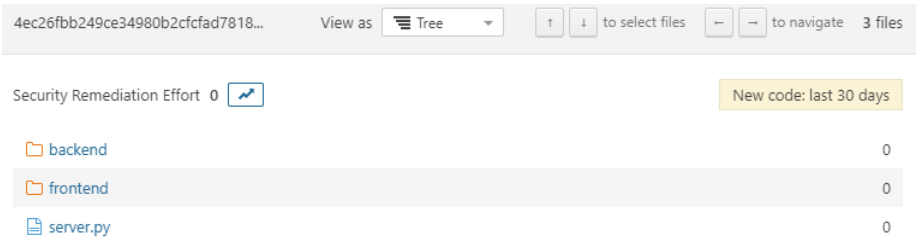


Abbildung 46: SonarCloud - Security Remediation Effort

Abbildung 47 sagt aus, dass keine Security Hotspots, also sicherheitskritische Stellen, die ein Review benötigen, gefunden wurden.

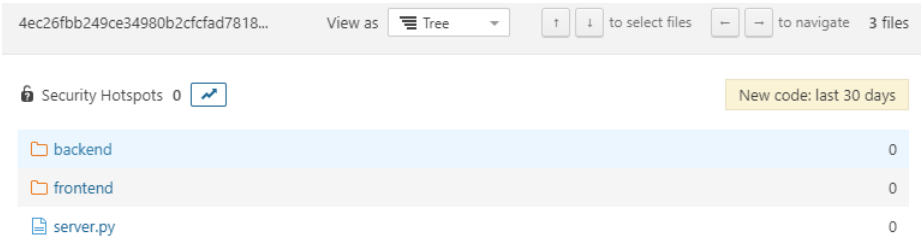


Abbildung 47: SonarCloud - Security Hotspots

Software Reliability
SonarCloud hilft mögliche Fehler zu erkennen. In diesem Abschnitt sind diesbezüglich Ausschnitte der Einstufung von SonarCloud zu finden.

In Abbildung 48 ist zu sehen, dass keine Fehler (Bugs) gefunden wurden. Der Source Code erhält somit das Rating A, was aus Abbildung 49 zu entnehmen ist.

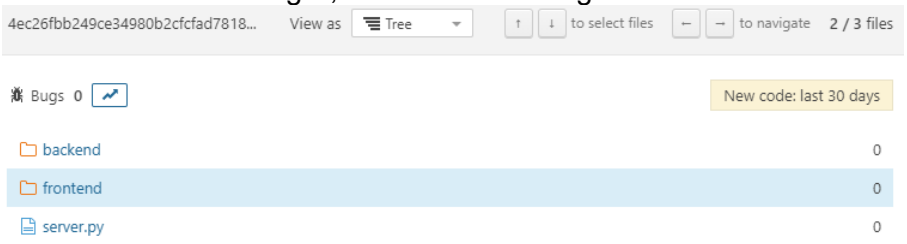


Abbildung 48: SonarCloud - Reliability Bugs

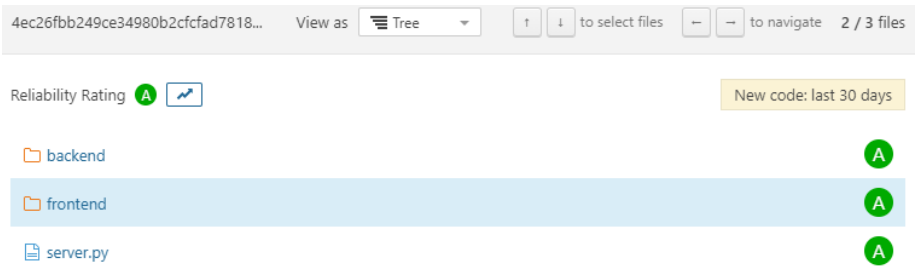


Abbildung 49: SonarCloud - Reliability Rating

Abbildung 50: DeepScan - Issues

Die total 19 gefundenen Issues von DeepScan, welcher das Frontend analysiert, wurden zuletzt am 7. Mai 2019 behoben. In Abbildung 50 ist zu sehen, dass der Source Code den Grade „Good" erhält, was der höchste zu erreichende Stufe entspricht. [56]

Software Maintainability

Nachfolgend wird aus SonarCloud die Wartbarkeit (Maintainability) der Software ausgelesen. In Abbildung 51 ist zu sehen, dass keine Code Smells gefunden wurden.

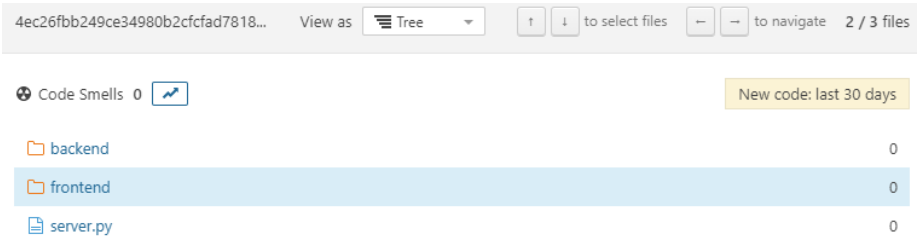


Abbildung 51: SonarCloud - Maintainability Code Smells

Abbildung 52 zeigt die „Technical Dept", bzw. den Effort, der aufgewendet werden muss um die Code Smells zu beheben. Da keine Code Smells vorhanden sind, beträgt der Wert gleich 0.

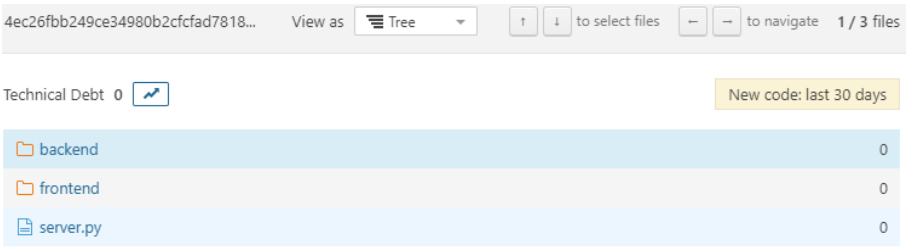


Abbildung 52: SonarCloud - Maintainability Dept

Der Source Code wird somit mit einem Rating der Stufe A bewertet (Abbildung 53).

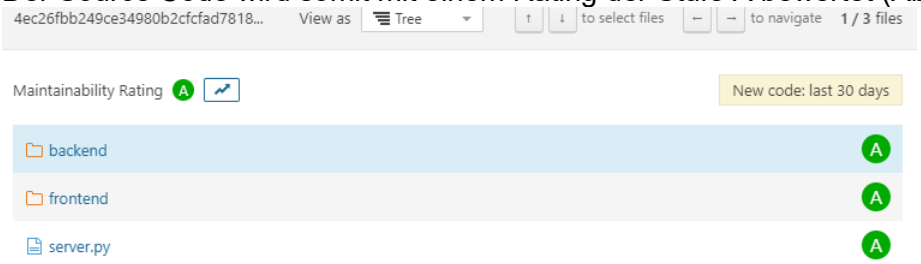


Abbildung 53: SonarCloud - Maintainability Rating

Software Duplications

SonarCloud konnte wie in Abbildung 54 zu sehen ist, keine duplizierten Code-Zeilen finden.

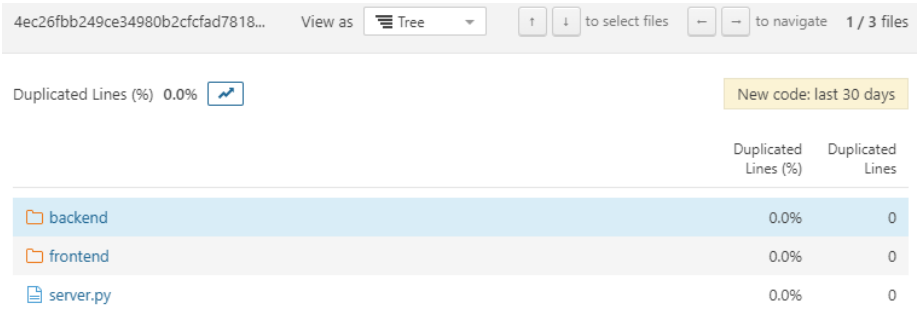


Abbildung 54: SonarCloud – Duplications

Namen

Um für Konsistenz zu sorgen, wurden im Python-Backend einige Namensgebungen eingeführt.

Endpoints

Die Funktionen der Endpoints (siehe Abbildung 14, Seite 43) werden konsequent nach dem Schema in Tabelle 52 benannt.

Tabelle 52: Namensgebung der Funktionen im Package endpoints

Prefix	Beschreibung
create_	Erstellt in der Datenbank oder auf dem Filesystem einen neuen Eintrag.
read_	Liest lediglich von der Datenbank oder vom Filesystem.
update_	Führt ein Update in der Datenbank oder dem Filesystem durch.

Anmerkung. Eigene Darstellung.

Dies gibt einen Hinweis, welche Operation letztendlich durchgeführt wird.

Services
Im Business Logic Layer (vgl. Abbildung 16: Backendarchitektur - Business Logic Layer, Seite 44) erhalten die Module den Suffix „_service”.

Manager
Module, die für das Speichern und Lesen von der Datenbank verantwortlich sind (vgl. Abbildung 17: Backendarchitektur - Data Access Layer, Seite 45), werden mit einem Suffix „_manager” benannt.

Entities
Entities (vgl. Abbildung 17: Backendarchitektur - Data Access Layer, Seite 45) werden alle mit einem Suffix „_model” benannt.

Testabdeckung
Anhand des Commits mit der Id bb0597c werden nachfolgend Ausschnite der berechneten Testabdeckung aus Codecov [57] gezeigt. Dieser Stand repräsentiert nicht den Endstand, der bei der Einreichung der Bachelorarbeit erreicht wurde. Seit einem Update der von Microsoft zur Verfügung gestellten Infrastruktur, die für das Continuous Integration verwendet wird, funktionieren die Builds nicht mehr. Die SQLite Datenbank, die automatisch mit Python ausgeliefert wird, scheint nicht vorhanden zu sein. Das Problem ist bei Microsoft bekannt und soll in den nächsten Wochen behoben werden. [58]

Für das Testen der Software wurde eine Mischung aus Unit- und Integrationtests gewählt.
In Abbildung 55 ist die Testabdeckung in Form eines Sunburst-Graphen zu sehen. Codecov beschreibt den Graph wie folgt: „The inner-most circle is the entire project, moving away from the center are folders then, finally, a single file. The size and color of each slice is representing the number of statements and the coverage, respectively.” [59]

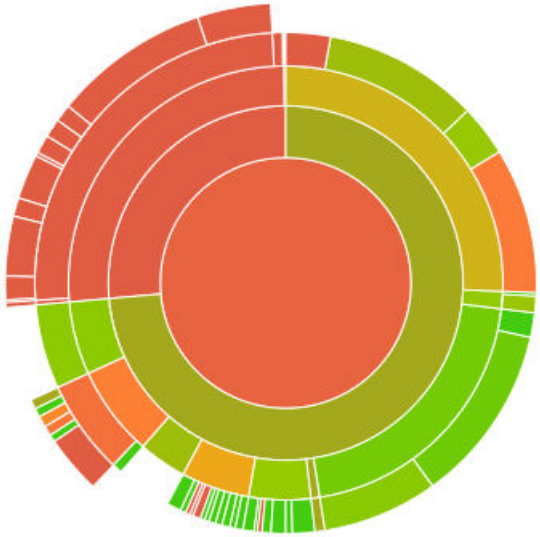


Abbildung 55: Codecov - Testabdeckung Sunburst-Graph

In der linken oberen Ecke sind die Frontend-Tests angesiedelt. Diese erscheinen in orange Farbe, da wie in Abbildung 56 zu sehen ist, im Frontend lediglich eine Testabdeckung von rund 24.5% erreicht wurde. Bewusst wurde im Frontend keine hohe Testabdeckung angestrebt, da diese beinahe nur Logik für die Darstellung der Weboberfläche enthaltet. Die Services, die genutzt werden, um mit dem Backend über HTTP zu kommunizieren (vgl. Abbildung 12, Seite 41) und die eigentliche Logik enthalten, werden bei der Berechnung der Testabdeckung nicht miteinbezogen. An dieser Stelle wird davon ausgegangen, dass diese Services bereits getestet wurden.

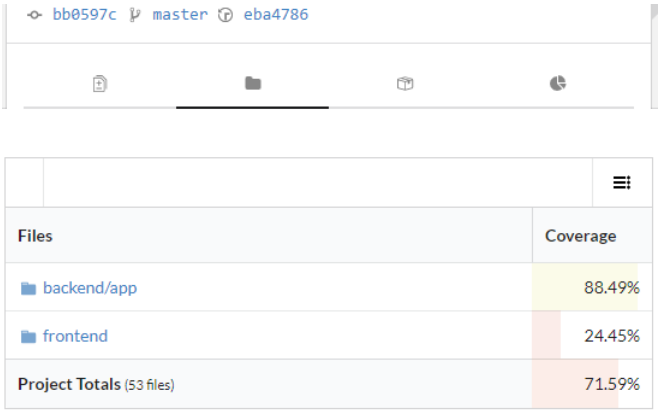


Abbildung 56: Codecov - Testabdeckung Übersicht

Insgesamt wurde eine Testabdeckung von rund 71.6 % erreicht, wobei das Backend eine Abdeckung von beinahe 88.5% aufweist.

