Universität Osnabrück
Fachbereich Humanwissenschaften
Studiengang Cognitive Science

# Bachelor-Arbeit

## Aufbau einer interaktiven Internetplattform als Django-Projekt zur Erstellung und Veröffentlichung von Vorher-Nachher-Bildpaaren

Sören Weber
Matrikel Nr.: 945328

Erstgutachter: Prof. Dr. Oliver Vornberger
Zweitgutachterin: Prof. Dr.-Ing. Elke Pulvermüller

Datum: 28.08.2015

# Inhaltsverzeichnis


**1 Einleitung** **4**

**2 Vorgaben** **6**

2.1 Detailansicht . . . . . 6
2.2 Stöbern . . . . . 6
2.3 Bewertung . . . . . 7
2.4 Erstellung . . . . . 7
2.5 Nutzerverwaltung . . . . . 7
2.6 Application Programming Interface . . . . . 7

**3 Umsetzung** **8**

3.1 Grundlegendes: Server . . . . . 8
3.1.1 Python . . . . . 8
3.1.2 virtualenv . . . . . 9
3.1.3 Django . . . . . 9
3.1.4 Mehrsprachigkeit . . . . . 12
3.1.5 PostgreSQL . . . . . 13
3.1.6 Verarbeitung von zeitlichen Informationen . . . . . 14
3.1.7 Administrationsoberfläche . . . . . 14
3.1.8 Einstellungen . . . . . 15
3.2 Grundlegendes: Client . . . . . 15
3.3 Nutzerverwaltung: Server . . . . . 15
3.3.1 Django-allauth . . . . . 16
3.3.2 Django-avatar . . . . . 16
3.3.3 Django-Ansichten . . . . . 16
3.3.4 Setzen der Zeitzone . . . . . 17
3.3.5 Profilansicht . . . . . 17
3.3.6 Nutzersuche . . . . . 17
3.4 Erstellen von Bildpaaren: Server . . . . . 17
3.4.1 Bildpaar-Zustände . . . . . 18
3.4.2 Validierung und Speicherung der Bilder . . . . . 19
3.4.3 Django-Ansichten . . . . . 20

3.4.4 Auslesen der Bild-Metadaten . . . 20
3.4.5 Ausrichtung der Bilder . . . 21
3.5 Erstellen von Bildpaaren: Client . . . 22
3.5.1 Zeitpunkt-Auswahl . . . 22
3.5.2 Wahl geografischer Position . . . 22
3.5.3 Mehrsprachigkeit . . . 23
3.5.4 Markieren korrespondierender Bildpunkte . . . 24
3.6 Detail-Ansicht eines Bildpaares: Server . . . 25
3.6.1 Django-Ansichten . . . 25
3.7 Detail-Ansicht eines Bildpaares: Client . . . 26
3.7.1 Schieberegler . . . 26
3.7.2 Zusätzliche Interaktionsmöglichkeiten . . . 26
3.7.3 Kommentarbereich . . . 28
3.8 Bewertung: Server . . . 29
3.8.1 Django-Ansichten . . . 30
3.9 Bewertung: Client . . . 30
3.10 Stöbern: Server . . . 30
3.10.1 Django-Ansicht . . . 31
3.10.2 Geografische Filterung . . . 31
3.11 Stöbern: Client . . . 32
3.11.1 Filter . . . 32
3.11.2 Sortierung . . . 34
3.11.3 Abfrage der Ergebnisse . . . 34
3.11.4 Anzeige der Ergebnisse . . . 34
3.12 Application Programming Interface: Server . . . 35
3.12.1 Abfrage . . . 36
3.12.2 Erstellung . . . 36
3.12.3 Serialisierung . . . 37

**4 Ausblick** **40**
4.1 Nutzung auf Mobilgeräten . . . 40
4.2 Nutzer-Tracking . . . 40
4.3 Content Delivery Network . . . 41
4.4 Bildformate . . . 41
4.5 Lizenzen der Bilder . . . 41
4.6 Meldemöglichkeit . . . 42

**5 Fazit** **43**

**6 Declaration of Authorship** **45**

# Abbildungsverzeichnis


1 Profilansicht des eigenen Nutzers . . . . . 18
2 Nutzer-Suchansicht . . . . . 18
3 Validierung einer hochgeladenen Datei . . . . . 19
4 Erstellung eines Bildpaares mit vorhandenen Exif-Daten . . . . . 23
5 Erstellung eines Bildpaares, Markierung korrespondierender Punkte . . 24
6 Detail-Ansicht eines Bildpaares als Ersteller (Auszug) . . . . . 27
7 Stöbern-Ansicht . . . . . 32


# Listings


1 Modell Compilation (Auszug) . . . . . 10
2 Modell Image (Auszug) . . . . . 11
3 Modell ImageMarker (Auszug) . . . . . 12
4 Behandlung bei Überschreitung des Antimeridians . . . . . 32
5 Serialisierer des Compilation-Modells (Auszuzg) . . . . . 37
6 Serialisierer des Image-Modells . . . . . 38
7 Serializer-Feldtypen AwareDateField & AwareTimeField . . . . . 39

# Kapitel 1

# Einleitung

Die vorliegende Bachelor-Arbeit beschreibt die Planung und Entwicklung einer Internetplattofrm zur Erstellung und Präsentation von Vorher-Nachher-Bildpaaren. Ziel war es, dem Nutzer das Hochladen zweier Bilder zu ermöglichen und diese möglichst automatisiert für einen direkten Vergleich aufzubereiten. Ein Bildpaar besteht dabei immer aus zwei Bildern, die denselben Ort, Menschen oder Gegenstand zu unterschiedlichen Zeitpunkten darstellen. Beispielsweise könnte dieses Internetportal also genutzt werden um die Entwicklung eines Gebäudes von der Bauphase bis zur Fertigstellung zu visualisieren. Mit einem Schieberegler kann der Nutzer nach erfolgreichem Erstellen des Bildpaares die Sichtbarkeit der beiden übereinandergelegten Bilder steuern. So können Veränderungen einer Szene besonders leicht wahrgenommen und ansprechend präsentiert werden.

Eine Hauptaufgabe dieses Internetportals liegt außerdem in der öffentlichen Bereitstellung der Vorher-Nachher-Bildpaare. Hierzu wurde eine spezielle Ansicht entwickelt, die allen Besuchern der Seite das einfache Stöbern durch die verschiedenen Bildpaare ermöglicht. Dabei kann der Nutzer die angezeigten Ergebnisse sowohl nach bestimmten Kriterien filtern (zum Beispiel Zeitspanne oder Ort der Szene) als auch sortieren (zum Beispiel nach Bewertung oder Erstellungsdatum). Bildpaare, die eine Ortsangabe enthalten, werden auf einer Karte dargestellt. Der Nutzer kann seine Suche auch auf den jeweiligen Kartenausschnitt beschränken. Zusätzlich können Nutzer Kommentare zu einem Bildpaar abgeben und dieses bewerten.

Dieses Projekt wurde serverseitig mithilfe des Web-Frameworks *Django*[1], unter Einsatz der Programmiersprache *Python*[2] realisiert. Nutzerseitig wurden *HTML5*, *CSS3* (unter Einsatz des CSS-Frameworks *Bootstrap*[3]), sowie die Programmiersprache *JavaScript* für Interaktionen im Browser des Nutzers eingesetzt. Bei der Konzeption und Realisierung wurden mögliche Sicherheitsrisiken beachtet und wo möglich minimiert.

Die folgende Arbeit gliedert sich in vier Kapitel. Kapitel 2 beschreibt zunächst kurz, welche Vorgaben für das Projekt zu erfüllen waren. In Kapitel 3 wird anschlie-

[1] `https://www.djangoproject.com/`
[2] `https://www.python.org/`
[3] `http://getbootstrap.com`

ßend dargestellt, wie diese Vorgaben umgesetzt wurden. Dabei wird grundsätzlich zwischen Funktionen auf Seiten des Servers und Funktionen auf Seiten des Nutzers unterschieden, da diese Umgebungen und die Anforderungen an sie sich stark unterscheiden. Kapitel 4 gibt schließlich einen Ausblick über weitere vielversprechende Entwicklungsmöglichkeiten des Projekts. In Kapitel 5 wird schließlich ein Fazit gezogen.

# Kapitel 2

# Vorgaben

Entwickelt werden sollte ein Internetportal, das die Erstellung und Präsentation von Vorher-Nachher-Bildpaaren durch den Nutzer ermöglicht. Ein solches Paar besteht aus zwei Bildern, die denselben Ort, Menschen oder Gegenstand zu unterschiedlichen Zeitpunkten darstellen und dadurch einen direkten Vergleich ermöglichen. Das Portal sollte zweisprachig nutzbar sein.

Die Anforderungen an das System lassen sich in die folgenden Bereiche unterteilen.

## 2.1 Detailansicht

Der Nutzer sollte ein Bildpaar in einer detaillierten Ansicht betrachten können. Hier sollte der Nutzer mit einem Schieberegler die Sichtbarkeit der beiden jeweiligen Bilder steuern können. Die Ansicht sollte außerdem weitere Informationen (Titel, Beschreibung, Ersteller, Bewertung, Aufnahmezeitpunkte und -ort) präsentieren. Des weiteren sollte diese Sicht das Kommentieren des jeweiligen Bildpaares erlauben.

## 2.2 Stöbern

Es sollte eine Ansicht geschaffen werden, die das Durchsuchen aller vorhandenen Bildpaare ermöglicht. Hierbei sollte der Nutzer die Möglichkeit haben, die Ergebnisse nach seinen Vorstellungen zu filtern und zu sortieren. Gewünscht wurde hierfür das Eingrenzen des gesamten Zeitraums, in dem beide Aufnahmezeitpunkte liegen müssen (zum Beispiel von 1990 bis 2000), das Eingrenzen der Zeitspanne, die zwischen beiden Bildern liegt (zum Beispiel von 5 bis 10 Jahren), der Kategorie, sowie das geografische Eingrenzen durch eine Karte.

Die entsprechenden Ergebnisse sollten dann sowohl auf der Karte, als auch tabellarisch dargestellt werden. Bei größerer Anzahl von Ergebnissen auf der Karte sollten „Cluster“ gebildet werden, die mehrere nahe beieinander stehende Marker zu einem neuen Symbol zusammen fassen. In der tabellarischen Ergebnisliste sollten Metadaten, durchschnittliche Bewertung und eine Visualisierung der Zeitspanne dargestellt

werden.

## 2.3 Bewertung

Angemeldeten Benutzern sollte es möglich sein, Bildpaare auf einer Fünf-Sterne-Skala zu bewerten. Besprochen wurden auch negative Eigenschaften der naiven Bewertungs-Sortierung nach durchschnittlicher Sternanzahl: das Sortierergebnis widerspricht bei einer kleinen Bewertungsanzahl in manchen Fällen der intuitiven Erwartung (siehe Abschnitt 3.8). Optional sollte deshalb ein anderes System implementiert werden, das in solchen Fällen ein besseres Verhalten zeigt.

## 2.4 Erstellung

Selbstverständlich sollten Nutzer auch selbst Bildpaare erstellen und für andere Nutzer zugänglich machen können. Hierzu sollte es möglich sein, die Bild-Dateien auf den Server zu laden. Vorhandene Aufnahmeort- oder Zeitpunktdaten der Bilder (in Form von Exif-Daten) sollten optional ausgelesen und verarbeitet werden. Um die korrekte Ausrichtung der beiden Bilder zueinander zu bestimmen, sollte der Nutzer korrespondierende Bildpunkte mit gleichfarbigen Markern markieren können. Diese Daten sollten dann genutzt werden, um die Bilder in einer solchen Weise auszurichten, dass sie sinnvoll in der Detail-Ansicht verglichen werden können. Es wurde vorgeschlagen, sowohl die Berechnung der notwendigen Ausrichtung als auch die Ausrichtung der Bilder selber über bereits implementierte Funktionen der Bibliothek *OpenCV*[4] zu realisieren.

## 2.5 Nutzerverwaltung

Um Bildpaare zu erstellen oder zu bewerten, sollte der Nutzer mit einem Account an der Seite angemeldet sein. Jeder Nutzer sollte ein Profil besitzen, das weitere Informationen zu der Person und den erstellten Bildpaaren präsentiert.

## 2.6 Application Programming Interface

Im Rahmen einer weiteren Bachelor-Arbeit sollte eine *iOS*-Applikation entwickelt werden, die den Nutzer beim Anfertigen und Ausrichten der Bildpaare unterstützt. Diese Applikation sollte das Ergebnis dann in das Internetportal übermitteln können. Zu diesem Zweck sollte eine API entwickelt werden.

[4] `http://opencv.org`

# Kapitel 3

# Umsetzung

Nachfolgend werden die eingesetzten Technologien und deren Einsatz in diesem Projekt beschrieben. Die Gliederung orientiert sich hier an den vorgestellten Bereichen der Vorgaben. Zur Wahrung der Übersichtlichkeit wird die Beschreibung der Umsetzung jeweils (falls zutreffend) in die Bereiche der Server- und Clientprogrammierung gegliedert.

## 3.1 Grundlegendes: Server

Die beiden folgenden Abschnitte beschreiben Konzeption und Umsetzung der für das Projekt grundlegenden Funktionen. Diese lassen sich nicht eindeutig einem der vorgestellten Bereiche zuordnen, da sie in mehreren oder allen Bereichen zur Umsetzung erforderlich waren.

### 3.1.1 Python

Für die Programmierung auf Serverseite wurde die Programmiersprache *Python* eingesetzt. *Python* eignet sich aufgrund der weiten Verbreitung und der hohen Anzahl von Erweiterungspaketen besonders als Grundlage für dieses Projekt, da sich die benötigten Bibliotheken und Sprachanbindungen über mehrere Themenfelder erstrecken. Geplant war ursprünglich der Einsatz der aktuelleren Version 3. Eine der Abhängigkeiten dieses Projektes, die Bildbearbeitungs- und Computer-Vision-Bibliothek *OpenCV* bietet in der stabilen Version allerdings nur eine Sprachanbindung für *Python 2.7* an, weshalb auf diese Version zurück gegriffen wurde. Aus dem Einsatz dieser Version ergibt sich kein Nachteil, da beide Versionen derzeit weiter aktiv betreut werden und eine *Python 3*-Unterstützung für das nächste Major Release von *OpenCV* angekündigt wurde. Der Quelltext dieses Projektes kann dann mit wenig Aufwand auch unter *Python 3* genutzt werden, da auf grundlegende Kompatibilität während der Erstellung geachtet wurde. Ein entscheidender Unterschied zwischen beiden Versionen ist beispielsweise die Interpretation von Zeichenketten. Version 2 kennt hier zwei Arten (*str* und *unicode*); in Version 3 werden Zeichenketten allerdings standardmäßig als Unicode-kodiert

betrachtet. In diesem Fall wurde einerseits ein spezielles, mitgeliefertes *Python*-Modul importiert, das die Unicode-Interpretation für *Python 2* aktiviert:

```
from __future__ import unicode_literals
```

*Django* erlaubt in den später näher beschriebenen Models das Überschreiben von Methoden, deren Rückgabewerte ausgegeben werden, falls eine lesbare Darstellung des Objektes gewünscht ist. Aufgrund der vorher beschriebenen Trennung der Zeichenketten-Typen in Version 2 existieren hier jeweils die Methoden *__str__* und *__unicode__*, während unter Version 3 nur die Methode *__str__* genutzt wird. Hier wurde ein von dem *Django*-Projekt bereitgestellter Dekorator genutzt, der unter Version 2 die Methode *__unicode__* entsprechend definiert und unter Version 3 keine Auswirkungen hat:

```
@python_2_unicode_compatible
class Compilation(models.Model):
[...]
```

### 3.1.2 virtualenv

Um das Bereitstellen und Aktualisieren von Abhängigkeiten des Projektes auf mehreren Rechnersysteme zu erleichtern und Versionskonflikte bei eingesetzten Bibliotheken zu verhindern, wurde *virtualenv*[5] genutzt. Diese Programmsammlung erstellt isolierte Umgebungen, in die die jeweils benötigen Abhängigkeiten installiert werden. Auf diese Weise kann im produktiven Einsatz auch dieselbe Bibliothek in unterschiedlichen Versionen für verschiedene Projekte eingesetzt werden und Versionskonflikte auf Entwicklungssystemen werden ausgeschlossen. Der Einsatz von *virtualenv* erleichtert des weiteren die Dokumentation der Abhängigkeiten eines Projektes. Hierfür kann mit dem Python-Paketverwaltungsprogramm *pip*[6] eine Textdatei generiert werden, die die aktuell installierten Pakete mit deren Versionen enthält. Solche Dateien (in diesem Projekt in dem Unterordner *requirements*) können dann mit dem Quelltext eines Projektes verteilt werden und auf anderen System dazu genutzt werden, automatisch eine identische Umgebung zu erstellen.

### 3.1.3 Django

Als Web-Framework für *Python* wurde *Django* eingesetzt. *Django* ermöglicht die Erstellung von Web-Projekten nach dem *Model-View-Controller*-Muster (MVC). Das Muster motiviert die Aufteilung der Programmfunktionalität Komponenten, die jeweils die zu nutzenden Daten beschreiben (Model), die Präsentation und Interaktion definieren (View) und die Verwaltung von Daten, Präsentationen und Interaktionen beschreibt (Controller). Durch diese isolierte Architektur kann die Lesbarkeit des Quelltextes

[5] `https://virtualenv.pypa.io/en/latest`
[6] `https://pypi.python.org/pypi/pip`

erhöht und die spätere Erweiterung erleichtert werden. *Django* implementiert bereits einige Lösungen für wiederkehrende Anforderungen an Web-Projekte. Beispielsweise implementiert *Django* auf Wunsch ein grundlegendes Authentifikationsverfahren und kann aus den Models eine Administrationsoberfläche generieren, die für Mitarbeiter gedacht ist.

### Modelle

*Django*-Applikationen definieren sogenannte Models, die die Datenstrukturen und das Verhalten von einer bestimmten Klasse von Objekten innerhalb des Projektes beschreiben. Models abstrahieren unter anderem den Zugriff auf die Datenbank. Eine Model-Instanz entspricht einer Zeile in der Tabelle des Models, während ein Datenfeld des Models einer Spalte entspricht.

Nachfolgend werden die zentralen Modelle des Projektes exemplarisch dargestellt und diskutiert.

```
class Compilation(models.Model):
    creator = models.ForeignKey(User)
    creation_time = models.DateTimeField(auto_now_add=True, editable=
        False)
    title = models.CharField(blank=True)
    description = models.TextField(blank=True)
    category = models.ForeignKey(Category)
    position = models.PointField(blank=True, null=True)
    image_before = models.OneToOneField('Image')
    image_after = models.OneToOneField('Image')
    timespan = models.DurationField()
    rating_sort_criterion = models.FloatField()
    rating_average = models.IntegerField(blank=True, null=True)
    is_public = models.BooleanField(default=False)
```

Listing 1: Modell Compilation (Auszug)

Das Modell **Compilation** (siehe Listing 1) beschreibt ein Bildpaar mit seinen zugehörigen Metadaten und Bildern. Das Feld *category* verweist auf ein hier nicht gelistetes Modell, das eine Kategorie beschreibt. Als eins-zu-eins-Beziehung wird außerdem auf das Vorher- und das Nachherbild verwiesen. Diese Beziehung wurde auf diesem Wege modelliert, um Kompatibilität mit der Filterungsbibliothek *django-filter*[7] zu wahren. Zur Definition der zu sortierenden Modellfelder verwendet diese Bibliothek eine Syntax, die den Zugriff auf die beiden Bilder nur ermöglicht, wenn diese als Felder in dem obigen Modell existieren. Ohne diese technische Beschränkung wäre der Verweis von jedem Bild-Objekt auf das zugehörige Bildpaar-Objekt als Fremdschlüssel aus Gründen der Erweiterbarkeit vorzuziehen. Das Feld *position* kann einen geografisch definierten Punkt aufnehmen dessen Angabe aber optional ist.. Der Feldtyp *PointField* stammt

[7] `https://github.com/alex/django-filter`

hier von *GeoDjango*[8]. Die Felder *timespan*, *rating_sort_criterion* und *rating_average* enthalten Werte, die aus anderen vorhandenen Daten berechnet wurden. Sie halten diese Daten also redundant vor. Dieses Vorgehen wurde gewählt, um die Sortier- und Filterzugriffe performanter zu gestalten. Diese Werte müssten ansonsten für jede Anfrage neu berechnet werden und müssen potentiell viele Datensätze bearbeiten, um zu dem Ergebnis zu kommen. Diese Daten werden außerdem ausschließlich zur Anzeige, Sortierung und Filterung und nicht in kritischen Funktionalitäten genutzt.

```
class Image(models.Model):
    original_file = models.ImageField(upload_to=get_original_image_path)
    original_file_thumb = ImageSpecField(source='original_file')
    aligned_file = models.ImageField(blank=True, upload_to=
        get_aligned_image_path)

    creation_datetime = models.DateTimeField()
    creation_time_set = models.BooleanField(default=False)
```

Listing 2: Modell Image (Auszug)

Das Modell **Image** (siehe Listing 2) beschreibt ein einzelnes Bild in einem Bildpaar. Die Angabe des Originalbildes ist verpflichtend, während die Angabe eines ausgerichteten Bildes optional ist. Direkt nach der Erstellung eines Bildpaares durch den Nutzer sind jeweils nur die Originalbilder in diesem Modell angegeben. Die Angabe der ausgerichteten Bilder geschieht dann automatisch, nachdem der Nutzer die korrespondierenden Bildpunkte markiert hat.