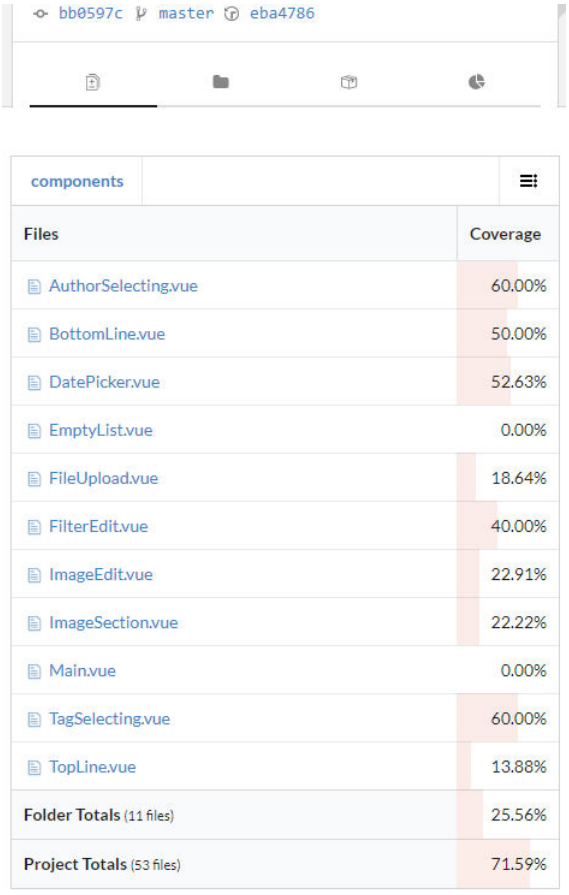


Abbildung 57: Codecov - Testabdeckung Frontend

Wie in Abbildung 57 zu sehen ist, wurden im Frontend vor allem die Komponenten AuthorSelecting, DatePicker und TagSelecting getestet. Diese werden von anderen Komponenten importiert (vgl. Abbildung 11Abbildung 11: Frontendarchitektur - Presentation Layer) und stellen somit gemeinsame Funktionalität zur Verfügung.

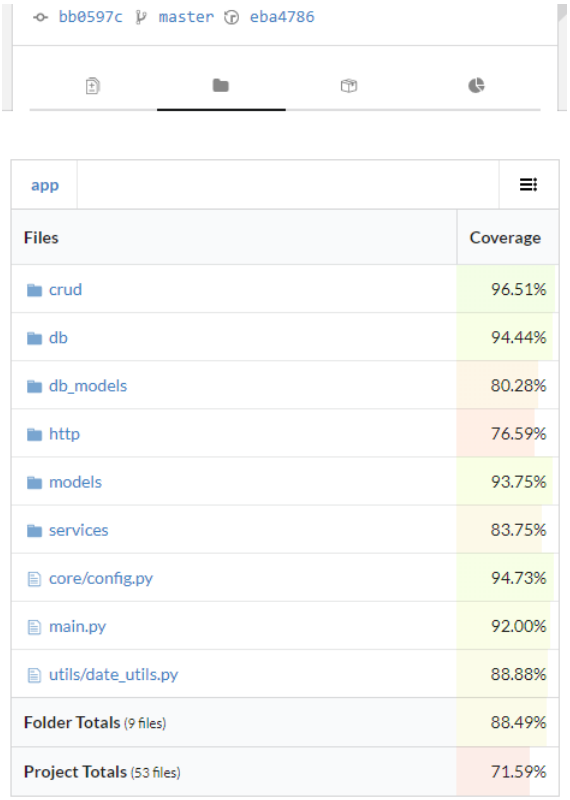


Abbildung 58: Codecov - Testabdeckung Backend

Die Darstellung in Abbildung 58 zeigt eine Übersicht über die Testabdeckung im Backend. Mit 76.59% weisen die HTTP-Endpoints die geringste Abdeckung im Backend auf. Dies wurde in einem späteren Commit erhöht, jedoch liegt aufgrund der erwähnten Probleme mit der Microsoft Infrastruktur kein neuerer Wert vor.

Allgemein kann gesagt werden, dass je näher sich die Abdeckung gegen 100% neigt, desto aufwendiger wird es, geeignete Tests zu schreiben. Deshalb ist es lediglich für komplexe Berechnungen sinnvoll, Unit Tests zu schreiben. Ansonsten erreicht man mit Integration Tests eine wesentlich höhere Abdeckung. Abschliessend ist zu erwähnen, dass auch eine Testabdeckung in der Höhe von 100% keine Garantie für funktionierende Software ist. Dies bedeutet lediglich, dass alle Pfäde einmal durchlaufen worden sind.

Anhang F Tools und Infrastruktur

In Abbildung 59 sind die für die Softwarequalität relevanten Bereiche und Tools abgebildet.

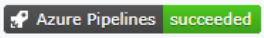
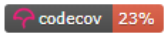
Area	Tool(s)
Repository	
Build	
Code Style	
Continuous Code Quality	
Test Coverage (Frontend, Backend)	

Abbildung 59: Softwarequalität - Tools

Repository
Der Source Code wird auf GitHub [45] gehostet. Dies erlaubt die kostenlose Nutzung (als Open Source Projekt) und einfache Anbindung von weiteren Tools.

Build
Für das Continuous Integration wird eine von Microsoft gehostet Azure Pipeline [60] verwendet.

Code Style
Im Frontend werden potenzielle Fehler mit eslint [61] erkannt. Dabei ergänzt der JavaScript Style Guide airbnb [62] die statische Prüfung und definiert zusätzlich Regeln, wie moderner Code auszusehen hat. Speziell für das Frontendframework Vue.js wurde das Plugin vue/recommended [63] installiert. Dieses Plugin wird ergänzend zu eslint eingesetzt und kennt drei Stufen (Priority A: Essential, Priority B – Strongly Recommended, Priority C: Recommended). In diesem Setup wird die Priority C anvisiert.

Im Python-Backend wird Black [64] eingesetzt, der als ein striktes Subset von PEP 8 [65] gesehen werden kann.

Continuous Code Quality: Das ganze Projekt wird mit SonarCloud [66] gescannt. Dabei werden die von den Code Styles definierten und nicht eingehaltene Regeln als Bugs oder Vulnerabilities gekennzeichnet. Ausserdem lassen sich aus SonarCloud noch weitere Metriken wie zum Beispiel Code Smells, Code Duplication und Technical Dept extrahieren. Ein Printscreen ist in Abbildung 60 zu sehen.

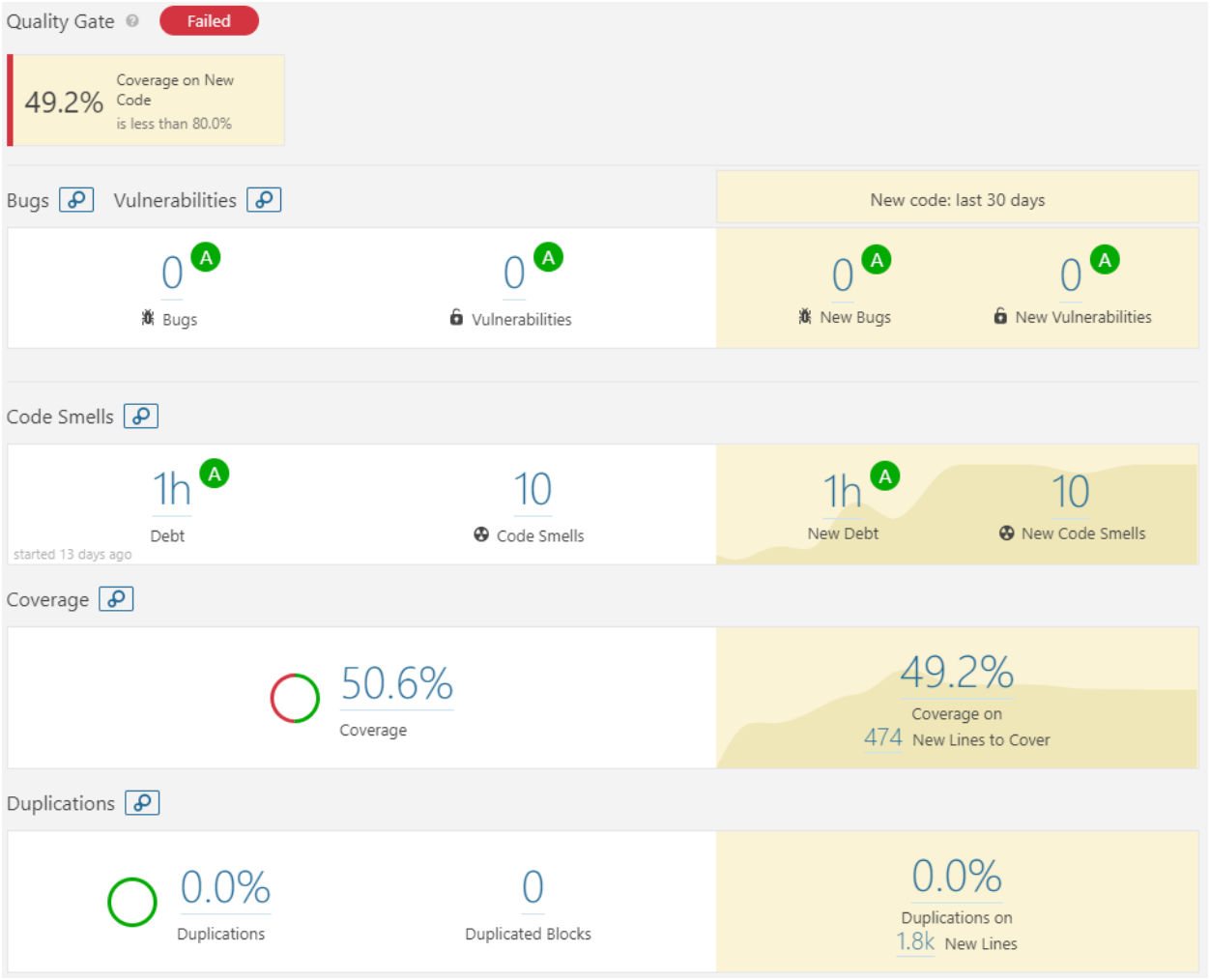


Abbildung 60: SonarCloud - Code Quality

Des Weiteren können OWASP Top 10 Sicherheitsrisiken, wie in Abbildung 61 ersichtlich, bis zu einem gewissen Grad erkannt werden. SonarCloud weist automatisch auf kritische Code Stellen hin, die dann manuell geprüft werden müssen.

Categories	Vulnerabilities	Security Hotspots		
		Open	In Review	Won't Fix
A1 - Injection	0 A	2	0	0
A2 - Broken Authentication	-	-	-	-
A3 - Sensitive Data Exposure	0 A	2	0	0
A4 - XML External Entities (XXE)	-	-	-	-
A5 - Broken Access Control	-	-	-	-
A6 - Security Misconfiguration	0 A	0	0	0
A7 - Cross-Site Scripting (XSS)	0 A	0	0	0
A8 - Insecure Deserialization	-	-	-	-
A9 - Using Components with Known Vulnerabilities	-	-	-	-
A10 - Insufficient Logging & Monitoring	-	-	-	-
Not OWASP	0 A	0	0	0

Abbildung 61: SonarCloud - OWASP Top 10

Alle Metriken und Sicherheitsrisiken werden, wie in **Fehler! Verweisquelle konnte nicht gefunden werden.** ersichtlich, mit einem Grade von A (beste) bis E (schlechteste) bewertet.

DeepScan
Zusätzlich wird DeepScan [67] für die Code Analyse eingesetzt. Dieses Tool folgt dem JavaScript Datenfluss und erkennt dadurch Fehler wie zum Beispiel „Unreachable Code". [68] Ausserdem unterstützt DeepScan Vue.js spezifische Regeln. [69] Ein Screenshot ist in Abbildung 62 zu sehen.

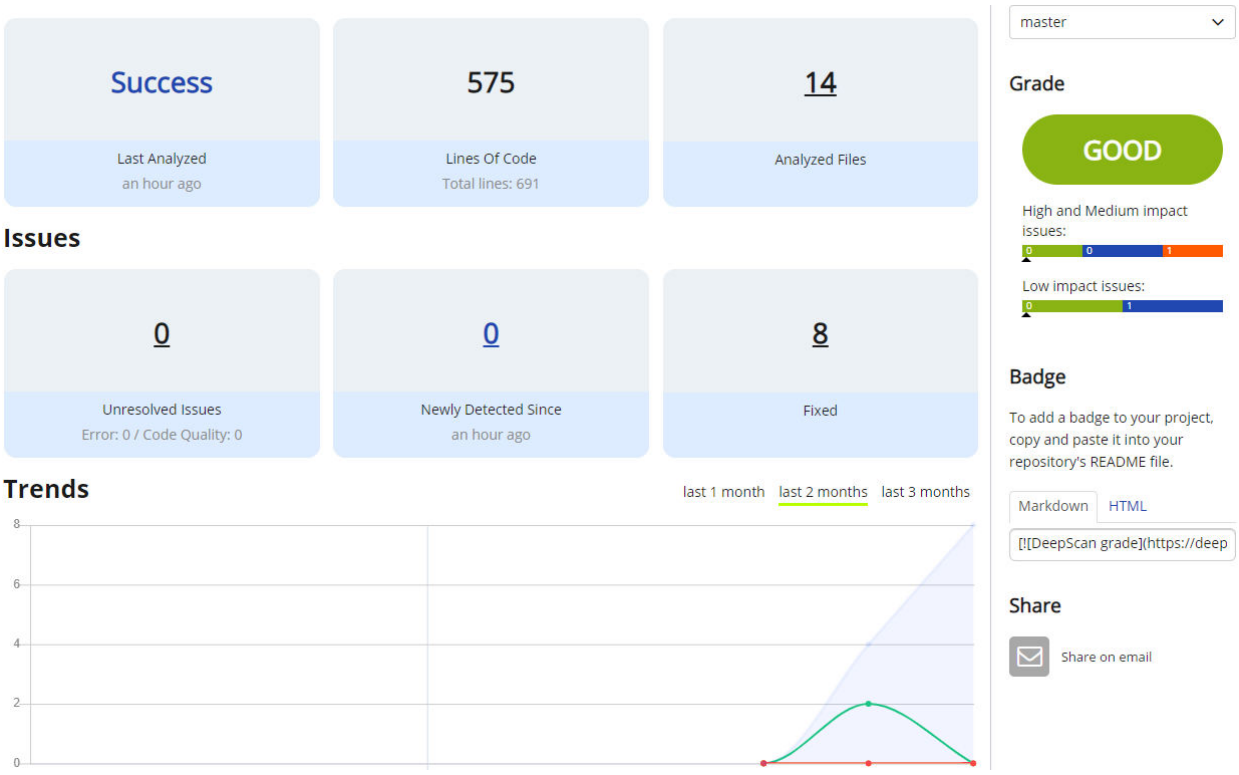


Abbildung 62: DeepScan Screenshot

Nach einem durchlauf wird der Source Code mit einem Rating von „Good", „Normal" oder „Poor" versehen.