Aus dem Originalbild wird automatisch ein Thumbnail, also eine verkleinerte und zugeschnittene Version, generiert. Dieses wird später in der Ergebnisanzeige verwendet, um den entstehenden Datenverkehr, und somit die Kosten für den Betreiber, aber auch die Ladezeiten des Nutzers zu minimieren.

Die Angabe der Aufnahmezeit ist verpflichtend und besteht aus Datum und Uhrzeit. Gibt der Nutzer in dem entsprechenden Formular keine Uhrzeit an, so wird das Feld *creation_datetime* auf das angegebene Datum und der Zeitpunkt auf Mitternacht, sowie das Feld *creation_time_set* auf falsch gesetzt. Bei der Verarbeitung des Aufnahmezeitpunktes wird die Uhrzeit in diesem Fall ignoriert. Dieser Aufbau ermöglicht das einheitliche Ablegen des Aufnahmezeitpunkts in dem Modell und ist hinsichtlich der später beschriebenen Beachtung von Zeitzonen sinnvoll, da für diese die Angabe von Datum *und* Uhrzeit erforderlich ist.

[8] `http://geodjango.org`

```
class ImageMarker(models.Model):
    image = models.ForeignKey(Image)
    order = models.PositiveSmallIntegerField()
    point_x = models.PositiveIntegerField()
    point_y = models.PositiveIntegerField()
```

Listing 3: Modell ImageMarker (Auszug)

Das Modell **ImageMarker** (siehe Listing 3) beschreibt einen von dem Nutzer gesetzten Marker auf einem einzelnen Bild, der mit einem anderen Marker auf dem anderen, zugehörigen Bild korrespondiert. Auf das zugehörige Bild wird per Fremdschlüssel verwiesen. Die Angabe der Markerposition erfolgt in Pixeln entlang der Bildachsen und wird in zwei getrennten Feldern abgelegt. Die Korrespondenz von zwei Markern ist über einen identischen Wert des Feldes *order* eindeutig definiert. Bei Änderung der Markerpositionen durch den Nutzer werden die vorherigen *ImageMarker*-Objekte gelöscht und mit den aktualisierten Werten neu angelegt.

### 3.1.4 Mehrsprachigkeit

Die geforderte Mehrsprachigkeit wird grundsätzlich von *Django* unterstützt. Hierbei wird allerdings nur die Übersetzung von Strings in dem Quelltext ermöglicht, nicht aber die Daten der Modelle. Um auch die Übersetzung von Modellfeldern zu ermöglichen wird die Applikation *django-modeltranslation*[9] genutzt. Dieses erlaubt die Definition von jenen Modellfeldern, die übersetzbar sein sollen und erstellt dann entsprechende Tabellenspalten in der Datenbank.

Bei dem Aufruf der Seite durch einen neuen Nutzer wird die auszulieferende Sprachversion anhand des *Accept-Language* HTTP-Headers bestimmt. Dieser wird von dem anfragenden Browser gesendet und enthält eine priorisierte Liste von möglichen Sprachen. Zusätzlich zu dieser automatisierten Auswahl enthält jede Ansicht des Projektes ein Drop-Down-Menü, mit dem zwischen deutscher und englischer Anzeigesprache gewechselt werden kann. Diese Auswahl wird in der aktiven Sitzung des Nutzers auf dem Webserver abgelegt und ist damit auch für anonyme (nicht angemeldete) Nutzer persistent. Diese Sitzungen werden von *Django* verwaltet. Die zugehörigen Daten der Sitzungen werden in einer eigenen Datenbanktabelle abgespeichert.

Einige mehrsprachige Webseiten codieren die Anzeigesprache innerhalb der URL. So wird der Sprachcode beispielsweise als Prä- oder Suffix in die URL eingebracht und erlaubt auf diesem Weg einerseits das Wechseln durch den Nutzer als auch das Zwischenspeichern der gewünschten Sprache während des Besuchs. In diesem Projekt wurde bewusst auf diese Methode verzichtet, da sie verschiedene URLs für die gleiche Ansicht in unterschiedlicher Sprache nutzt. Da die Anzeigesprache bei einem Seitenaufruf automatisch bestimmt wird, können Ansichten dieses Projektes auch über Sprachen

[9] `https://github.com/deschler/django-modeltranslation`

hinweg mithilfe der gleichen, eindeutigen Adresse ausgetauscht werden. Das Teilen eines Vorher-Nachher-Bildes in sozialen Netzwerken oder auf anderen Wegen wird somit deutlich vereinfacht.

#### Django

Alle zu übersetzenden Zeichenketten innerhalb des Quelltextes wurden mithilfe der in *Django* enthaltenen Lokalisierungshilfen übersetzt. Hierzu wurden die englischen Texte in *Python*-Quellcode und *HTML*-Templates durch das Umschließen mit einer zusätzlichen Funktion als übersetzbar markiert. Diese werden über einen Wartungsbefehl von *Django* in einer Datei im *GNU gettext Portable Object-Format* zusammen getragen. Nach der Übersetzung wird diese Datei in das binäre *Machine Object*-Format überführt, das zur Suche der benötigten Übersetzungen genutzt wird.

Nicht übersetzte Zeichenketten werden in ihrer Ursprungssprache ausgegeben (in diesem Fall Englisch).

#### django-modeltranslation

*Django-modeltranslation* erlaubt zwar pro Sprachversion die Konfiguration von optionalen und verpflichtenden Feldern, hiermit lassen sich jedoch die Vorgaben des Projektes nicht umsetzen. Laut Vorgaben sollten die Anzeigesprachen gleich gestellt sein; es sollte also genügen, wenn der Nutzer ein verpflichtendes Textfeld in *einer* Sprache ausfüllt. Das gewünschte Verhalten wurde realisiert, indem Titel- und Beschreibungsfelder in dem *Django*-Modell als optional markiert und über eine zusätzliche Modell-Validierungsmethode auf obengenannte Bedingung geprüft wurden. Diese Methode wird von Formularen, die auf das Modell zugreifen, im Rahmen der Datenvalidierung aufgerufen und erstellt eine Ausnahme, falls ein verpflichtendes Feld in keiner Sprache ausgefüllt wurde. Diese Ausnahme führt zu der erneuten Anzeige des Formulars mit der Bitte die Daten in einer Sprache anzugeben.

Sollte ein Feld nicht in die anzuzeigende Sprache übersetzt worden sein, so wird es (falls vorhanden) in der jeweils anderen Sprache ausgegeben. Die Anzeigesprache weiterer übersetzter Zeichenketten ist hierdurch unberührt. Im Falle einer Erweiterung der Sprachversionen kann in den Einstellungen von *django-modeltranslation* eine priorisierte Liste der Sprachen, auf die ausgewichen werden soll, bestimmt werden.

### 3.1.5 PostgreSQL

Als Datenbankmanagementsystem wurde *PostgreSQL*[10] gewählt, da dieses System weit verbreitet ist und das *Django* Object-Relational Mapping (ORM) einige *PostgreSQL*-spezifische Datentypen und Funktionen unterstützt. *PostgreSQL* ist außerdem kompatibel mit *PostGIS*, das die Datenbank als Geoinformationssystem (GIS) nutzbar

[10] `http://www.postgresql.org`

macht. Dies erleichtert die Filterung der Suchergebnisse nach geografischen Vorgaben und ermöglicht zukünftige Weiterentwicklungen in diesem Bereich.

### 3.1.6 Verarbeitung von zeitlichen Informationen

#### Zeitzonen

Da das Projekt mehrsprachig gestaltet wurde, ist es wahrscheinlich, dass die Nutzer in verschiedenen Ländern leben. Die Eingabe, Verarbeitung und Ausgabe von Zeitdaten ist in diesem Falle nicht trivial, da unterschiedliche Zeitzonen berücksichtigt werden müssen. Um dieses Problem zu lösen, wird von dem Nutzer bei der Eingabe von zeitlichen Informationen die Angabe der jeweiligen Zeitzone erwartet. Intern werden diese Daten dann in die koordinierte Weltzeit (Coordinated Universal Time, UTC) umgerechnet und in diesem Format gespeichert.

Für die Ausgabe der Daten wird auf die im Profil des Nutzers eingestellte Zeitzone zugegriffen und für die Umwandlung genutzt. So sieht der Nutzer sämtliche Zeitangaben relativ zu der Zeitzone, in der er lebt. Im Profil-Bereich können Nutzer ihre Zeitzone jederzeit ändern.

*Django* unterstützt den Umgang mit Zeitzonen durch Modellfelder, die mit Zeitzonenangaben arbeiten und die Zeitdaten konsistent in der Datenbank ablegen können.

#### Zeitdifferenzen

Ein weiteres Problem im Umgang mit den auftretenden Zeitdaten betrifft die Spanne zwischen zwei Zeitpunkten. Während die Anzahl der Tage zwischen zwei Daten wohldefiniert ist, ist für die Angabe der Anzahl von Monaten oder Jahren keine triviale Berechnung möglich, da Monate und Jahre (in dem gregorianischen Kalender) aus einer unterschiedlichen Anzahl von Tagen bestehen. Die Anzahl der Tage in einem Jahr variiert aufgrund des zusätzlichen Schalttages. Um diese Spannen zu berechnen wurde die Bibliothek *python-dateutil*[11] genutzt, die Kalender, Schaltjahre und Zeitzone berücksichtigt.

Die Ausgabe der Differenz erfolgt über eine Hilfsmethode, die die jeweils größten enthaltenen Zeiteinheiten „ausschreibt“. So wird eine genauere Zeitdifferenz beispielsweise zu der Zeichenkette „1 Jahrzehnt und 2 Monate“ zusammen gefasst. Die Anzahl der maximal auszugebenden Zeiteinheiten kann frei konfiguriert werden und ist zur Abgabe dieser Arbeit auf zwei Einheiten eingestellt.

### 3.1.7 Administrationsoberfläche

Um eine Möglichkeit zu schaffen, die vorhandenen Bildpaare und Nutzer zu moderieren, wurde die Administrationsoberfläche von *Django* aktiviert. Diese generiert die

[11] `https://pypi.python.org/pypi/python-dateutil`

gewünschten Ansichten größtenteils automatisch anhand der erstellten Modelle. Die Oberfläche erlaubt es entsprechend berechtigten Nutzern, sich an einem anderen URL-Endpunkt anzumelden und von dort Bildpaare und Nutzer neu zu erstellen, zu bearbeiten oder zu löschen.

### 3.1.8 Einstellungen

Die Einstellungen des *Django*-Projekts wurden modular für verschiedene Einsatzszenarien (beispielsweise Entwicklung, Demonstration, produktiver Einsatz) angelegt. Läuft das Projekt nicht im Entwicklungsbetrieb, wird die Angabe aller geheimen Daten (zum Beispiel private API-Schlüssel) über Umgebungsvariablen erwartet. Ist dies nicht der Fall, kann das Projekt nicht gestartet werden. So wird das Risiko minimiert, dass geheime Daten als Teil des Quelltextes versehentlich veröffentlicht werden. Bei Speicherung der Geheimnisse im Quelltext kann dies beispielsweise passieren, falls das Projekt (beispielsweise über ein Versionskontrollsystem öffentlich zugänglich ist.

## 3.2 Grundlegendes: Client

Auf Nutzerseite sollte das Projekt über einen Web-Browser bedienbar sein. Um dies zu erreichen, wurden die Auszeichnungssprache *HTML5* (Hypertext Markup Language) eingesetzt. Mithilfe dieser Sprache werden die Seiteninhalte semantisch strukturiert. Für die Formatierung der Seiteninhalte wurde *CSS3* (Cascading Style Sheets) eingesetzt. Diese Sprache ermöglicht das Festlegen von Darstellungsoptionen. Interaktive Elemente, die auf der Nutzerseite berechnet werden mussten, wurden mit der Skriptsprache *JavaScript* realisiert. *JavaScript*-Code wird innerhalb des Browsers interpretiert und im Kontext der aufrufenden Seite ausgeführt. Die drei genutzten Sprachen sind standardisiert und werden von sämtlichen aktuellen Browsern interpretiert.

Um die Gestaltung des Projektes zu erleichtern wurde außerdem das *CSS*-Framework *Bootstrap* eingesetzt. Dieses enthält unterschiedliche Hilfswerkzeuge, um häufig benötigte Gestaltungsmuster umzusetzen und bietet sinnvolle Standardeinstellungen. *Bootrstrap* unterstützt des weiteren die (*Responsive Design* genannte) Anpassung der Darstellung an verschiedene Geräte-Arten (zum Beispiel Tablets oder Smartphones). Diese Anpassung war nicht Teil dieser Arbeit, ist durch den Einsatz dieses Frameworks allerdings zukünftig mit weniger Aufwand zu erreichen.

## 3.3 Nutzerverwaltung: Server

Die grundlegende Funktionalität für die Nutzerverwaltung ist bereits in *Django* implementiert. Hierzu zählen Login, Nutzergruppen und Berechtigungen. Das Framework schlägt außerdem bereits sinnvolle und bekannte Standards zum Umgang mit Nutzern vor (beispielsweise das korrekte Hashing von Passwörtern).

### 3.3.1 Django-allauth

Um das massenhafte Veröffentlichen von Werbung oder anderen unerwünschten Inhalten zu erschweren, wird eine verifizierte E-Mail-Adresse für einen aktiven Account vorausgesetzt. Da *Django* diese Funktionalität nicht standardmäßig implementiert, wurde die Applikation *django-allauth* eingesetzt. Diese erlaubt die Nutzung und detaillierte Konfiguration von Anmelde- und Verwaltungsvorgängen, die häufig in diesem Bereich eingesetzt werden. In diesem Projekt realisiert *django-allauth*[12] die Registrierung, An- und Abmeldung, die Verwaltung und Verifizierung der E-Mail-Adresse des Nutzers, sowie die Möglichkeit, vergessene Passwörter über einen per Mail versandten Link zurück zu setzen. Das Anmelden über einen Drittanbieter (wie beispielsweise Google oder Facebook) wird hiermit ebenfalls grundsätzlich unterstützt. Spätere diesbezügliche Erweiterungen können anhand der Projekt-Einstellungen vorgenommen werden.

### 3.3.2 Django-avatar

Angemeldete Nutzer können Avatare (kleine Bilder, die auf ihrer Profilseite sichtbar sind) bestimmen. Avatare können über eine eigene Maske über die Profilseite auf den Server geladen und zur Nutzung frei gegeben werden. Wurde kein Avatar auf diesem Wege festgelegt, wird der Drittanbieter *Gravatar*[13] genutzt, der webseitenübergreifende Avatare anbietet, die über die E-Mail-Adresse des Nutzers zugeordnet werden. In diesem Fall wird der Avatar durch das Übermitteln des Hash-Wertes der Nutzer-E-Mail-Adresse angefordert und angezeigt. Sollte auch dort kein Avatar hinterlegt sein, wird aus dem übermittelten Wert ein eindeutiges Muster generiert, das an dieser Stelle verwendet wird (siehe Abbildung 1).

Die Umsetzung dieser Funktionalität erfolgte unter Nutzung der Applikation *django-avatar*[14] , die sowohl den Upload von Bilddateien auf den Server, als auch die oben beschriebene Abfrage des Anbieters *Gravatar* ermöglicht.

### 3.3.3 Django-Ansichten

Die Funktionen Nutzeranmeldung, Änderung der E-Mail-Adresse und des Passworts, sowie die Änderung des Avatars sind über *Django*-Ansichten der oben genannten Applikationen implementiert worden. Diese benötigen in zur Implementation der gewünschten Funktionalität keine weitere Konfiguration und wurden deshalb direkt über den *URL-Dispatcher* von *Django* eingebunden.

Das Abändern der eigenen Profildaten durch einen Nutzer, das Betrachten von Profilseiten, sowie die Suche nach Benutzern wurde über eigene Ansichten realisiert. Diese Ansichten wurden über Nutzung der *class-based generic views* umgesetzt, deren

[12] `https://github.com/pennersr/django-allauth`
[13] `https://www.gravatar.com`
[14] `https://github.com/grantmcconnaughey/django-avatar`

Standardverhalten weitgehend erhalten wurde.

### 3.3.4 Setzen der Zeitzone

Im Gegensatz zu den von dem Nutzer gesprochenen Sprachen existiert derzeit noch kein standardisierter HTTP-Header, der die Zeitzone des Nutzers anzeigt. Falls also eine Lokalisierung der Zeitangaben gewünscht ist, muss eine entsprechend zu verwendende Zeitzone hinterlegt werden. Diese Angabe kann von angemeldeten Nutzern in den eigenen Profil-Einstellungen bearbeitet werden. Dort wird ein Drop-Down-Menü eingeblendet, das sämtliche Zeitzonen beinhaltet.

### 3.3.5 Profilansicht

Es wurde eine Ansicht implementiert, die dem Nutzer Informationen zu seinem eigenen Benutzerkonto, oder dem eines anderen Nutzers präsentiert (siehe Abbildung 1). Diese Information bestehen aus nutzergesetzten Attributen (zum Beispiel dem Wohnort), aggregierten Daten (zum Beispiel die Anzahl der veröffentlichten Bildpaare oder die durchschnittliche Bewertung aller Bildpaare des Nutzers), sowie einer Auflistung der Bildpaare des Nutzers. Wird ein fremdes Profil betrachtet, so werden an dieser Stelle ausschließlich die veröffentlichten Bildpaare angezeigt, bei Ansicht des eigenen Profils erscheinen zusätzlich auch die eigene, noch unveröffentlichte Bildpaare. Diese Funktionalität kann beispielsweise genutzt werden, falls der Erstellungsprozess nach Eingabe der Metadaten abgebrochen wurde. In diesem Fall wurde das Bildpaar bereits erstellt, aber noch nicht ausgerichtet. Wählt der Nutzer aus dieser Ansicht ein solches Bildpaar aus, wird er automatisch in die Ausrichtungs-Ansicht umgeleitet. Bei dem Betrachten des eigenen Profils werden außerdem Links zu Ansichten eingeblendet, die das Bearbeiten der Profildaten, der E-Mail-Adresse, des Passworts, sowie des Avatars ermöglichen.

### 3.3.6 Nutzersuche

Eine Ansicht erlaubt es authentifizierten Nutzern, alle Nutzer des Portals anhand ihres Namens zu durchsuchen (siehe Abbildung 2). Die Zeichenkette nach der gesucht werden soll, muss derzeit mindestens drei Zeichen lang sein. Diese Begrenzung ist optional und wurde zur Begrenzung der Ergebnismenge implementiert.

## 3.4 Erstellen von Bildpaaren: Server

Um ein Bildpaar zu erstellen, wird der Nutzer gebeten, in drei aufeinanderfolgenden Ansichten die benötigten Daten einzugeben. Die beiden ersten Schritte (Hochladen der Dateien, Eingabe der zusätzlichen Daten) liefern nach vollständiger Bearbeitung durch

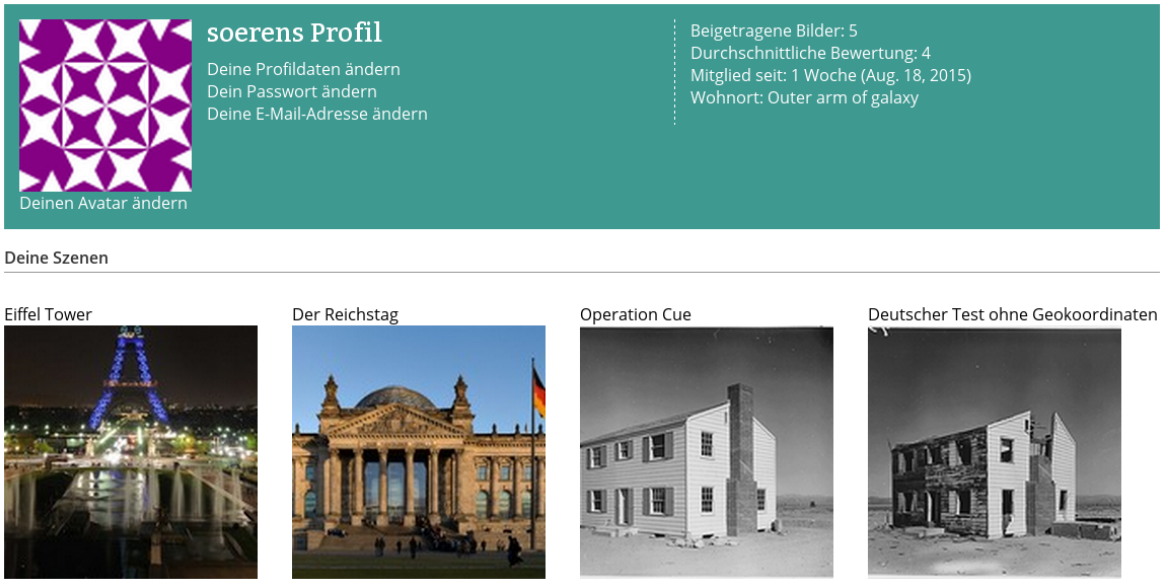


Abbildung 1: Profilansicht des eigenen Nutzers


Abbildung 2: Nutzer-Suchansicht

den Nutzer bereits genug Daten, um entsprechende Instanzen der Modelle *Compilation* und *Image* anzulegen. Diese werden deshalb in Anschluss an den zweiten Schritt angelegt.