Code Coverage
Bei jedem Build wird die Testabdeckung des Frontends und des Backends gemessen und auf Codecov [57] publiziert.

Entwicklungsumgebung
PyCharm Professional 2019.1 [70] dient als Integrated Development Environment (IDE).

PyInstaller
Damit aus Python-Code ein unter Windows ausführbares Programm (.exe) erstellt werden kann, wird der Source Code mit PyInstaller [54] paketiert.

Anhang G Evaluation Backend / Technologien

In diesem Kapitel wird nach einem für die Umsetzung geeigneten Webframework gesucht. Dabei wird zuerst die CRC-Card-Methode verwendet. Anschliessend werden die geeigneten Kandidaten näher betrachtet und in einer Nutzwertanalyse evaluiert.

Python Webframework - Class Responsibility Collaboration (CRC) Card
In Tabelle 53 sollen die Aufgabenbereiche sowie die Kollaborateure in Form einer für den Zweck angepassten CRC Card [71] festgehalten werden, wie es an der HSR im Modul „Application Architecture“ gelehrt wird.

Tabelle 53: CRC Card - Python Webframework

Component: Python Webframework	
Responsibilities	**Collaborators** (Interfaces to/from)
• Soll als Webserver dienen oder einen Adapter für einen Webserver bereitstellen (WSGI-Standard) • Abstraktion (Middleware) für HTTP-Requests • Entgegennahme von HTTP-Requests via gängige HTTP-Verben (GET, POST, PUT, DELETE) • Statische Files (HTML) ausliefern • Weitere Prozesse und/oder Threads für CPU-lastige Arbeit (Blockierung eines Cores für länger als 10 Sekunden in Folge) starten	• Presentation Layer • Business Logic Layer • Data Access Layer • Filesystem • (Kommunikation durch WSGI-Schnittstelle von und zu andere Python Webserver)
Candidate implementations technologies (and known uses)	
Django, TurboGears, Web2Py, Bottle, CherryPy, Flask, Sanic, Tornado, Pyramid, Responder, Hug, Falcon, FastAPI	

Anmerkung. Eigene Darstellung.

Evaluation Backend Webframework
Da die Programmiersprache Python enorm populär geworden ist, scheint es nicht verwunderlich, dass sehr viele Webframeworks aufgekommen sind. In Tabelle 54 wird auf eine Auswahl eingegangen.

Tabelle 54: Webframeworks im Vergleich

Webframework	**Slogan**	**Besonderheiten**
Django	The web framework for perfectionists with deadlines	• Ausgezeichnete Dokumentation • Grosse Community • Viele out-of-the-box Features • Template in PyCharm integriert • Django REST kann als Modul hinzugefügt werden[72] • •

Turbo-Gears	The web framework that scales with you	• 20 minütiges Full-Stack Tutorial[73] • • • Möglichkeit mit einem Microframework zu starten und zu einem Full-Stack Framework aufzurüsten • Beginner freundlich, z.B. Hello World Einführungs-Video
Web2Py	Free and open source full-stack enterprise framework […]	• „Batteries includes[74]“ Ansatz: Monolith, der ohne Dependencies von Dritten auskommt • Integrierter, multi-threaded Webserver • „Enterprise Framework”[75] • • REST nur teilweise unterstützt[76] • •
Bottle	[…] fast and simple micro-framework for python web-applications	• Sehr schlank und für Prototypen sowie kleinere Webapplikationen geeignet • Single-File Modul, keine Abhängigkeiten ausser Python selbst
CherryPy	CherryPy is a pythonic, object-oriented HTTP framework	• Konnte sich bereits über zehn Jahre erfolgreich behaupten[77] • • • Integrierter Webserver • Wird von vielen Python Webframeworks als Webserver verwendet
Flask	Web development, one drop at a time	• Grosse Community[78] • Im Einsatz bei einigen der weltweit grössten Websites[79] • Gute Beschreibung von Patterns[80]
Sanic	Build fast. Run fast.	• Modern, Python 3.5+ • Integrierter Async Webserver • Ähnlich wie Flask • Sehr performantes Request-Handling[81] • •
Tornado	Tornado is a Python web framework and asynchronous networking library, originally developed at FriendFeed […]	• Webframework und asynchrone Netzwerkbibliothek • Anzahl Suprozesse kann einfach als Parameter mitgegeben werden (funktioniert nur unter Linux[82])
Pyramid	The Start Small, Finish Big Stay Finsihed Framework	• Kann mit wenig Code beginnen und benötigte Features mit add-ons hinzufügen • Viele Tutorials[83] und Videos

Responder	a familiar HTTP Service Framework for Python	• Integrierter „production" Webserver: Nutzt uvicorn[84] • • • Verfolgt den „ASGI" Ansatz • f-string Syntax Route Deklaration • Backgroundtasks werden in einem ThreadPoolExecutor erstellt • Konzepte von Flask und Falco abgeschaut und vereint[85] • • • Es wird nur Python 3.6+ unterstützt • Baut auf Starlette auf[86] •
Hug	Embrace the APIs of the future	• Von Amazon in „20 Python Libraries You Aren't Using (But Should)" empfohlen[87] • • Web-API's lassen sich mit wenig Code sehr einfach erstellen
Falcon	Falcon is a reliable, high-performance Python web framework for building large-scale app backends and microservices. […]	• Wirbt vor allem mit Geschwindigkeit • Pures REST-Framework • Low-level
FastAPI	FastAPI framework, high performance, easy to learn, fast to code, ready for production	• OpenAPI-Dokumentation lässt sich aus Code generieren • Baut auf Starlette[88] auf • Sehr aktuelle Dokumentation • Wirbt mit „ready for production"-Slogan

Anmerkung. Eigene Darstellung.

Weitere Webframeworks wie CubicWeb [89] wurden nicht berücksichtigt, da eine Python Version grösser oder gleich 2.5 und kleiner als 3 installiert werden muss [90], wodurch dieses nicht mit TensorFlow kompatibel ist. Ebenfalls wurden WebCore [91] und andere ausser Acht gelassen, da diese auf GitHub nicht mehr als hundert Sterne erreicht haben und somit nicht den Status eines soliden Frameworks geniessen. Die Mindestanzahl Sterne wurde analog zu Microsoft's neuem IntelliSense Feature gewählt, bei dem automatisch diejenigen Elemente zuoberst vorgeschlagen werden, die häufig in Open Source-Projekten auf GitHub verwendet werden und von denen jedes Repository mehr als hundert Sterne hat [92]. Ein eleganter Kandidat wäre auch Vibora [93] gewesen. Dieser befindet sich jedoch noch in einer Alpha-Phase und wird momentan komplett neu geschrieben [94].

Full-Stack vs Microframeworks
Die zu evaluierenden Webframeworks wurden in Abbildung 63 grob in zwei Kategorien eingeteilt: Full-Stack und Microframework [95]. Microframeworks werden oft als „Lightweight" bezeichnet und sind im Umfang sehr schlank, jedoch modular erweiterbar. Full-Stack Frameworks bieten out-of-the-box eine Vielzahl an Features wie Object-Relational Mapping (ORM), Session und Cookie-Handling, Template Engine, Request Dispatcher, Form Handling, Authentifizierungsmodule und andere Komponenten an. Die Einteilung ist nicht immer eindeutig, da sich zum Beispiel TurboGears sowohl als Full-Stack wie auch Microframework verkauft [96].

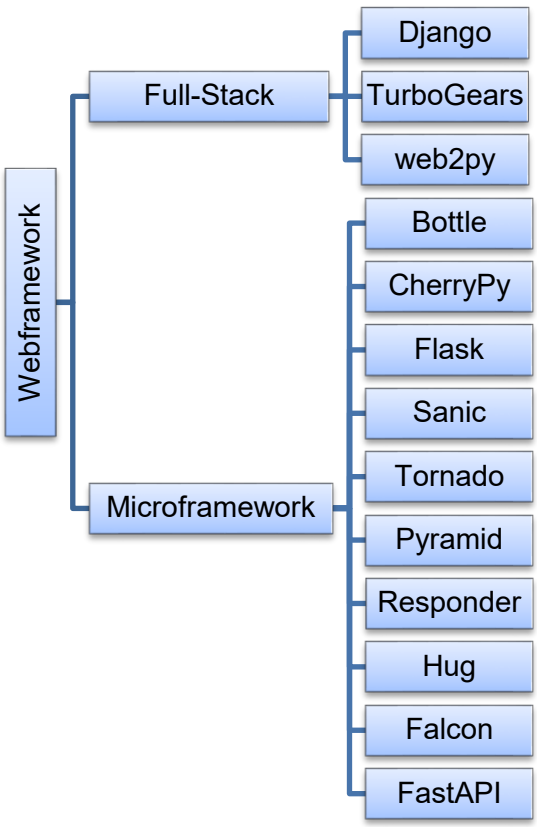


Abbildung 63: Webframework-Aufteilung Full-Stack - Microframework

Anmerkung. Eigene Darstellung.

Deployment-Möglichkeiten
Da weder Anforderungen bezüglich Sicherheit, Skalierbarkeit noch Performanz im Sinne von Anzahl gleichzeitig zu verarbeitende Requests bestehen, reicht ein einfacher built-in HTTP Server aus. Dies ermöglicht ein einfacheres Deployment, da nicht noch komplexe Webserver wie Apache [97] oder nginx [98] konfiguriert werden müssen. Da die Evangelisch-reformierte Kirche Horgen mehrheitlich Windows-Geräte im Einsatz hat, werden in Tabelle 55: Framework-AnalyseTabelle 55 nur diejenigen berücksichtigt, die auch unter Windows lauffähig sind. Cloud-Deployment-Varianten werden aufgrund der Datensicherheit, bzw. Restriktionen seitens der Evangelisch-reformierten Kirche Horgen, nicht in Erwägung gezogen.

Tabelle 55: Framework-Analyse

Framework	Offiziell empfohlen nach Dokumentation	Bemerkungen
Django[99]	Apache + mod_wsgi, uWSGI	uWSGI unterstützt Windows mit Hilfe des cygwin POSIX Emulationssystem. Laut uWSGI scheint die Portierung immer noch „experimental" zu sein.[100]
Pyramid[101]	Apache + mod_wsgi, IIS, über ASGI mit uvicorn[102], uWSGI	Laut Dokumentation macht es Sinn, einen Webserver wie CherryPy oder Twisted in einem Windows Service laufen zu lassen.[103]
TurboGears[104]	Apache + mod_wsgi, Chaussette + Circus	Das Deployment ist aufwändig, da Chaussette und Circus benötigt werden. Chaussette hatte die letzte Neuerung am 19.04.2017 erhalten[105] und scheint somit nicht in aktiver Entwicklung zu sein.
Web2Py[106]	IIS, Apache + mod_wsgi, built-in rocket server + nssm (Windows Service)	Web2Py hat einen integrierten Webserver namens „rocket server", der in einen Windows Services deployed werden kann. Allerdings geschieht dies mit Hilfe eines Service-Helpers namens „nssm", der schon seit 3.8.2017 kein Update erhalten hat und nur von einer Person entwickelt wird[107].
Flask[108]	Apache + mod_wsgi, uWSGI, Twisted[109], Gevent[110], Waitress[111]	Flask empfiehlt Twisted oder Waitress als WSGI-Container zu nehmen oder eine Netzwerkbibliothek wie Gevent.
Bottle[112]	WSGI Server wie Waitress	Bottle bietet „ready-to-use" Adapter für viele WSGI Server an.
CherryPy[113]	Built-in „production-ready" HTTP Server[114]	CherryPy ist eine reine Python-Library, die einen WSGI Server sowie ein minimales Framework enthält.
Sanic[115]	Built-in HTTP Server[116]	Sanic hat einen eingebauten Webserver und unterstützt das in Python 3.5 eingeführte async/await Konzept. Dadurch kann non-blocking Code geschrieben werden.
Tornado[117]	Integrierte, asynchrone Netzwerk-Library	Tornado bietet neben einem Webframework auch eine asynchrone Netzwerk-

		Library an, somit können Long Polling Verbindungen und WebSockets realisiert werden.[118]
Responser[119]	Built-in „production webserver”	Responser verfolgt den modernen ASGI-Ansatz und kommt mit einem integrierten Webserver, basierend auf uvloop. Des Weiteren wird WhiteNoise genutzt, um static files auszuliefern.
Hug[120]	WSGI Server wie uWSGI	Hug empfiehlt einen WSGI-kompatiblen Webserver wie uWSGI zu verwenden.
Falcon[121]	WSGI Server wie Waitress oder uWSGI	Falcon spricht WSGI und listet deshalb Waitress oder uWSGI auf.
FastAPI[122]	Docker Templates vorhanden, ASGI Server wie Uvicorn, Hypercorn	FastAPI stellt zwei Fullstack-Templates[123] [124] zur Verfügung, die auf Docker basieren. Alternativ kann auch ein ASGI-Server wie Uvicorn oder Hypercorn verwendet werden.

Anmerkung. Eigene Darstellung.

Wie in Abbildung 64 zu sehen ist, haben folgende Webframeworks bereits einen integrierten Webserver, der auch in produktiv Umgebungen genutzt werden kann.

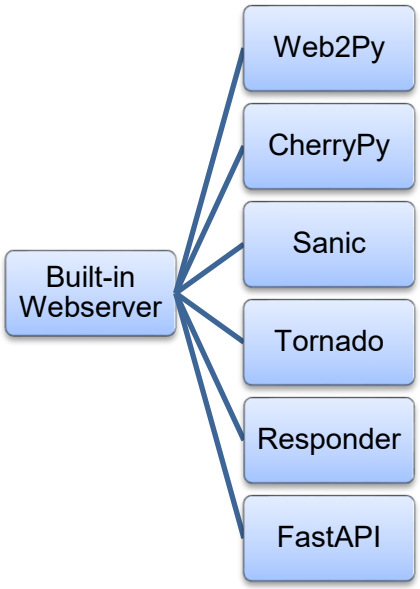


Abbildung 64: Webframeworks mit integriertem Webserver

Anmerkung. Eigene Darstellung.

Anmerkung: Responder und FastAPI verfügen über keinen echten Built-in Webserver. Diese Frameworks unterstützen jedoch den ASGI-Standard. So lassen sich mit lediglich einer Zeile Code ein Webserver wie Uvicorn [125] oder Hypercorn [126] integrieren.