### 3.4.1 Bildpaar-Zustände

Das erstellte Bildpaar gilt in diesem Moment als „nicht ausgerichtet“. Ein Bildpaar gilt als ausgerichtet, wenn die beiden zugehörigen Bilder (*Django*-Modell *Image*) jeweils eine ausgerichtete Version (Modellfeld *aligned_file*) des Originalbildes (Modellfeld *original_file*) enthalten. Ist dies nicht der Fall, kann ein Bildpaar nicht veröffentlicht (also für andere Nutzer sichtbar gemacht) werden.

Der Veröffentlichungszustand wird in dem Modellfeld *is_public* abgebildet. Eine Veränderung dieses Zustandes innerhalb des Projektes wird mit Nutzung der Methode *set_visibility* beantragt. Ist die obige Voraussetzung zur Zeit des Methodenaufrufs nicht

gegeben, wird *is_public* nicht verändert.

### 3.4.2 Validierung und Speicherung der Bilder

Um die hochgeladenen Bilddateien in die Modelle von *Django* einzupflegen, wurde das bereits implementierte *ImageField* genutzt, das die Validierung und Speicherung einer Bilddatei unterstützt.

Von Nutzern hochgeladene Dateien stellen in mehrerer Hinsicht ein Sicherheitsrisiko dar. So könnte ein Angreifer beispielsweise versuchen, eine Datei anderen Formats hochzuladen, die dann von anderen Nutzern im Browser geladen würde. Dies könnte beispielsweise *Cross-Site-Scripting*-Attacken (XSS) ermöglichen. Ein Angreifer könnte außerdem versuchen, den Speicher des Servers durch das Hochladen einer sehr großen Datei voll laufen zu lassen. Dieser Angriff könnte zu einer Nichterreichbarkeit des Dienstes für andere Nutzer führen (*Denial of Service*, DoS).

Um diese Risiken zu minimieren werden die hochgeladenen Dateien (zusätzlich zu der Prüfung durch oben genannten Feldtypen) über die Magische Zahl der Dateien auf deren Dateitypen geprüft. Diese Prüfung erfolgt unter Nutzung der Bibliothek *python-magic*[15]. Aufgrund der weiten Verbreitung und der Kompatibilität mit *OpenCV* wurden die zulässigen Typen somit auf die *MIME*-Typen „image/jpeg“ und „image/png“ begrenzt. Andere Datei-Formate führen dazu, dass dem Nutzer eine Fehlermeldung angezeigt wird (siehe Abbildung 3).

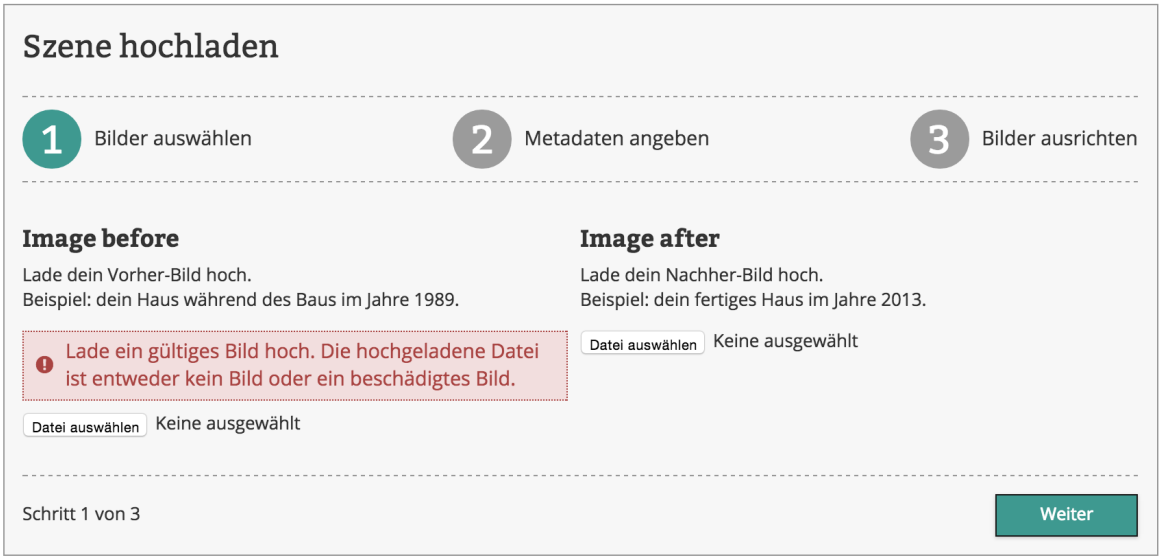


Abbildung 3: Validierung einer hochgeladenen Datei

Der vorgelagerte Webserver wird außerdem so konfiguriert, dass er nur eine maximale Dateigröße während eines Uploads akzeptiert, bevor er die Übertragung unterbricht und dem Nutzer eine entsprechende Fehlermeldung anzeigt.

[15] `https://github.com/ahupp/python-magic`

### 3.4.3 Django-Ansichten

Diese Funktionalität wurde mit zwei *Django*-Ansichten realisiert: *CompilationCreateWizard*, sowie *CompilationAlignmentView*.

Das Hochladen und die Eingabe der weiteren Daten zu einem Bildpaar werden innerhalb der ersten Ansicht implementiert. Diese Ansicht erbt von *FormWizard* der Bibliothek *django-formtools*[16]; diese ermöglicht das Aufteilen von Objektveränderungen oder -erstellungen auf mehrere Schritte (Seiten) und unterstützt außerdem das Vorausfüllen von Daten für den Nutzer in diesen Schritten. Zur Nutzung von *FormWizard* werden einerseits die Schrittreihenfolge und andererseits die jeweiligen anzuzeigenden Formulare der Schritte festgelegt. In diesem Fall wurden die benötigten Daten innerhalb der Formulare *CompilationUploadForm* und *CompilationMetaDataForm* modelliert. Nach dem Upload-Schritt werden eventuell vorhandene *Exif*-Daten als Initialwerte in das Metadaten-Formular aufgenommen. Wurde mehr als ein Aufnahmeort in diesen Daten gefunden, können diese Daten nicht mehr innerhalb des eigentlichen Formulares dargestellt werden. In diesem Fall wird eine spezielle Kontext-Variable an das Template übergeben, die einen Hinweistext einblendet. Hier kann der Nutzer außerdem einen der beiden Orte auswählen oder über die Karte einen alternativen Ort angeben.

Die Sicht *CompilationAlignmentView* ist kein eigentlicher Teil des Erstellungsassistenten, sondern ein eigenständiger Teil des Projektes. Das Template dieser Sicht wurde so umgesetzt, dass es von zwei verschiedenen Basis-Templates erben kann. Ist das auszurichtende Bildpaar noch nicht ausgerichtet, so erbt es von einem „Wizard“-Template, das auch zur Anzeige der beiden vorigen Schritte genutzt wurde. Auf diese Weise fügt sich der Ausrichtungsschritt visuell als notwendiger Bestandteil der Erstellung für den Nutzer in den Assistenten ein. Wurde das Bildpaar bereits ausgerichtet, wird das Standard-Eltern-Template aller Ansichten des Projektes ausgegeben.

### 3.4.4 Auslesen der Bild-Metadaten

*JPEG*-Bilddateien können Metadaten in dem standardisierten Format *Exif* (Exchangeable Image File Format) enthalten. Diese Daten können die genutzte Kamera, deren Einstellungsparameter, Zeitpunkt und Ort der Aufnahme beinhalten. Um den Nutzer bei der Angabe von Daten über die beiden Bilder zu unterstützen, wird nach dem Hochladen versucht, Standorte und Aufnahmezeitpunkte aus den Exif-Daten der Bilder zu extrahieren. War dies erfolgreich, so werden sie dem Nutzer im nächsten Schritt als Formularwerte vorgeschlagen. Da die Werte fehlerhaft sein können, kann der Nutzer sie jederzeit übersteuern. Das Auslesen dieser Daten aus den Bilddateien erfolgte mit der Programmbibliothek *ExifRead*[17]. Diese erlaubt den Zugriff auf eventuell vorhandene Daten und macht diese als *Python*-Datentypen zugänglich.

[16] `https://github.com/django/django-formtools`
[17] `https://pypi.python.org/pypi/ExifRead`

In den *Exif*-Daten wird die Zeitzone der gespeicherten Zeitpunkte nicht abgelegt. Zur Berechnung des tatsächlichen Zeitpunkts wird diese Angabe allerdings benötigt. Der Nutzer wird daher bei Angabe der Metadaten darum gebeten, die Zeitangaben um die entsprechenden Zeitzonen für beide Bilder zu ergänzen. Für die Auswahl wird stets die gespeicherte Zeitzone des Nutzers als Standardwert vorgeschlagen.

### 3.4.5 Ausrichtung der Bilder

Das Ausrichten beider Bilder zueinander wird über Funktionen der Bildbearbeitungs- und Machine-Vision-Bibliothek *OpenCV 2* realisiert. *OpenCV* bietet in der derzeitig stabilen Version eine Sprachanbindung für *Python 2.7* (siehe Abschnitt 3.1).