Dokumentation
Für die Entwicklung ist eine umfangreiche Dokumentation relevant. In Tabelle 56 wurden die Anzahl Seiten festgehalten.

Tabelle 56: Webframework - Anzahl Seiten der Dokumentation

Webframework	Dokumentation PDF in Anzahl Seiten
Django[127]	1894
TurboGears[128]	241
Web2Py[129]	614
Bottle[130]	105
CherryPy[131]	215
Flask[132]	344
Sanic[133]	126
Tornado[134]	241
Pyramid[135]	1239
Responder[136]	37
Hug[120]	(lediglich online)
Falcon[137]	176
FastAPI[138]	(lediglich online)

Anmerkung. Eigene Darstellung.

Es muss jedoch festgehalten werden, dass andere Faktoren wie Aktualität der Dokumentation genauso relevant sind. Ebenfalls können keine Schlüsse zwischen dem Umfang in Anzahl Seiten und Qualität gezogen werden. Tabelle 56 soll deshalb lediglich ein Grössengefühl vermitteln. Grafisch lässt sich dies in Abbildung 65 gut betrachten.

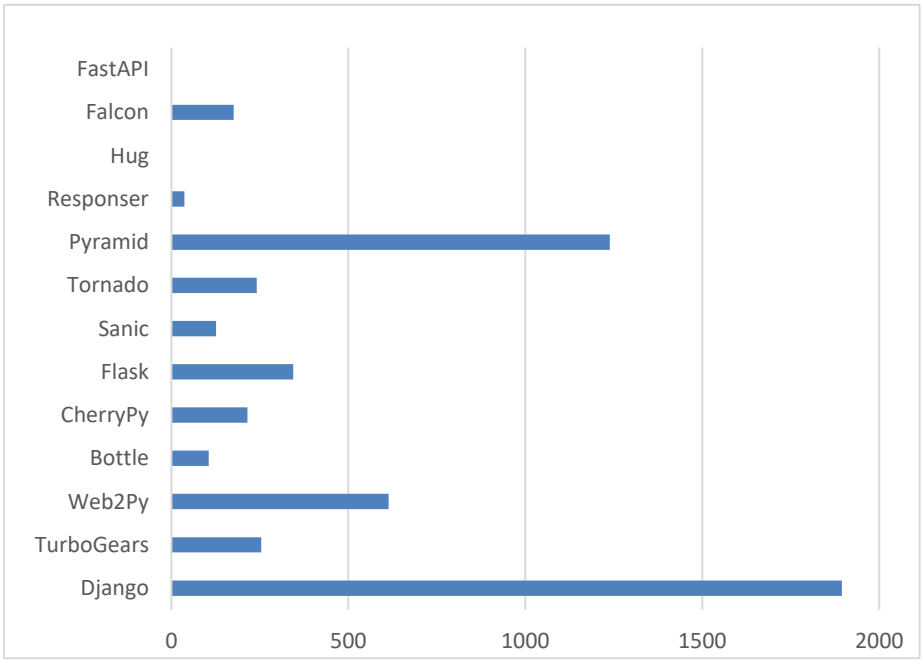


Abbildung 65: Webframework Dokumentation Grafisch

Anmerkung. Eigene Darstellung.

Django scheint die umfangreichste Dokumentation zu haben, mit einem Abstand von über 500 Seiten gegenüber Pyramid. Danach wird wieder ein Sprung von 600 Seiten zu Web2Py gemacht.

Community
Interessant zu sehen, ist immer auch wie etabliert ein Framework in der Community ist. Untersucht wurden deshalb die Anzahl Tags auf Stack Overflow, die Anzahl an Sterne und Mitwirkende auf GitHub. Dies wird in Tabelle 57 festgehalten.

Tabelle 57: Vergleich der Communities

Webframework	Stack Overflow Tags	GitHub Sterne	GitHub Mitwirkende
Django	191689	39833	1706
TurboGears	255[2]	277	26
Web2Py	2045	1711	169
Bottle	1317	6016	154
CherryPy	1257	913	97
Flask	26449	42300	503
Sanic	75	11509	199
Tornado	3322	17333	301
Pyramid	2075	3062	261
Responder	0[3]	2847	59
Hug	23	5595	88
Falcon	68	6284	119
FastAPI	0[4]	1912	22

Anmerkung. Eigene Darstellung.

In Abbildung 66 sind die drei Community-Kategorien veranschaulicht (Stand 3. März 2019).

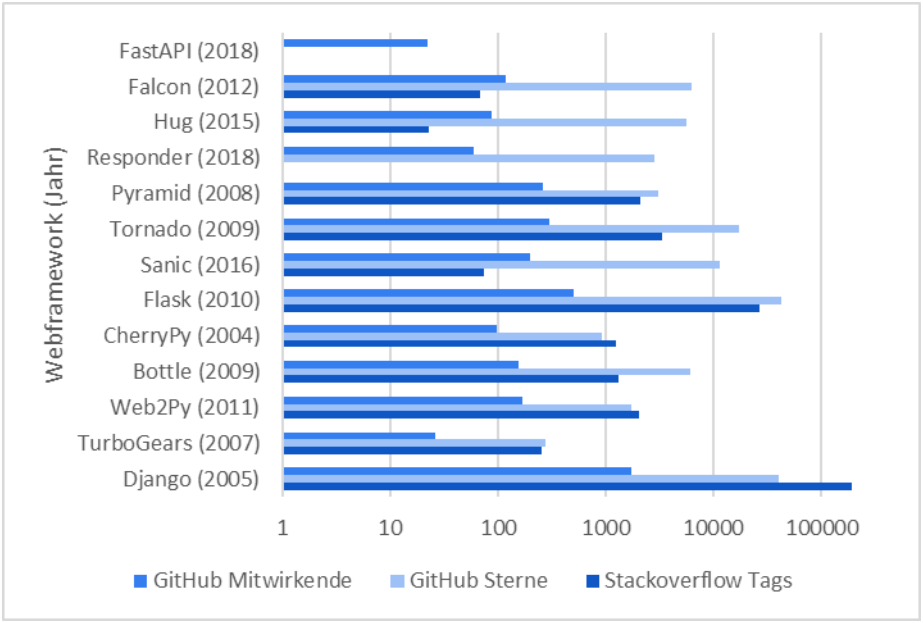


Abbildung 66: Evaluation Webframework, Kategorie Community

Anmerkung. Eigene Darstellung.

[2] 108 von turbogears + 147 von turbogears2
[3] Noch kein eigener Stackoverflow Tag vorhanden
[4] Noch kein eigener Stackoverflow Tag vorhanden

Django existiert bereits seit 2005 und konnte sich stark etablieren. Nur Flask weist noch eine grössere Popularität, gemessen an Anzahl GitHub Sternen, auf. Eine beachtliche Beliebtheit geniessen auch die Newcomer Responder, Sanic und FastAPI. Im Gegenzug gibt es bis dato für Responder und FastAPI noch keine Fragen auf Stack Overflow.

Dokumentation für Upload und Download von Bildern
Eine Kernfunktionalität der ImageFinder Software ist das Hochladen von Bildern. Deshalb wird Wert daraufgelegt, dass dieses gut vom Framework dokumentiert und unterstützt wird. In Tabelle 58 wird näher auf die Dokumentation eines File-Uploads eingegangen.

Tabelle 58: Vergleich Dokumentation für Upload und Download von Bildern

Web-framework	File-Upload Dokumentation	Beschreibung
Django	https://docs.djangoproject.com/en/2.1/topics/http/file-uploads/	Dokumentation beschreibt, wie man Files entgegennehmen kann, die über eine HTML Form gesendet wurden.
TurboGears	http://www.turbogears.org/1.0/docs/FileUploadTutorial.html	Ein etwas komplexeres Tutorial mit Anbindung an eine Datenbank.
Web2Py	http://www.web2py.com/books/default/chapter/29/07/forms-and-validators#Storing-the-original-filename	Es wird zwar von einem Fileupload gesprochen. Der zur Verfügung gestellte Code ist allerdings sehr schlecht leserlich. Es wird Code vermischt, der für Datenbankzugriffe sowie Darstellung der Benutzeroberfläche zuständig ist.
Bottle	https://bottlepy.org/docs/dev/tutorial.html	In der Section „FILE UPLOADS" ist ein einfaches Biespiel zu finden.
CherryPy	http://docs.cherrypy.org/en/latest/_modules/cherrypy/tutorial/tut09_files.html	Ein kurzes Beispiel deutet auf eine sehr low-level Handhabung hin. Ein File wird innerhalb einer Endlosschleife in Stücke gelesen.
Flask	http://flask.pocoo.org/docs/1.0/patterns/fileuploads/	Einfache Anleitung, die genau auf das Hochladen von Files zielt.
Sanic	-	Keine Angabe in der offiziellen Dokumentation auffindbar.
Tornado	https://github.com/tornadoweb/tornado/blob/master/demos/file_upload/file_receiver.py	Auf GitHub ist ein offizielles Beispiel zu finden, wo mehrere Files hochgeladen werden.
Pyramid	https://docs.pylonsproject.org/projects/pyramid-cookbook/en/latest/forms/file_uploads.html	Pyramid zeigt einfachen Code, der sich um die Entgegennahme des Requests kümmert und ein File permanent auf die Festplatte speichert.
Responder	https://media.readthedocs.org/pdf/responder/latest/responder.pdf	In der Responder-Dokumentation ist ein Beispiel im Kapitel 3.1.9 zu finden. Dieses zielt auf die Entgegennahme von „Data" und könnte so gut wie

		alles sein. Auffallend ist, dass der Router mit einem Decorator @api.background.task geschmückt ist, der automatisch die Arbeit im Hintergrund weiter führt, während dem Client sofort eine Response gegeben werden kann.
Hug	https://github.com/timothycrosley/hug/blob/develop/examples/file_upload_example.py	Das gezeigte Beispiel ist nicht sehr intuitive und zeigt lediglich den Zugriff auf den Namen und die Länge des Files, jedoch nicht, wie das File letztendlich gespeichert werden kann.
Falcon	https://falcon.readthedocs.io/en/stable/user/tutorial.html	Tutorial für einen einfachen Image Sharing Service
FastAPI	https://fastapi.tiangolo.com/tutorial/request-files/	Exzellentes (asynchrones) Beispiel, dass den gesamten Code zeigt, der für einen Fileupload benötigt wird

Anmerkung. Eigene Darstellung.

Datenbankanbindung und ORM

Damit zu den Bildern der Evangelisch-reformierte Kirche Horgen jeweils Metadaten wie zum Beispiel Fotograf und Datum gespeichert und später abgefragt werden können, erscheint den Autoren eine Anbindung an eine Datenbank sinnvoll. Da jedoch nur sehr einfache Abfragen getätigt werden müssen, wird nicht zwingend ein Object-Relational-Mapping benötigt. Generell kann angenommen werden, dass heutzutage eine API ohne CRUD-Operationen kaum vorzufinden ist und dies alle Webframeworks in irgendeiner Form unterstützen. Es wird daher nicht weiter auf dieses Kriterium eingegangen.

Reife der Webframeworks
Die nachfolgenden Daten sind von einem Stand vom 3. März 2019. In Tabelle 59 wird festgehalten, seit wann das Framework existiert, bzw. wann der erste Commit auf GitHub erfolgte. Auch kann gesehen werden, ob das Framework noch unter aktiver Entwicklung steht und wie gross es mittlerweile geworden ist. Dabei wird lediglich der benötigte Speicherplatz des von GitHub heruntergeladenen Source Codes auf der Festplatte gemessen.

Tabelle 59: Evaluation Webframework: Reife

Web-framework	Erster GitHub Commit	ID des ersten Commits	Neuster GitHub Commit	Grösse auf Festplatte [MB]
Django	13.07.2005	d6ded0e91b	1.03.2019	237.00
Turbo-Gears	27.06.2007	662968e	22.02.2019	7.07
Web2Py	23.11.2011	d421c132	3.03.2019	59.00
Bottle	30.06.2009	f302723	5.01.2019	8.58
CherryPy	20.11.2004	90892be9	2.03.2019	29.30
Flask	6.04.2010	33850c0e	24.02.2019	8.61
Sanic	26.05.2016	60f1004	3.03.2019	3.05
Tornado	09.09.2009	11de50a8	3.03.2019	11.30
Pyramid	04.07.2008	612d74784	19.02.2019	32.20
Responder	08.10.2018	e84f851	3.03.2019	31.10
Hug	17.06.2015	78cac2a	16.02.2019	8.63
Falcon	6.12.2012	d0d6d18	2.03.2019	8.20
FastAPI	5.12.2018	406c092	3.03.2019	4.76

Anmerkung. Eigene Darstellung.

Mit 237 MB Grösse ist Django beinahe 4-mal grösser als Web2Py. Im Mittelfeld befinden sich Pyramid mit einer Grösse von 32.2 MB und CherryPy mit 29.3 MB. Abbildung 67 zeigt dies grafisch anhand den Metriken: Erster GitHub Commit, bzw. Erscheinungsjahr (x-Achse), GitHub Sterne (y-Achse) und Grösse auf der Festplatte in Megabyte (Grösse der Blase) an.