Die enthaltene Funktion *findHomography* wurde genutzt, um unter Angabe der korrespondierenden Punkte beider Bilder eine perspektivische Transformationsmatrix, die beide Punktemengen möglichst genau aufeinander abbildet, zu erstellen. Diese Matrix wird in einem weiteren Schritt über die Funktion *warpPerspective* dazu genutzt, das flächenmäßig kleinere Bild auf das größere abzubilden. Diese Vorgehensweise wurde gewählt, um Informationsverlust durch Herunterskalieren während der Transformation zu minimieren. Der Nutzer kann außerdem in diesem Schritt ein automatisches Zuschneiden des Bildpaares aktivieren, das Bildbestandteile entfernt, die nicht in beiden Bildern enthalten sind. Der eingesetzte Algorithmus berechnet in diesem Fall die Position der Eckpunkte des kleineren Bildes nach der Transformation mithilfe der Funktion *perspectiveTransform*. Anhand dieser Punkte sind die Flächen, die keine Bilddaten enthalten, eindeutig bestimmt. Daraufhin wird innerhalb der Punkte die zweitgrößten und zweitkleinsten Werte für beide Achsen bestimmt. Diese vier Koordinaten beschreiben ein Rechteck, das zumeist sämtliche Nicht-Bild-Flächen ausschließt und deshalb zum Beschneiden beider Bilddateien genutzt wird. Dieser Algorithmus ist nicht optimal, das heißt er findet nicht das größtmögliche Rechteck innerhalb des transformierten Bildes, und liefert unter bestimmten Umständen nicht das erwartete Ergebnis. Dies ist beispielsweise der Fall, falls die angewandte Transformation das Bild stark verzerrt hat. Da hauptsächlich ähnliche Bilder in diesem Portal genutzt werden, produziert dieses Verfahren in den meisten Fällen aber ein sinnvolles Ergebnis. Sollte der Nutzer mit dem Ergebnis nicht zufrieden sein, so kann er diesen Schritt jederzeit wiederholen und die Beschneidung deaktivieren.

Nach der erfolgreichen Transformation werden beide Bilddateien in einem definierten Ordner für ausgerichtete Bilder abgelegt, eventuell vorhandene ausgerichtete Bilder gelöscht und die entsprechend aktualisierten Pfade im Modell gespeichert.

## 3.5 Erstellen von Bildpaaren: Client

### 3.5.1 Zeitpunkt-Auswahl

Zur Unterstützung der Zeitpunktauswahl durch den Nutzer wurde eine JavaScript-Bibliothek namens *bootstrap-datetimepicker*[18] genutzt. Die Bibliothek zeigt dem Nutzer eine grafische Auswahlmaske für Datum und Uhrzeit und überträgt die gewählten Daten in entsprechend verknüpfte Textfelder.

Nach den Vorgaben sollte für die Erstellung eines Bildpaares die Angabe eines Aufnahmedatums verpflichtend, die Angabe einer Uhrzeit aber optional sein. Dies wird ermöglicht und verdeutlicht, indem dem Nutzer zwei Eingabefelder für diese Angaben präsentiert werden. Auf der Server-Seite wird der Zeitpunkt stets in einem Modellfeld des Typs *DateTime* gespeichert (siehe Abschnitt 3.1.3).

### 3.5.2 Wahl geografischer Position

Um die Auswahl und Visualisierung des Aufnahmeortes zu realisieren, kam die *Google Maps JavaScript API v3*[19] zum Einsatz. Diese ermöglicht die Anzeige einer Landkarte, darauf platzierten Markern, sowie das Überführen von Adressen in Geo-Koordinaten und vice versa (*Geocoding*).

Der Nutzer wird bei der Auswahl der geografischen Position durch ein Kartenelement unterstützt. Auf dieser Karte ist ein Marker positioniert, der von dem Nutzer verschoben werden kann. Bei einer solchen Verschiebung werden über ein JavaScript-Ereignis die aktuellen geografischen Koordinaten des Markers in Textfelder übertragen. Ebenso wird bei Änderung der Textfelder der Marker entsprechend auf der Karte neu positioniert. Über ein weiteres Textfeld ist es außerdem möglich, den Marker an eine bestimmte Adresse zu setzen. Diese Funktion wurde über eine *Geocoding*-Anfrage über *Google Maps* realisiert. Nach Beantwortung der Anfrage wird der Marker auf die Position des relevantesten Suchergebnisses gesetzt.

Falls in den *Exif*-Daten der Bilder Standortdaten gefunden wurden, werden diese dem Nutzer auf unterschiedlichem Wege dargestellt. Falls nur ein Bild ein solches Datum enthielt, wird dieser Ort direkt auf der Karte und in den Textfeldern zur Eingabe der Geokoordinaten eingetragen. Falls beide Bilder jeweils Daten enthielten, wird ein Hinweisfeld angezeigt, das beide Koordinatenpaare enthält und den Nutzer um Auswahl bittet (siehe Abbildung 4). Da die meisten Nutzer vermutlich keinen intuitiven Zugang zu Geokoordinaten haben werden, wird nach Laden der Seite eine rückwärtsgewandte *Geocoding*-Anfrage gestellt. Die Koordinaten werden nach Beantwortung der Anfrage durch eine menschenlesbare Adresse ersetzt. Durch Klick auf eine der Optionen werden Kartenmarker und Koordinaten-Textfelder entsprechend gesetzt.

[18] `https://github.com/Eonasdan/bootstrap-datetimepicker`
[19] `https://developers.google.com/maps/documentation/javascript`

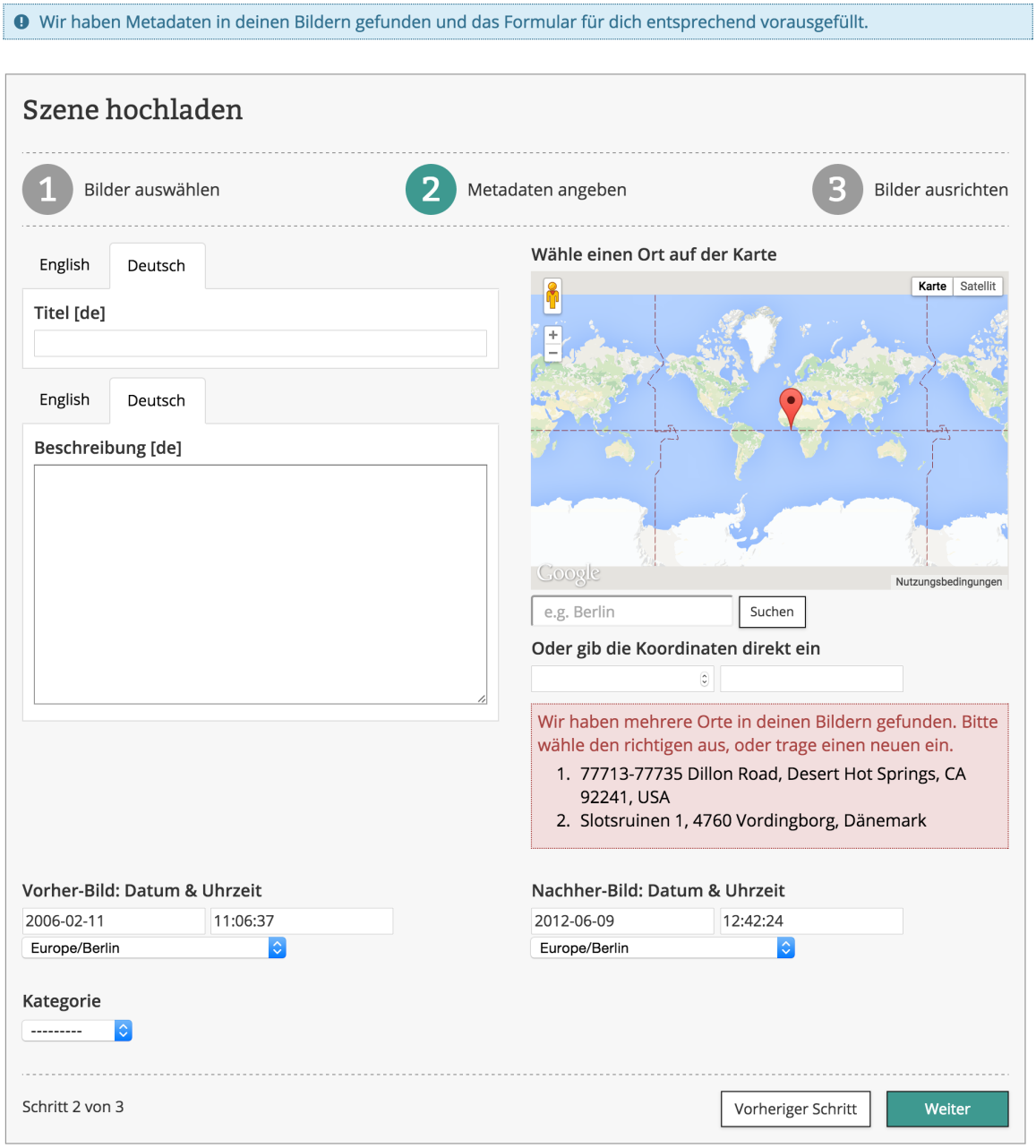


Abbildung 4: Erstellung eines Bildpaares mit vorhandenen Exif-Daten

### 3.5.3 Mehrsprachigkeit

Da das Verfassen von Titel und Beschreibung mehrsprachig möglich sein sollte, wird der das Bedienelement *Tabs* der Bibliothek *JQuery UI*[20] genutzt, um dem Nutzer eine Auswahl über die einzugebende Sprache über grafische Reiter zu geben. Übersetzbare Modellfelder werden standardmäßig in Formularen mithilfe eines Formularfeldes pro möglicher Sprache dargestellt. Zur Wahrung der Nutzbarkeit und Übersichtlichkeit wird ein Modellfeld immer nur in einer Sprache angezeigt. Der Nutzer kann über Reiter-Elemente über den Textfeldern, die bei Klick zu der gewünschten Textfeld-Version wechseln, die gewünschte Sprache einblenden.. Standardmäßig wird der Reiter für die

[20] `https://jqueryui.com`

aktuell gewählte Sprache des Nutzers geöffnet.

### 3.5.4 Markieren korrespondierender Bildpunkte

Um eine Ausrichtung der Bilder zueinander zu erreichen, muss der Nutzer korrespondierende Bildpunkte auf beiden Bildern markieren. Hierzu wurde eine Ansicht geschaffen, in der farbige Marker auf den beiden Bildern durch Ziehen des Markers mit der Maus (*Drag'n'Drop*) auf einen gewünschten Bildpunkt positioniert werden können. Der Nutzer wird hierbei durch eingeblendete vergrößerte Ansichten der markierten Punkte unterstützt (siehe Abbildung 5).

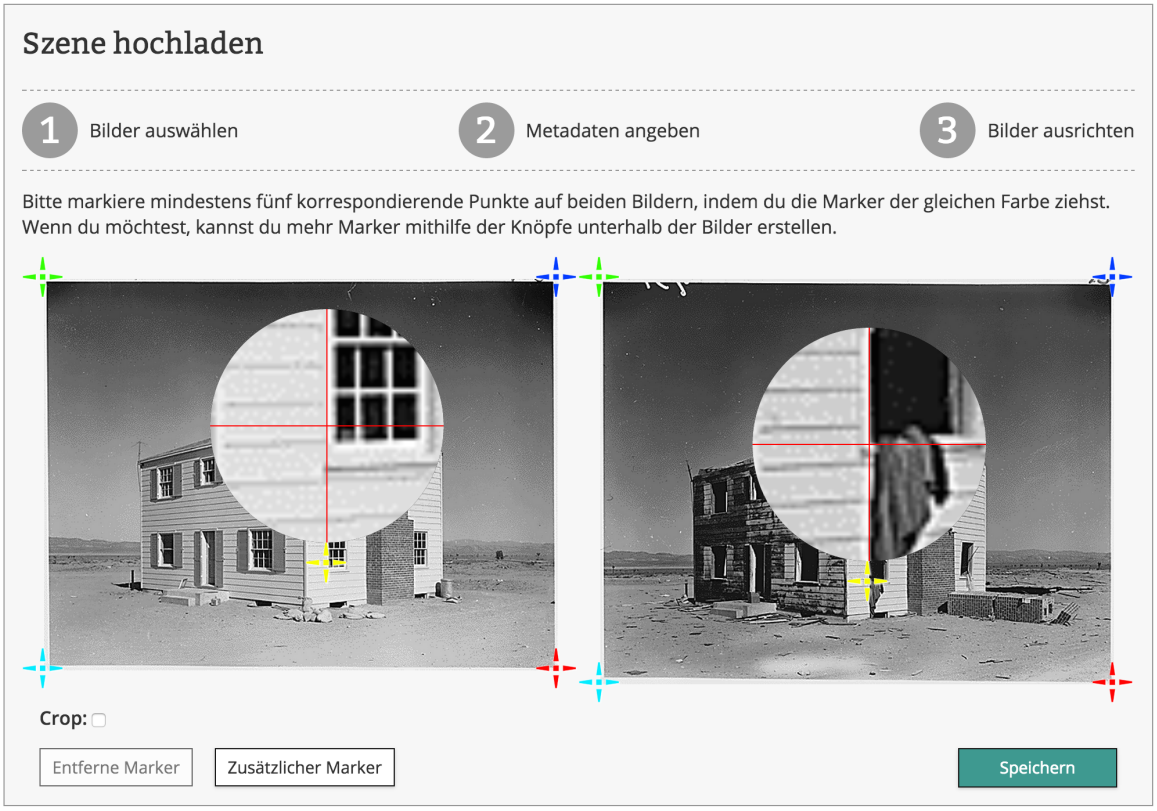


Abbildung 5: Erstellung eines Bildpaares, Markierung korrespondierender Punkte

#### Interaktion

Die angezeigten Marker sind *Scalable Vector Graphics*-Objekte (SVG), die per *JavaScript* erstellt und positioniert werden. Die Marker wurden als *SVG*-Objekte realisiert, da diese über *CSS* in ihrem Erscheinungsbild angepasst werden können. In diesem Fall wurde so das Einfärben der Marker möglich, ohne für jede Version eine eigene Bilddatei generieren zu müssen. Die Möglichkeit, die Marker mit der Maus zu verschieben, wurde mit Einsatz des *Draggable*-Typs der Interaktionselement-Bibliothek *JQuery UI* realisiert.

Während des Ziehens eines Markers wird über diesem eine stark vergrößerte Ansicht des Bildes unter der Mitte des Markers angezeigt. Diese Ansicht kann den Nutzer dabei

unterstützen, den Marker möglichst genau zu positionieren, um eine optimale Ausrichtung der Bilder zu erreichen. Diese vergrößerte Ansicht wird außerdem auch stets für den korrespondierenden Markers auf dem anderen Bild eingeblendet, um einen Vergleich der Positionen zu erleichtern. Realisiert wurde die vergrößerte Ansicht mithilfe eines *HTML*-Containers, der das stark vergrößerte Bild als Kindelement enthält. Dieses Bild ist nur dort zu sehen, wo das kleinere Elternelement sichtbar ist. Bei Bewegung des Markers wird dieses Bild innerhalb des Containers entsprechend des neuen markierten Punktes verschoben. Außerdem folgt der Container der Position des Mauszeigers.

#### Speicherung

Die Marker-Koordinaten werden in einem *HTML*-Formular mit einer definierten Form gespeichert, das von *Django FormSet* genannt wird. Dieses standardisierte Formular erlaubt die Angabe von einer dynamischen Anzahl von gleich geformten Datensätzen. Zur Verwaltung der enthaltenen Daten, benötigt das Formular verpflichtende Felder, die Metadaten (wie beispielsweise aktuelle oder maximale Datzensatz-Anzahl) enthalten. In diesem Fall besteht ein Datensatz aus jeweils zwei korrespondierenden Koordinatenpaaren. Die Koordinaten beziehen sich auf die absoluten Pixel der Bilder in ihrer wahren Größe. Diese Daten werden also bereits auf der Clientseite umgerechnet und abgelegt.

Bei Verschieben des Markers werden die Daten in dem entsprechenden Formularbereich abgeändert. Ebenso werden die Marker zur Zeit des Ladens der Seite auf den Bildern positioniert, falls der Nutzer bereits vorher Marker auf diesen Bildern positioniert hatte.

Eine gute Ausrichtung der Bilder erfordert in manchen Fällen mehr als die fünf minimalen Markerpaare. In diesem Fall kann der Nutzer per Knopfdruck ein weiteres Markerpaar erstellen. Ebenso können nicht mehr benötigte Markerpaare (bis zu der minimalen Marker-Anzahl) wieder entfernt werden. Diese Funktionalität wurde per *JavaScript* realisiert, das Marker-Objekte in dem *Document Object Model* (DOM) bearbeitet und ebenso das zugehörige FormSet erweitert oder kürzt.

## 3.6 Detail-Ansicht eines Bildpaares: Server

### 3.6.1 Django-Ansichten

Die Detailansicht eines Bildpaares wird innerhalb der *Django*-Sicht *CompilationDetailView* implementiert. Da eine Ansicht, die eine bestimmte Modell-Instanz detailliert darstellt, in vielen Szenarien benötigt wird, ist diese Art von Ansicht bereits in den sogenannten *class-based generic views* implementiert. Um diese im Projekt zu nutzen, wurde eine Ansicht erstellt, die von der Klasse *DetailView* erbt. Diese wurde angepasst, um die Antwort der Ansicht davon abhängig zu machen, ob das zu betrachtende

Bilderpaar öffentlich und ausgerichtet ist. Sollte es nicht öffentlich sein, so wird nur dem Ersteller Zugriff gewährt. Sollte der Ersteller auf ein nicht ausgerichtetes Bildpaar zugreifen, so wird er per *HTTP*-Umleitung auf die Ausrichtungsansicht umgeleitet. In anderen Fällen wird das Standard-Template genutzt.

## 3.7 Detail-Ansicht eines Bildpaares: Client

Der Nutzer hat in dieser Ansicht die Möglichkeit, die sichtbaren Bereiche der beiden Bilder mit einem Schieberegler-Element gegeneinander zu verschieben und weitere Informationen über das Bildpaar einzusehen (siehe Abbildung 6). Abhängig von dem aktiven Nutzer ermöglicht diese Ansicht auch weitere Aktionen. Ist der aktuelle Nutzer nicht angemeldet, so kann er das entsprechende Bilderpaar und die zugehörigen Daten nur betrachten. Ein angemeldeter Nutzer erhält zudem die Möglichkeit, eine Bewertung abzugeben. Ist der aktuelle Nutzer der Ersteller des Bildes, werden Buttons, die das Bearbeiten der Metadaten, das Neu-Ausrichten der Bilder, das Löschen und das Veröffentlichen/Verstecken des Bilderpaares ermöglichen, eingeblendet.

### 3.7.1 Schieberegler

Dieses Element wird von der JavaScript-Bibliothek *JuxtaposeJS*[21] generiert. Diese stellt beide Bilder in der gleichen Fläche der Seite gleichzeitig dar. Durch Klick oder Ziehen des Schiebereglers kann der Nutzer so bestimmen, wieviel Fläche des Vorher- und des Nachher-Bildes sichtbar ist. *JuxtaposeJS* wurde der vorgeschlagenen Bibliothek *JQuery before/after plugin* [22] vorgezogen, da sie derzeit aktiver entwickelt und von mehreren besucherstarken Internetseiten genutzt wird.

### 3.7.2 Zusätzliche Interaktionsmöglichkeiten

Die Anzeige des Aufnahmeortes wird auch in dieser Ansicht durch das Einbinden eines *Google Maps*-Kartenelements erreicht. Wurde kein Ort festgelegt, wird stattdessen ein Platzhalter eingeblendet.

Das Bearbeiten der Metadaten und das Neu-Ausrichten der Bilder erfolgt durch zusätzliche *Django*-Ansichten auf die Formulare, die bereits bei dem Erstellen des Bilderpaares in den entsprechenden Schritten angezeigt wurden. Das Löschen und die Änderung des Sichtbarkeitsstatus wurden über weitere Ansichten realisiert. Die entsprechenden Funktionen werden nur nach Übermittlung eines HTML-Formulars, das einen Token enthält, durchgeführt. Auf diese Weise wird werden *cross-site request forgery* (CSRF)-Angriffe verhindert. Ohne diesen Zwischenschritt könnte beispielsweise

[21] `https://juxtapose.knightlab.com`
[22] `http://www.catchmyfame.com/catchmyfame-jquery-plugins/jquery-beforeafter-plugin`