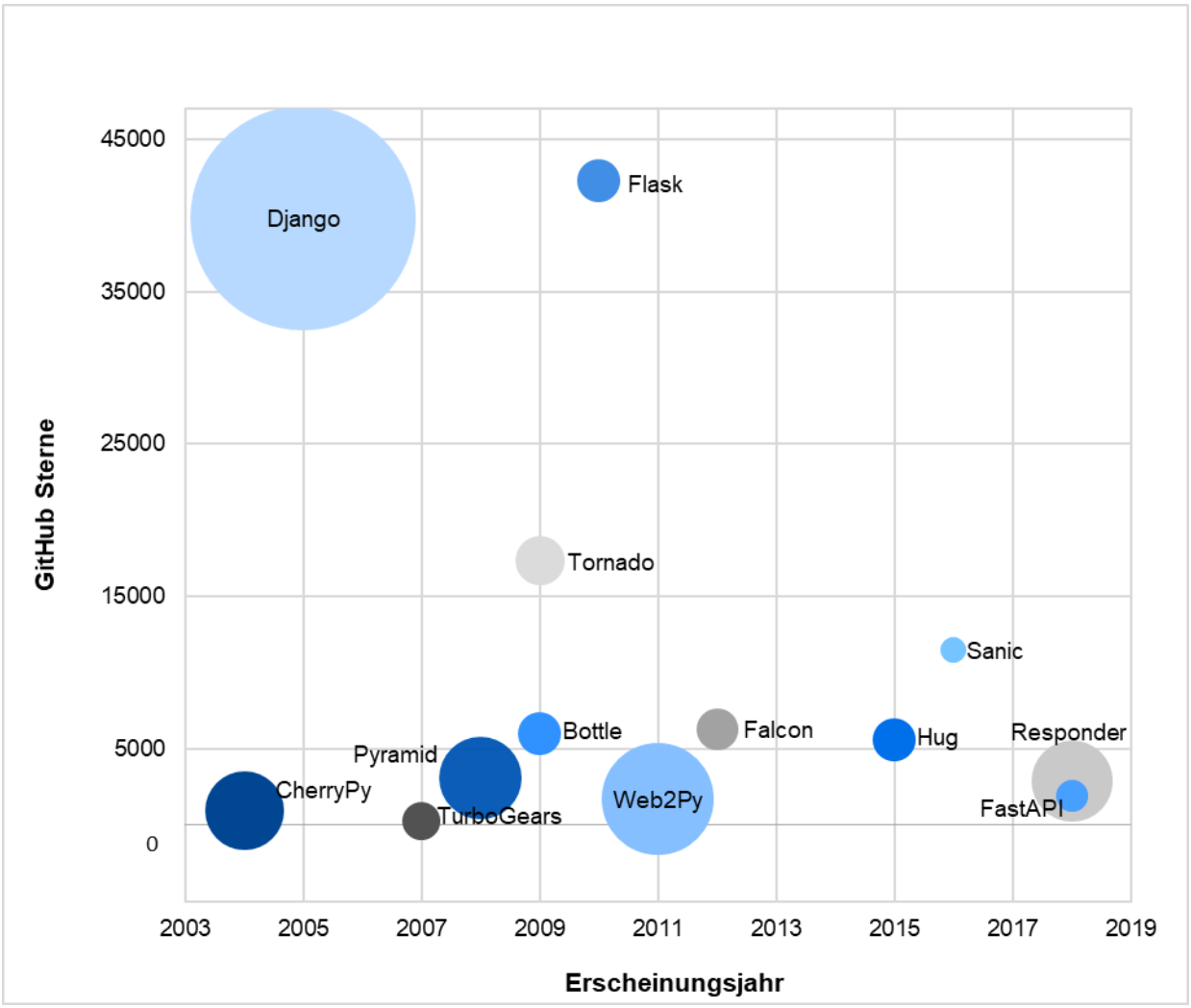


Abbildung 67: Übersicht über Python Webframeworks, Metriken: GitHub Sterne, Erscheinungsjahr, Grösse in MB

Anmerkung. Eigene Darstellung.

Nutzwertanalyse Backend Webframework
Für die einzelnen Webframeworks wurde eine Nutzwertanalyse mit den Anforderungen für den ImageFinder durchgeführt. Die Ergebnisse sind in Tabelle 60 und Abbildung 68 ersichtlich.

Tabelle 60: Vergleich Nutzwertanalyse Backend Webframework

Kriterium	**Gewich-tung in %**	**Django**		**TurboGears**		**Web2py**		**Flask**		**Bottle**	
		N	P	N	P	N	P	N	P	N	P
Built-in Server	25	1	0.25	1	0.25	5	1.25	1	0.25	1	0.25
Dokumentation Allgemein	15	6	0.9	5	0.75	5	0.75	5.5	0.825	4.5	0.675
Dokumentation Fileupload	10	5	0.5	4.5	0.45	4	0.4	5	0.5	4.5	0.45
Gesamteindruck	20	5.5	1.1	4	0.8	4	0.8	6	1.2	5	1
Stack Overflow Tags	10	6	0.6	4	0.4	5	0.5	6	0.6	5	0.5
GitHub Sterne	20	6	1.2	4.5	0.9	5	1	6	1.2	5.5	1.1
Total	**100**		**4.55**		**3.55**		**4.7**		**4.575**		**3.975**

Kriterium	Gewich-tung in %	CherryPy		Sanic		Tornado		Pyramid		Responder	
		N	P	N	P	N	P	N	P	N	P
Built-in Server	25	5.5	1.375	6	1.5	5.5	1.375	1	0.25	6	1.5
Dokumentation Allgemein	15	4.5	0.675	5	0.75	4.5	0.675	5.5	0.825	4	0.6
Dokumentation Fileupload	10	4.5	0.45	1	0.1	6	0.6	5	0.5	5	0.5
Gesamteindruck	20	5	1	5.5	1.1	4.5	0.9	5	1	5.5	1.1
Stack Overflow Tags	10	5	0.5	4	0.4	5.5	0.55	5	0.5	1	0.1
GitHub Sterne	20	4.5	0.9	6	1.2	6	1.2	5	1	5	1
Total	**100**		**4.9**		**5.05**		**5.3**		**4.075**		**4.8**

Kriterium	Gewichtung in %	Hug		Falcon		FastAPI	
		N	P	N	P	N	P
Built-in Server	25	1	0.25	1	0.25	6	1.5
Dokumentation Allgemein	15	4	0.6	5	0.75	6	0.9
Dokumentation Fileupload	10	4	0.4	6	0.6	6	0.6
Gesamteindruck	20	5	1	5	1	5.5	1.1
Stack Overflow Tags	10	4	0.4	4	0.4	1	0.1
GitHub Sterne	20	5.5	1.1	5.5	1.1	5	1
Total	**100**		**3.75**		**4.1**		**5.2**

Anmerkung. Eigene Darstellung.

Auf zwei Nachkommastellen gerundet

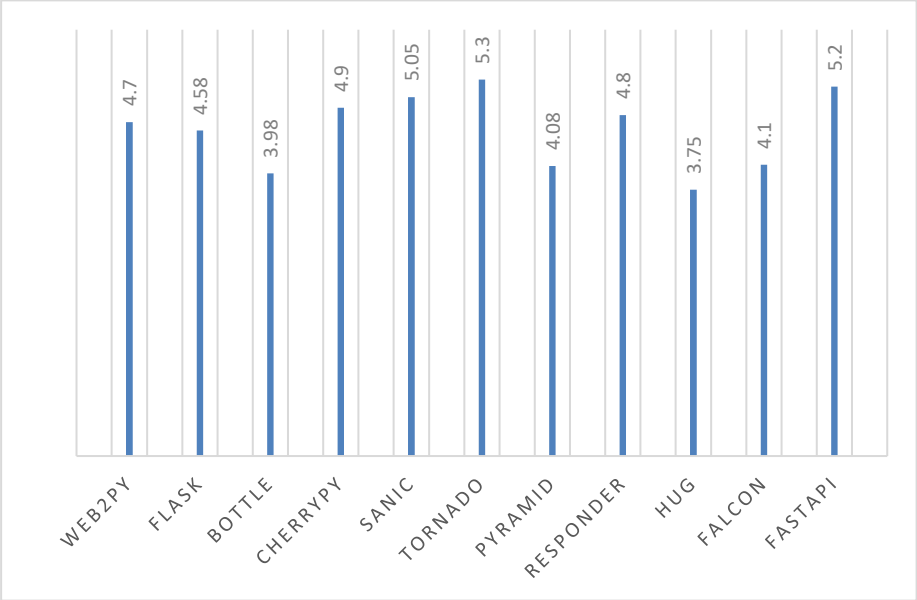


Abbildung 68: Nutzwertanalyse grafisch

Anmerkung. Eigene Darstellung.

Top 5 Hello World

Anhand einer Hello World Applikation kann verglichen werden, wie die einzelnen Frameworks aufgebaut sind. Zur Illustration wurden fünf Hello World Beispiele nachfolgend aufgelistet.

Tornado

```
from abc import ABC

import tornado.ioloop
import tornado.web


class MainHandler(tornado.web.RequestHandler, ABC):
    def get(self):
        self.write("Hello, world")


def make_app():
    return tornado.web.Application([
        (r"/", MainHandler),
    ])


if __name__ == "__main__":
    app = make_app()
    app.listen(8888)
    tornado.ioloop.IOLoop.current().start()
```

Routing

```
class MainHandler(tornado.web.RequestHandler, ABC):

    # http://127.0.0.1:8888/101
    def post(self, *args, **kwargs):
        param_id = args[0]
        self.write(param_id)


def make_app():
    return tornado.web.Application([
        (r"/(\d+)$", MainHandler),
    ])
```

FastAPI

```
import uvicorn as uvicorn
from fastapi import FastAPI

app = FastAPI()


@app.get("/")
async def read_root():
    return {"Hello": "World"}


if __name__ == '__main__':
    uvicorn.run(app)
```

Routing

```
@app.get("/items/{item_id}")
async def read_item(item_id: int, q: str = None):
    return {"item_id": item_id, "q": q}
```

FileUpload

```
@app.post("/uploadfile/")
async def create_upload_file(file: UploadFile = File(...)):
    return {"filename": file.filename}
```

Sanic

```
from sanic import Sanic
from sanic.response import json

app = Sanic()


@app.route("/")
async def test(request):
    return json({"hello": "world"})

if __name__ == "__main__":
    app.run(host="127.0.0.1", port=8000)
```

Routing

```
@app.route('/tag/<tag>')
async def tag_handler(request, tag):
    return text('Tag - {}'.format(tag))
```

File Upload

```
@app.route('/upload', methods=['POST'])
async def omo(request):
    images = request.files.getlist('images')

    for image in images:
        print(image.body)
```

CherryPy

```
import cherrypy


class HelloWorld(object):
    @cherrypy.expose
    def index(self):
        return "Hello World!"


cherrypy.quickstart(HelloWorld())
```

Routing

```
@cherrypy.popargs('year', 'month')
class Blog:

    @cherrypy.expose
    def index(self, year=None, month=None):
        return f"Year: {year}, Month: {month}"
```

Fileupload

```
@cherrypy.expose
def upload(self, files):

    for file in files:
        data = file.file.read()
        print(f'{data}')
```

Responder

```
import responder

api = responder.API()


@api.route("/{greeting}")
async def greet_world(req, resp, *, greeting):
    resp.text = f"{greeting}, world!"


if __name__ == '__main__':
    api.run()
```

Routing

```
@api.route("/hello/{who}")
def hello_to(req, resp, *, who):
    resp.text = f"hello, {who}!"
```

FileUpload

```
@api.route("/")
async def upload_file(req, resp):
    @api.background.task
    def process_data(data):
        f = open('./{}'.format(data['file']['filename']), 'w')
        f.write(data['file']['content'].decode('utf-8'))
        f.close()

    data = await req.media(format='files')
    process_data(data)

    resp.media = {'success': 'ok'}
```

Anmerkung 12. April 2019: Nach einem Update-Release von Sanic auf Version 19.3.1 funktioniert das Multiprocessing unter Windows 10 nicht mehr. Diesbezüglich wurde mittels einem Issue auf GitHub aufmerksam gemacht. [139] Sanic hat offiziell bekannt gegeben, Windows nicht, bzw. „experimental" und nach „best-effort" zu unterstützen.
[140] Das Sanic-Framework unterstützt noch nicht den ASGI-Standard. Einige Bemühungen sind diesbezüglich im Gange. [141]Der Server ist deshalb tief in das Framework verwurzelt und lässt sich nicht austauschen. Die Autoren haben sich deshalb entschlossen, Sanic durch ein anderes Framework zu ersetzen.

Im Kontext des Webframworks für die Software ImageFinder der Evangelisch-reformierte Kirche Horgen,

um den Nicht-funktionale Anforderung „NFR13 - Webapplikation" und „NFR8 - Win 10" sowie NFR5, NFR6 und NFR7 aus der Kategorie Performance entgegen zu treten,

haben sich die Autoren für den Einsatz des schlanken FastAPI-Frameworks entschieden und andere Webframeworks wie Tornado, Sanic, CherryPy und Responder verworfen,

damit die Webapplikation auf Windows 10 installiert werden kann, eine out-of-the-box OpenApi-Dokumentation zur Verfügung steht sowie der zukunftssichere ASGI-Standard unterstützt wird,

nehmen dafür aber in Kauf, dass es sich hier um noch ein sehr junges Framework handelt, das hauptsächlich von einem Hauptentwickler gewartet wird.

Anhang H Evaluation Frontend Framework

Für das Frontend sollte ein Webframework verwendet werden. Dabei wurden die Erfahrungen der Autoren in die Evaluation miteinbezogen. Folglich wurden vor allem Frameworks in Betracht gezogen, mit welchen die Autoren bereits Erfahrungen gesammelt haben. In Abbildung 69 sind diese Frameworks aufgelistet.

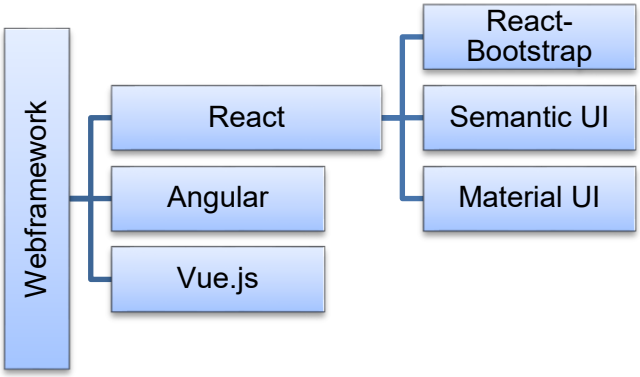


Abbildung 69: Vergleich der Webframeworks nach Technologie

Anmerkung. Eigene Darstellung.

Für die Auswahl des geeigneten Frameworks wurden Kriterien definiert, welche später bei der Implementation nützlich sein könnten. Diese werden nachfolgend aufgelistet:

- Drag and Drop
- Infinite Scrolling
- Slider
- MVVM
- Server-seitiges Rendern
- HTML-based Template Syntax
- Grundkomponenten wie Buttons, Listen, ect.
- Keine Beschränkung auf Single-Page-Application

Die Recherche zeigte, dass sämtliche Frameworks die genannten Punkte erfüllen. Folglich wurden weitere Abklärungen getroffen, um das geeignetste Framework festzulegen. Dazu wurde die die Internetseite dzone.com [142] konsultiert, welche die gängigen Frameworks untereinander vergleicht.

Besonderheiten
In Tabelle 61 wurden Besonderheiten der Frameworks gesammelt. Dabei zeigte sich, dass Angular für unseren Anwendungszweck zu umfangreich war, weshalb es von der weiteren Evaluation ausgeschlossen wurde.

Tabelle 61: Besonderheiten der Webframeworks

Frontend Framework	Besonderheiten
React Frameworks	• Wiederverwendbare Komponenten • Methoden und Markup gehören zusammen • Nutzt neuste Features von JavaScript • Fördert pure function • Server Side Rendering

	• Unterstützt Virtual DOM • Unterstützt Template Enginge JSX • Einfache Testmöglichkeiten • Flux / Redux Architektur
React-Bootstrap	• Responsive web design support • Ausführliche Dokumentation • Grosse Anzahl an HTML Klassen und DOM Elementen • Einfache Entwicklung • Nutzt Vorteile von React (wiederverwendbare Komponenten)
Semantic-UI	• Simpel • Kleine File Grössen und schnelle Ladezeiten • Vielseitige Elemente • Begrentzte Möglichkeiten
Material UI	• Bekanntestes Framework für die Umsetzung von Google's Material Design • Hohe Anpassungsfähigkeit • React Kenntnisse von Vorteil
Angular	• Für grosse Projekte gedacht • Vielzahl an Komponenten • Hohe Modularität • Hohe Komplexität • Typescript muss verwendet werden
Vue.js	• Ausführliche Dokumentation • Einfaches Konzept • HTML based Template Syntax • Hohe Flexibilität • Hohe Performanz • Tooling • Aktive Community • Vuex Architektur

Anmerkung. Eigene Darstellung.

Obwohl React basierte Frameworks und Vue.js sich in einigen Punkten ähnlich sind, wie zum Beispiel dem Arbeiten mit root libraries, das Virtual DOM Modell oder die auf Komponenten basierte Struktur, gibt es trotzdem Unterschiede. In Tabelle 62 wurden die Kerneigenschaften, welche die Frameworks unterscheiden aufgelistet und bewertet. Nebst den Erfahrungen der Autoren wurde dabei auf die Webseite www.dzone.com [142] zurückgegriffen.

Tabelle 62: Webframeworks Kerneigenschaften

Web Framwork	**React-Bootstrap**	**Semantic-UI**	**Material-UI**	**Vue.js**
Qualität der Dokumentation von 1 bis 6	5	4	5	6
Lernkurve	3	3	3	6
Skalierbarkeit	6	6	6	4
Rendering	4	4	4	6
State-Management	5	5	5	6
Summe	**23**	**22**	**23**	**26**

Anmerkung. Eigene Darstellung.

Community
Anderst als bei der Evaluation des Backend Frameworks sind, wie in Tabelle 63 zu sehen, die Communities bei allen Frameworks für das Frontend sehr aktiv.

Tabelle 63: Community Frontend-Framework (Stand 3. März 2019

Webframework	Stackoverflow Tags	GitHub Sterne	GitHub Mitwirkende
React	126417	129669	1285
React-Bootstrap	1180	14904	236
Semantic-UI	2108	44658	190
Material-UI	3754	44863	1148
Vue	30918	129655	264

Anhang I Installationsanleitung

Windows Service
Der ImageFinder sollte als Windows Service beim Bootvorgang gestartet werden. Für die Integration wurde das Freeware Tool NSSM verwendet. [143] Das Tool garantiert, dass der Service korrekt gestartet wird und kann problemlos in die Dienste von Windows 10 integriert werden. Nachfolgend ist eine detaillierte Installationsanleitung aufgeführt.

Installation as a Windows Service

TobiHSR edited this page now · 9 revisions

Description

The following instructions describe how to run the application as a windows service.

Prerequirements

You should have pipenv and npm installed.

Download

Download the files from this repository and extract them to a local folder, or just clone the repository from the repository.

Requirements

You will need to have: 1. pipenv 2. [npm] (https://www.npmjs.com/)

installed to build this software.

Create .exe file

Install all dependencies with

```
1. `$ pipenv install`

2. `$ npm install`
```

Run the script `create_windows_service_exe.ps1` provided in the scripts folder. Pyinstaller creates a folder `/dist/ImageFinder-WinService`. You can copy the folder `ImageFinder-WinService` to a destination of your choice. In the folder, there should be the executable "ImageFinder.exe".

Bind your Google API account

Follow the instructions on this GitHub page to prepare your Google API account. After that, create a file `.secrets.toml` and store it in the root folder of the previously created project. The file content should look as such:

google_developer_key = "[your key]" google_custom_search_cx = "[your custom cx]"

Create windows service

Download the NSSM application from their web page and start the application by running `win64/nssm.exe` in a PowerShell with administrator priviledges.

Install the nssm service by running:

./nssm.exe install

In the GUI, add the path to ImageFinder.exe and make sure the service runs automatically.

Start the service

You should find the service in the Windows Services, and you can start it there immediately. The service should also run when you start your computer. The first time you run the application, Imagenet will be loaded and TensorFlow will prepare all services. This will take some time. After that, you can start your browser and go to the homepage. You should then see the dashboard.

Anhang J Usability-Tests

Es wurden drei Usability Tests durchgeführt. Die Testpersonen haben keine Informatikausbildung, arbeiten jedoch regelmässig am Computer, meist mit Office-Anwendungen.

Usability Tests ImageFinder

Datum:	2.5.2019
Auftraggeber:	Reformierte Kirche Horgen
Projektleiter:	Tobias Saladin, Martin Odermatt
Version:	1.0
Klassifizierung:	Nicht klassifiziert
Testperson	Nadine Saladin

Inhaltsverzeichnis

1 Aufgaben

Nr	Aufgabe	Erwartung
1	Du arbeitest das erste Mal mit dem ImageFinder und möchtests als erstes deine Bilder hochladen. Deine Bilder sollen dabei alle ein Tag und einen Autor erhalten.	Bilder hinzufügen Erstellen neues Tag Erstellen neuer Autor Ordner importieren
2	Du hast deine Bilder eingefügt und kannst diese in der Übersicht sehen. Du möchtest nun einem Bild ein weiteres Tag vergeben.	Bild markieren / Klick auf Plus Markierte Bilder bearbeiten Erstellen neues Tag Speichern
3	Wechsle bei 5 Bildern gleichzeitig den Autor.	5 Bilder markieren Markierte Bilder bearbeiten Erstellen neuer Autor Speichern
3	Nun Möchtest du eine Bildersuche starten. Die Merkmale, die für dich relevant sind befinden sich aber auf zwei verschiedenen Bildern. Starte eine Suche mit beiden Bildern.	Markieren der beiden Bilder Suche starten
4	Filtere die Ergebnisse nach einem der beiden Autoren.	Klick auf Filtersymbol Auswählen Autor Speichern
5	Setze den Filter wieder zurück	Klick auf Filtersymbol Zurücksetzen
6	Wähle ein Bild deiner Wahl aus der Übersicht. Finde heraus wann das Bild gemacht wurde. Filtere alle Ergebnisse nach diesem Datum.	Klick auf Plus bei Bild Abbrechen Klick auf Filtersymbol Eingabe von Datum Eingabe bis Datum Filter anwenden
7	Suche Bilder mit Hilfe eines Schlagwortes	Klick Google Suche Textfeld Eingabe Schlagwort Enter

2 Ergebnisse

	Erfüllt	Bemerkung
1		
2	ja	
3	ja	Der Anwendungsfall scheint nicht ganz klar. Gemäss der Testperson sei eine Suche mit mehreren Referenzbildern eher für eine Verstärkung eines Merkmales.
4	ja	
5	ja	
6	nein	Lediglich Datum von eingetragen, Datum mit Punkt eingetragen
7	ja	

3 Konsequenzen

- Beim Datum wurde zusätzlich eine „von“ und „bis“ Beschriftung hinzugefügt.

Usability Tests ImageFinder

Datum:	15.5.2019
Auftraggeber:	Reformierte Kirche Horgen
Projektleiter:	Tobias Saladin, Martin Odermatt
Version:	1.0
Klassifizierung:	Nicht klassifiziert
Testperson	Laima Masaleviciute

Inhaltsverzeichnis

1 Aufgaben

Nr	Aufgabe	Erwartung
1	Du arbeitest das erste Mal mit dem ImageFinder und möchtests als erstes deine Bilder hochladen. Deine Bilder sollen dabei alle ein Tag und einen Autor erhalten.	Bilder hinzufügen Erstellen neues Tag Erstellen neuer Autor Ordner importieren
2	Du hast deine Bilder eingefügt und kannst diese in der Übersicht sehen. Du möchtest nun einem Bild ein weiteres Tag vergeben.	Bild markieren / Klick auf Plus Markierte Bilder bearbeiten Erstellen neues Tag Speichern
3	Wechsle bei 5 Bildern gleichzeitig den Autor.	5 Bilder markieren Markierte Bilder bearbeiten Erstellen neuer Autor Speichern
3	Nun Möchtest du eine Bildersuche starten. Die Merkmale, die für dich relevant sind befinden sich aber auf zwei verschiedenen Bildern. Starte eine Suche mit beiden Bildern.	Markieren der beiden Bilder Suche starten
4	Filtere die Ergebnisse nach einem der beiden Autoren.	Klick auf Filtersymbol Auswählen Autor Speichern
5	Setze den Filter wieder zurück	Klick auf Filtersymbol Zurücksetzen
6	Wähle ein Bild deiner Wahl aus der Übersicht. Finde heraus wann das Bild gemacht wurde. Filtere alle Ergebnisse nach diesem Datum.	Klick auf Plus bei Bild Abbrechen Klick auf Filtersymbol Eingabe von Datum Eingabe bis Datum Filter anwenden
7	Suche Bilder mit Hilfe eines Schlagwortes	Klick Google Suche Textfeld Eingabe Schlagwort Enter

2 Ergebnisse

	Erfüllt	Bemerkung
1		
2	JA	
3	JA	Wenn nach einem Merkmal gesucht wird, beispielsweise „Affe“, erscheinen keine geeigneten Treffer.
4	JA	
5	JA	
6	JA	Anwählen des Datums ist umständlich. Besser wäre die Eingabe mit Tastatur.
7	JA	

3 Konsequenzen

- Wenn in der Bildersammlung keine geeigneten Bilder vorhanden sind, werden die ähnlichsten Bilder angezeigt. Dies auch wenn diese das Merkmal nicht enthalten. Eine Änderung dieser Funktionalität ist nicht vorgesehen.
- Die besten Treffer werden gruppiert angezeigt, auch in der ersten Zeile. Das Aufsplitten der ersten Zeile würde dem User zumindest die vier besten Treffer anzeigen.
- Die Eingabe des Datums mit der Tastatur wird umgesetzt.

Usability Tests ImageFinder

Datum:	28.5.2019
Auftraggeber:	Reformierte Kirche Horgen
Projektleiter:	Tobias Saladin, Martin Odermatt
Version:	1.0
Klassifizierung:	Nicht klassifiziert
Testperson	Tiana Limberger

Inhaltsverzeichnis

1 Aufgaben

Nr	Aufgabe	Erwartung
1	Du arbeitest das erste Mal mit dem ImageFinder und möchtests als erstes deine Bilder hochladen. Deine Bilder sollen dabei alle ein Tag und einen Autor erhalten.	Bilder hinzufügen Erstellen neues Tag Erstellen neuer Autor Ordner importieren
2	Du hast deine Bilder eingefügt und kannst diese in der Übersicht sehen. Du möchtest nun einem Bild ein weiteres Tag vergeben.	Bild markieren / Klick auf Plus Markierte Bilder bearbeiten Erstellen neues Tag Speichern
3	Wechsle bei 5 Bildern gleichzeitig den Autor.	5 Bilder markieren Markierte Bilder bearbeiten Erstellen neuer Autor Speichern
3	Nun Möchtest du eine Bildersuche starten. Die Merkmale, die für dich relevant sind befinden sich aber auf zwei verschiedenen Bildern. Starte eine Suche mit beiden Bildern.	Markieren der beiden Bilder Suche starten
4	Filtere die Ergebnisse nach einem der beiden Autoren.	Klick auf Filtersymbol Auswählen Autor Speichern
5	Setze den Filter wieder zurück	Klick auf Filtersymbol Zurücksetzen
6	Wähle ein Bild deiner Wahl aus der Übersicht. Finde heraus wann das Bild gemacht wurde. Filtere alle Ergebnisse nach diesem Datum.	Klick auf Plus bei Bild Abbrechen Klick auf Filtersymbol Eingabe von Datum Eingabe bis Datum Filter anwenden
7	Suche Bilder mit Hilfe eines Schlagwortes	Klick Google Suche Textfeld Eingabe Schlagwort Enter

2 Ergebnisse

1	Erfüllt	Bemerkung
2	JA	
3	JA	
4	JA	
5	JA	
6	NEIN	Die Testperson gab an, dass beim Filtern nach einem Datum nicht alle Bilder angezeigt würden.
7	JA	

3 Konsequenzen

- Wenn eine Suche durchgeführt wird und anschliessend nach zum Beispiel einem Datum gefiltert wird, werden nicht alle Bilder zu diesem Datum angezeigt. Es erscheinen nur diese, welche ähnlich zum Referenzbild sind. Möchte man trotzdem wieder alle Bilder anzeigen, muss der Browser mit F5 aktualisiert werden. Um dies zu verhindern soll mit dem Logo das der Browser aktualisiert werden. Danach können mit dem Filter alle Treffer aus der Datenbank gefunden werden.

Anhang K Abnahmeprotokoll

Release
Abnahmeprotokoll
ImageFinder

Datum:	28.5.2019
Auftraggeber:	Reformierte Kirche Horgen
Projektleiter:	Tobias Saladin, Martin Odermatt
Version:	1.0
Klassifizierung:	Nicht klassifiziert

Inhaltsverzeichnis

1 Abnahmegegenstand

Abnahmegegenstand	Beschreibung
Software ImageFinder	Anwendung installiert am Arbeitsplatz von Frau Tiana Limberger

2 Abnahmebeteiligte

Rolle	Name
Auftraggeber / Kunde	Reformierte Kirche Horgen / Tiana Limberger
Projektleiter	Tobias Saladin, Martin Odermatt

3 Grundlagen

Bezeichnung	Versions-Nr. / Bezeichnung / Datum
Mündliche Vorstellung des zu entwickelnden Produkts, Horgen	12.3.2019
Vorstellung der Benutzeroberfläche und der implementierten Funktionalität, Horgen	7.5.2019
Kurze Benutzerführung, Horgen	28.5.2019

4 Abnahmeverfahren

Festgestellte Fehler und Mängel müssen innert 10 Tage nach Erhalt der Software bekannt gegeben und protokolliert werden. Als Grundlage dient die nachfolgende Mängelliste.
Mängel und Fehler werden nach Absprache innert vereinbarter Frist behoben.

5 Mängelliste

#	Bezeichnung des Fehlers	Beschreibung des Fehlers	Schritte zur Reproduzierung des Fehlers
0	*Beispiel: Tag hinzufügen und Filtern*	*Beispiel: Wen ein neuer Tag eingegeben und gespeichert wird, erscheint dieser nicht im Filter*	*Beispiel:* *1. Ein Bild mit der Lupe öffnen* *2. Tag eingeben* *3. Klick auf Speichern* *4. Klick auf Filter-Icon* *5. Tag nicht auswählbar*
1	Suche nach Datum	z.B. Taizé-GD vom 20.4.19: In der Ursprungsdatei auf Sharpoint ist der Ordner mit 190420_Taizé_E Abbühl bezeichnet. Darin finden sich 27 Bilder. Diesen ist offenbar das Erstellungsdatum 20.4.19 hinterlegt. Wenn ich in ImageFinder nach diesem Datum suche erscheinen mir lediglich 9 Bilder aus dieser Serie. Das gleiche geschieht mit der Bilderserie zum Osterkonzert: 190421_Osterkonzert_E Abbühl. Dieser Ordner beinhaltet am Urspungsort 38 Bilder, auf ImageFinder finde ich mit der Eingrenzung 21.4.19 nur 7 Bilder.	

6 Abnahmeergebnis

☐ Das Abnahmeobjekt wurde geprüft ohne Mängelanzeige. Die Abnahme erfolgt ohne Vorbehalt.

☒ Das Abnahmeobjekt wurde geprüft und unter Vorbehalt abgenommen. Die Mängel sind innerhalb der vorgegebenen Frist zu beheben und die Lösung ist mittels Nachprüfung nochmals abzunehmen.

☐ Das Abnahmeobjekt wurde geprüft. Die Abnahme wird verweigert.

7 Unterschrift

Name	Ort und Datum	Unterschrift
Tiana Limberger	Horgen, 7.6.19	
Jürg Fisch	Horgen, 4.6.19	
Petra Gassmann	Horgen, 8.6.19	

Anhang L Referenzsystem

- CPU: Intel(R) Core(TM) i7-4720HQ CPU @ 2.60GHz
- GPU: GeForce GTX 960M
- RAM: 16 GB DDR3
- SSD: Kingston RBU-SNS8100S3256GD
- HDD: WDC WC10JPVX-22JC3T0

Anhang M Wireframes

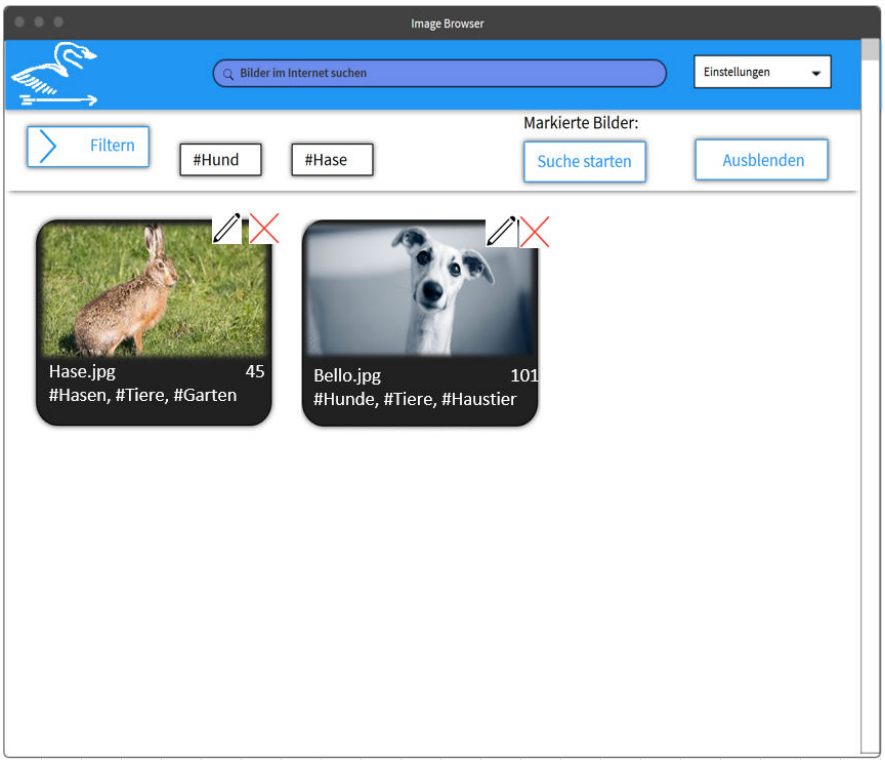


Abbildung 70: Wireframe - Dashboard

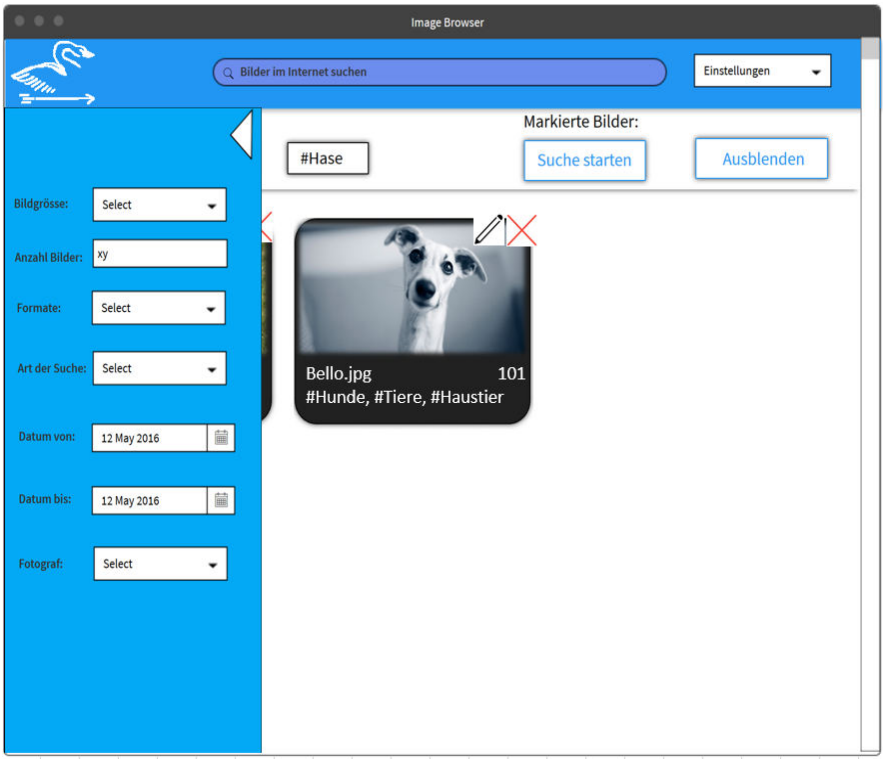


Abbildung 71: Wireframe - Filter

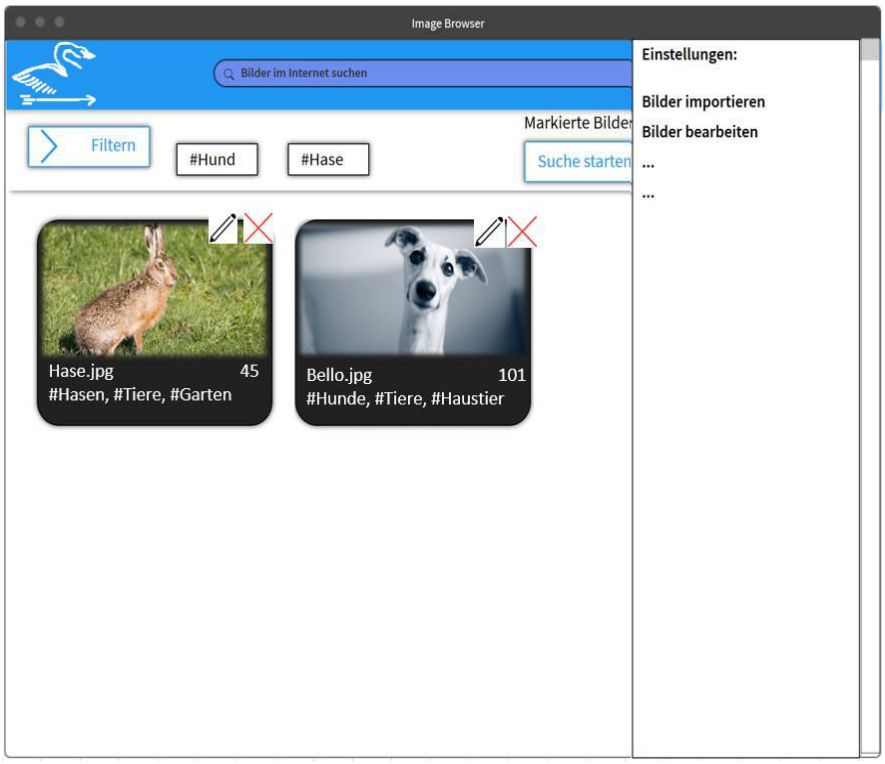


Abbildung 72: Wireframe - Einstellungen

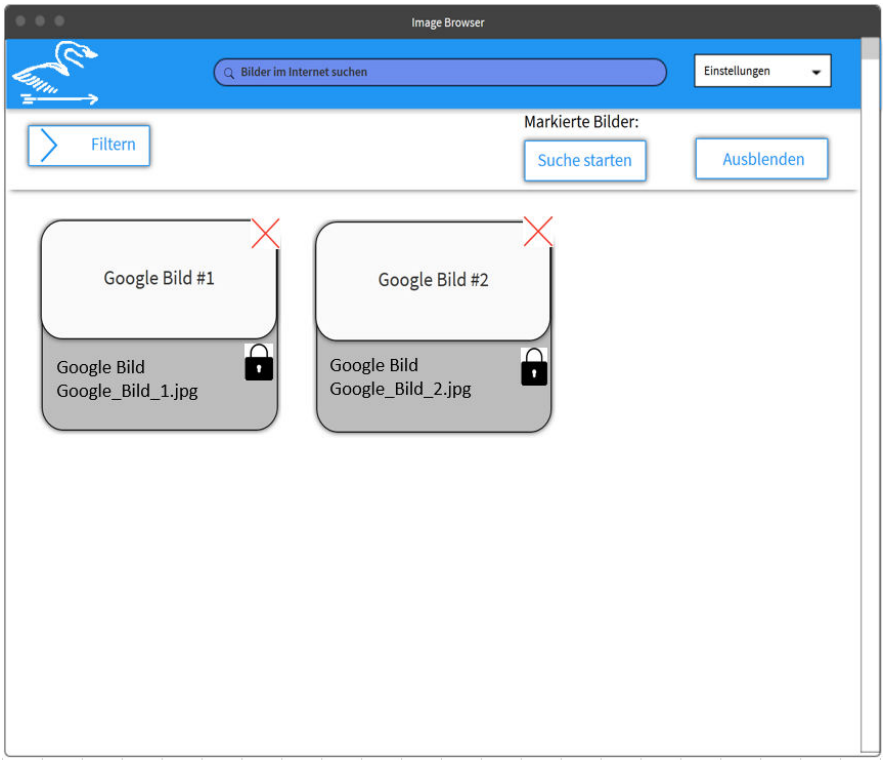
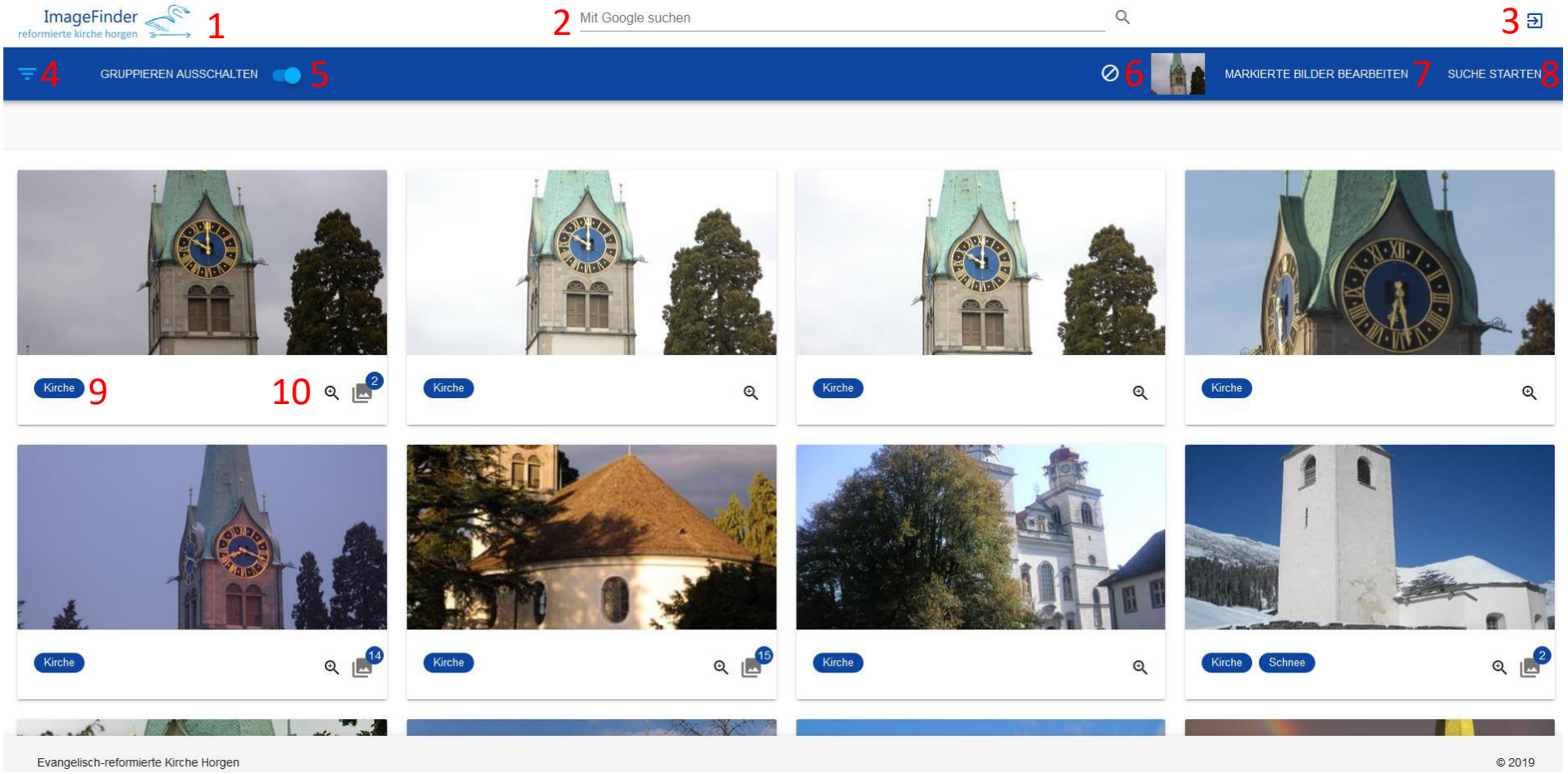


Abbildung 73: Wireframe - Google Suchergebnisse

Anhang N Screens

Dashboard

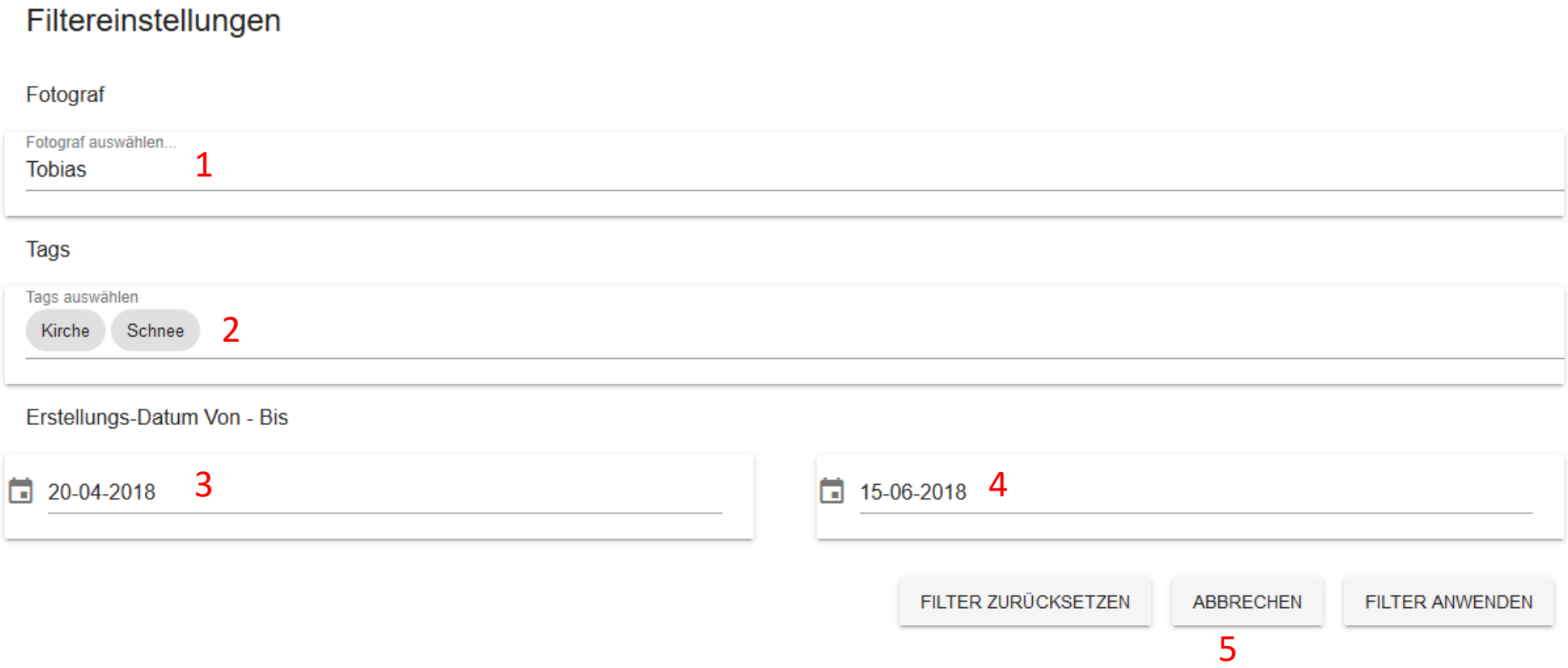


1. Logo
2. Google Suche
3. Einstellungen
4. Filter-Optionen
5. Clustering ein/aus
6. Anzeige der markierten Bilder
7. Menü für das Bearbeiten der markierten Bilder
8. Ähnliche Bilder-Suche starten

Editieransicht

1. Tags bearbeiten
2. Fotograf bearbeiten
3. Erstellungsdatum anpassen
4. Speicher oder Abbrechen
5. Anzeige der zu editierenden Bilder

Filteransicht



1. Fotograf filtern
2. Tags filtern
3. Von-Datum filtern
4. Bis-Datum filtern
5. Filter anwenden, zurücksetzen oder abbrechen

Anhang O Dokumentation der Schnittstellen

Die aus dem FastAPI-Framework generierte Schnittstellenbeschreibung nach OpenAPI Version 3 Spezifikation ist nachfolgend angehängt.

ImageFinder 1.0 OAS3

/openapi.json

ImageFinder

clusters

POST /api/clusters Create Clusters

Parameters Try it out

No parameters

Responses

Code	Description	Links

201	*Successful Response*
	
	application/json
	Controls Accept header.
	
	Example Value Schema
	
	{
	"message": "string"
	}
		No links

heartbeats

GET /api/heartbeat Read Heartbeat

Parameters Try it out

No parameters

Responses

Code	Description	Links

200	*Successful Response*
	application/json
	Controls Accept header.
	Example Value Schema
	{
	"message": "string"
	}
		No links

images

POST /api/image Create Image

Parameters Try it out

No parameters

Request body required multipart/form-data

file * required
string($binary)

tags
array

author
string

Responses

Code	Description	Links

Code	Description	Links
201	*Successful Response* **application/json** Controls `Accept` header. **Example Value** Schema `{ "message": "string" }`	*No links*
400	*Additional Response* **application/json** **Example Value** Schema `{ "message": "string" }`	*No links*
422	*Validation Error* **application/json** **Example Value** Schema `{ "detail": [ { "loc": [ "string" ], "msg": "string", "type": "string" } ] }`	*No links*

POST `/api/images/google` Create Google Images

Parameters Try it out

Name	Description
keywords * required `string` *(query)*	

Responses

Code	Description	Links
201	*Successful Response* **application/json** Controls `Accept` header. **Example Value** Schema `{ "images": [ { "id": 0, "resolution": "string", "format": "string", "thumbnail_uri": "string", "download_uri": "string", "cluster_uri": "string", "edit_uri": "string", "author": "string", "date_taken": "2019-06-12", "tags": [ { "name": "string", "id": 0 } ], "is_google": false } ] }`	*No links*
422	*Validation Error* **application/json** **Example Value** Schema `{ "detail": [ { "loc": [ "string" ], "msg": "string", "type": "string" } ] }`	*No links*
429	*Additional Response* **application/json** **Example Value** Schema `{ "message": "string" }`	*No links*

GET `/api/images` Read Images

Parameters Try it out

Name	Description
neighbors `array[integer]` *(query)*	
filter_by_tags `array[string]` *(query)*	
start_date `string` *(query)*	
end_date `string` *(query)*	
filter_by_author `string` *(query)*	
clustering `boolean` *(query)*	*Default value* : true
page `integer` *(query)*	*Default value* : 1

Responses

Code	Description	Links

Code	Description	Links
200	*Successful Response* **application/json** Controls `Accept` header. **Example Value** Schema	*No links*


```
"size": 0,
"image": {
  "id": 0,
  "resolution": "string",
  "format": "string",
  "thumbnail_uri": "string",
  "download_uri": "string",
  "cluster_uri": "string",
  "edit_uri": "string",
  "author": "string",
  "date_taken": "2019-06-12",
  "tags": [
    {
      "name": "string",
      "id": 0
    }
  ],
  "is_google": false
},
"neighbors": [
  "string"
],
"thumbnails": [
  "string"
]
}
]
}
```


422	*Validation Error* **application/json** **Example Value** Schema	*No links*


```
{
  "detail": [
    {
      "loc": [
        "string"
      ],
      "msg": "string",
      "type": "string"
    }
  ]
}
```


PUT `/api/images` Update Image

Parameters Try it out

No parameters

Request body required application/json

Example Value Schema


```
{
  "ids": [
    0
  ],
  "tags": [
    "string"
  ],
  "author": "string",
  "date_taken": "string",
  "override": true
}
```


Responses

Code	Description	Links
200	*Successful Response* application/json Controls `Accept` header. **Example Value** Schema `{ "message": "string" }`	*No links*
422	*Validation Error* application/json **Example Value** Schema `{ "detail": [ { "loc": [ "string" ], "msg": "string", "type": "string" } ] }`	*No links*

GET /api/google Read Google

Parameters

Try it out

No parameters

Responses

Code	Description	Links
200	*Successful Response* application/json Controls Accept header. **Example Value** Schema	*No links*


```
{
  "clusters": [
    {
      "id": 0,
      "size": 0,
      "image": {
        "id": 0,
        "resolution": "string",
        "format": "string",
        "thumbnail_uri": "string",
        "download_uri": "string",
        "cluster_uri": "string",
        "edit_uri": "string",
        "author": "string",
        "date_taken": "2019-06-12",
        "tags": [
          {
            "name": "string",
            "id": 0
          }
        ],
        "is_google": false
      },
      "neighbors": [
        "string"
      ],
      "thumbnails": [
        "string"
```


authors

GET `/api/authors` Read Authors

Parameters

Try it out

No parameters

Responses

Code	Description	Links

Code	Description	Links

200	*Successful Response*
	
	application/json
	Controls Accept header.
	
	Example Value Schema
	
	{
	"authors": [
	{
	"name": "string"
	}
	]
	}
		No links

statics

GET / Read Index

Parameters Try it out

No parameters

Responses

Code	Description	Links
200	*Successful Response* text/html Controls Accept header. **Example Value** Schema string	*No links*

tags

GET /api/tags Read Tags

Parameters Try it out

No parameters

Responses

Code	Description	Links

200	*Successful Response*
	
	application/json
	Controls `Accept` header.
	
	Example Value Schema
	
	{
	"tags": [
	{
	"name": "string",
	"id": 0
	}
	]
	}
		No links

Schemas

Tag {

name*	string *title: Name*
id*	integer *title: Id*

}

Image {

id	integer *title: Id*
resolution	string *title: Resolution*
format	string *title: Format*
thumbnail_uri	string($uri) *title: Thumbnail_Uri* *maxLength: 65536* *minLength: 1*
download_uri	string($uri) *title: Download_Uri* *maxLength: 65536* *minLength: 1*
cluster_uri	string($uri) *title: Cluster_Uri* *maxLength: 65536* *minLength: 1*
edit_uri	string($uri) *title: Edit_Uri* *maxLength: 65536* *minLength: 1*
author	string *title: Author*
date_taken	string($date) *title: Date_Taken*
tags	**Tags** [...]
is_google	boolean *title: Is_Google* *default: false*

}

Cluster {

id	integer *title: Id*
size	integer *title: Size* *default: 0*
image	**Image** {...}
neighbors	**Neighbors** [...]
thumbnails	**Thumbnails** [...]

}

ClusterListNext {

clusters	**Clusters** [*title: Clusters* *default: List []*

Cluster {

id	integer *title: Id*
size	integer *title: Size* *default: 0*
image	**Image** {...}
neighbors	**Neighbors** [...]
thumbnails	**Thumbnails** [...]

}]

}

Body_update_image {

ids*	**Ids** [	
	title: Ids	
	uniqueItems: true	
	integer]	
tags*	**Tags** [	
	title: Tags	
	uniqueItems: true	
	string]	
author	string	
	title: Author	
date_taken	string	
	title: Date_Taken	
override	boolean	
	title: Override	
	default: true	

}

TagList {

tags	**Tags** [	
	title: Tags	
	default: List []	
	Tag {	
	name*	string
		title: Name
	id*	integer
		title: Id
	}]	

}

ImageList {

images **Images** [
title: Images
default: List []

Image {

id **integer**
title: Id

resolution **string**
title: Resolution

format **string**
title: Format

thumbnail_uri **string**($uri)
title: Thumbnail_Uri
maxLength: 65536
minLength: 1

download_uri **string**($uri)
title: Download_Uri
maxLength: 65536
minLength: 1

cluster_uri **string**($uri)
title: Cluster_Uri
maxLength: 65536
minLength: 1

edit_uri **string**($uri)
title: Edit_Uri
maxLength: 65536
minLength: 1

author **string**
title: Author

date_taken **string**($date)
title: Date_Taken

tags **Tags** [...]

is_google **boolean**
title: Is_Google
default: false

}]

}

Body_create_image {

file* **string**($binary)
title: File

tags **Tags** [
title: Tags
uniqueItems: true
string]

author **string**
title: Author

}

Author {

name* **string**
title: Name

}

AuthorList {
authors
Authors [
title: Authors
default: List []
Author {
name* **string**
title: Name
}]
}

Message {
message* **string**
title: Message
}

ValidationError {
loc* **Location** [...]
msg* **string**
title: Message
type* **string**
title: Error Type
}

HTTPValidationError {
detail
Detail [
title: Detail
ValidationError {
loc* **Location** [...]
msg* **string**
title: Message
type* **string**
title: Error Type
}]
}