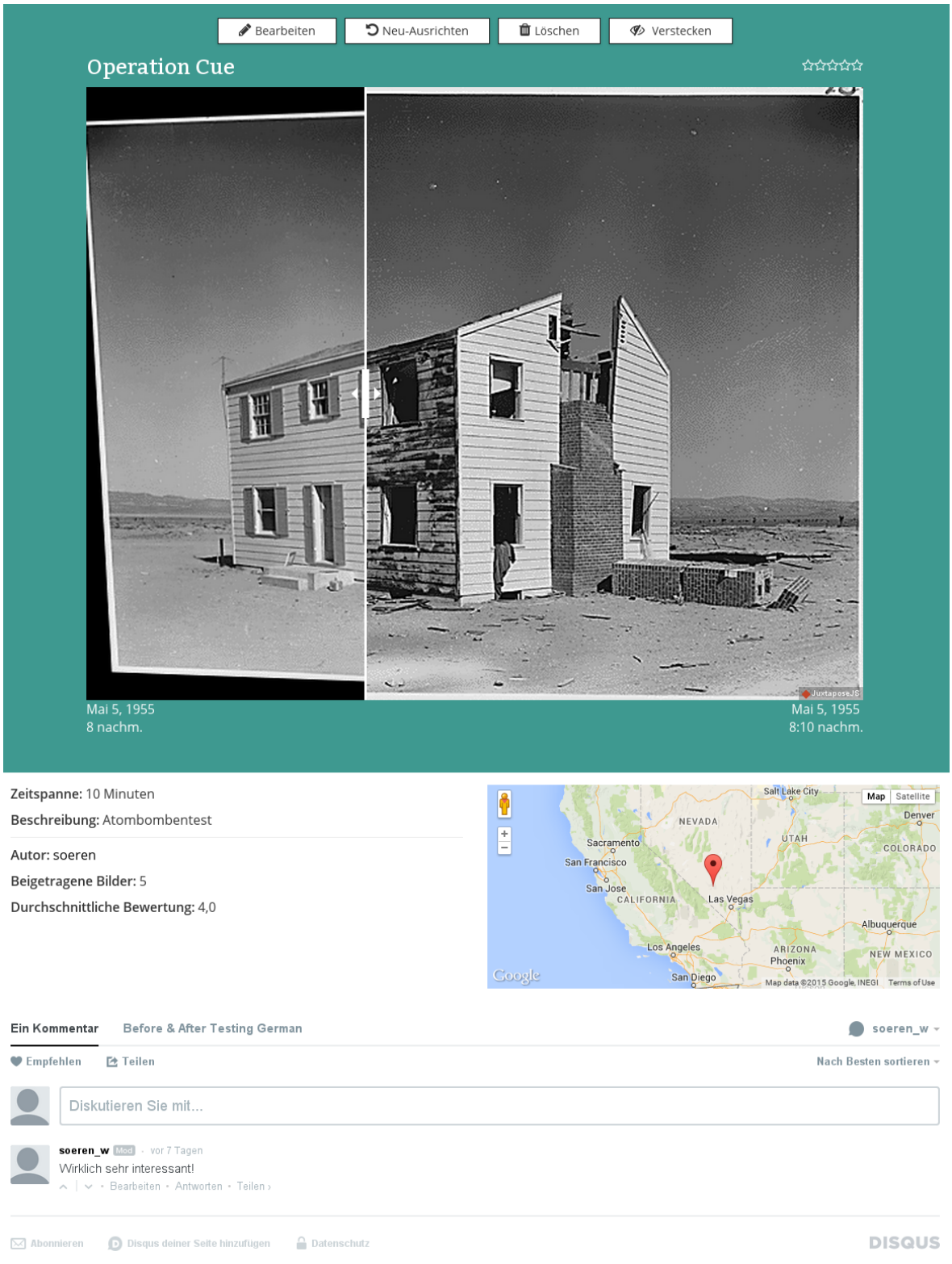


Abbildung 6: Detail-Ansicht eines Bildpaares als Ersteller (Auszug)

die URL zum Löschen eines Bildpaares an den Ersteller gesandt werden. Würde dieser den Link aufrufen, würde das Bildpaar ohne weitere Interaktion gelöscht werden.

### 3.7.3 Kommentarbereich

#### Discourse versus Disqus

Diese Ansicht sollte außerdem eine Möglichkeit bieten, die Bildpaare zu kommentieren. Ursprünglich war für diesen Zweck die Nutzung der Open-Source Forensoftware *Discourse*[23] geplant. Diese Software unterstützt die Moderation des Forums, indem sie automatisch Moderationsrechte an vertraute Nutzer vergeben kann. Bestimmte Moderationstätigkeiten können so aus der Nutzergemeinschaft rekrutiert werden.

Aufgrund von technischen Limitationen der Software war es allerdings nicht möglich, diese in einem sinnvollen Kontext einzusetzen. So muss das Forum beispielsweise über die Wurzeladresse der aufgerufenen Domain erreichbar sein, da es nicht die Anpassung der Ladepfade der Ressourcen unterstützt. In diesem Projekt würde ein Nutzer unter der Wurzeladresse allerdings vermutlich das eigentliche Projekt erwarten und nicht die nachgeordnete Forensoftware. Es wäre auch nicht möglich gewesen, das *Python*-Projekt und die *Discourse*-Instanz auf verschiedenen (Sub-)Domains laufen zu lassen, da das Einbinden der entsprechenden Diskussionsstränge in der Detail-Ansicht sonst durch Sicherheitsbeschränkungen verhindert worden wäre, die das Ausführen von JavaScript einer anderen Domain in demselben Seitenkontext blockieren.

Aus diesen Gründen wurde stattdessen die kostenlose Kommentarumgebung *Disqus*[24] eingesetzt. Diese erlaubt ebenfalls einzelne Kommentarstränge pro Bildpaar, bietet Hilfsmittel zur Moderation und erlaubt das Exportieren der Kommentardaten.

#### Privatsphäre

Da für die Einbindung von *Disqus* das Laden von *JavaScript*-Code von Servern des Unternehmens erforderlich ist, ist es diesem möglich, ein Profil der Besucher zu erstellen – auch wenn diese nicht mit den Kommentaren interagieren. Um dies falls gewünscht zu verhindern, kann in dem Django-Projekt die Einstellungsvariable *DISQUS_LOAD_ON_DEMAND* auf wahr gesetzt werden. In diesem Fall wird das JavaScript-Programm erst nach Knopfdruck des Nutzers angefragt und ausgeführt. Diese Einstellung kann somit die Privatsphäre derjenigen Nutzer schützen, die nicht auf Kommentare zugreifen möchten.

Disqus ermöglicht außerdem ein *Single Sign On* (SSO), sodass bei Nutzung angemeldete Nutzer des Portals auch automatisch in dem Kommentarbereich angemeldet wären. Die Schnittstelle ist gut dokumentiert, wurde allerdings bewusst nicht zum Einsatz gebracht, da sie für diesen Vorgang die E-Mail-Adresse des Nutzers erwartet. Diese

[23] `http://www.discourse.org/`
[24] `https://disqus.com`

Daten würden bei jedem Laden der Kommentare durch einen angemeldeten Nutzer gesendet werden, unabhängig davon, ob dieser Kommentare lesen oder verfassen möchte. Stattdessen wurde der Kommentarbereich so konfiguriert, dass keine Anmeldung oder ein Login über Drittanbieter verpflichtend (aber weiterhin möglich) ist. Über das Aktivieren einer Checkbox kann der Nutzer so als unangemeldeter Gast kommentieren.

#### Mehrsprachigkeit

Die Mehrsprachigkeit des Projektes wurde auch in diesem Bereich beachtet, indem für jede Anzeigesprache ein eigenes Forum über *Disqus* eingerichtet wurde. Dieser Trennung wurde gewählt, um den Nutzern mit hoher Wahrscheinlichkeit Kommentare anzuzeigen, die für ihn verständlich sind. Abhängig von der aktuell ausgewählten Sprache wird das jeweilige Forum eingebunden. Die Projekt-Einstellungsvariable *DISQUS_WEBSITE_SHORTNAMES* erlaubt das Zuordnen von Sprachcode zu einem *Disqus*-Forum. Nutzer, die die Kommentare in mehreren Sprachen lesen möchten, können dies durch das kurzzeitige Umschalten der Anzeigesprache erreichen, das auf jeder Unterseite möglich ist.

## 3.8 Bewertung: Server

Eine weitere zentrale Anforderung an das Projekt war die Sortierung von Bildpaaren nach Nutzerbewertungen. Eine solche Sortierung könnte das arithmetische Mittel der Bewertungen als Sortierkriterium nutzen, allerdings besitzt dieses bei einer kleinen Anzahl von Bewertungen ungewünschte Eigenschaften. Ein Bildpaar, das ein Mal mit fünf Sternen bewertet wurde, würde in diesem Fall an höherer Stelle angezeigt werden als ein Bildpaar, das zehn Mal mit fünf Sternen und ein Mal mit vier Sternen bewertet wurde. Dies entspricht nicht dem intuitiven Verständnis, da mit einer höheren Anzahl von Bewertungen meist auch eine höhere Glaubwürdigkeit des Durchschnitts einhergeht.

Es wurde daher ein Sortierkriterium implementiert[25], das sich im Falle kleiner Bewertungsanzahlen konservativer verhält. Dieses Kriterium wird nur zur Sortierung der Ergebnisse verwendet und ist für den Nutzer nicht sichtbar. Wenn sich die Breite des glaubwürdigen Intervalls auf unter einen Stern reduziert hat, wird dem Nutzer das aktuelle arithmetische Mittel angezeigt. Das Sortierkriterium nähert sich diesem für hohe Bewertungsanzahlen an.

Die Abgabe einer Bewertung setzt eine Authentifizierung voraus und wird außerdem verhindert, falls der aktuelle Nutzer auch der Ersteller des Bildpaares ist.

[25] `http://www.evanmiller.org/ranking-items-with-star-ratings.html`

### 3.8.1 Django-Ansichten

Die Annahme einer Bewertung eines Nutzers nutzt die Ansicht *RatingUpdateCreateView*. Diese Ansicht nimmt ausschließlich HTTP *POST*-Anfragen entgegen, da die Bewertung mithilfe eines *HTML*-Formulars erfolgen sollte, um die Möglichkeit von *CSRF* durch einen angehängten Token zu minimieren. Die Ansicht prüft zunächst, ob das übermittelte Formular valide ist. Das genutzte Formular besteht aus einem Feld, das die Sternanzahl enthält. Dieses Feld gilt als valide, falls dessen Wert zwischen (und einschließlich) 1 und 5 liegt. Ist dies nicht der Fall wird der Nutzer auf die Detail-Seite des Bildpaares zurück umgeleitet und eine entsprechende Nachricht wird eingeblendet. Daraufhin wird geprüft, ob der aktuelle Nutzer der Ersteller des Bildpaares ist. Ist dies der Fall wird die Bewertung auf dem bereits beschriebenen Weg abgelehnt. Falls die Bewertung akzeptiert wird, wird diese über die *Django*-Manager-Funktion *update_or_create* in der Datenbank abgelegt.

## 3.9 Bewertung: Client

Die statische Ausgabe von einem Bewertungsdurchschnitt wurde über eine Schleife in den *Django*-Templates realisiert, die fünf Stern-Symbole aus der *Font Awesome*-Bibliothek[26] zeichnet und je nach tatsächlichem Durchschnitt einfärbt.

Die JavaScript-Bibliothek *bootstrap-star-rating*[27] wurde für die Anzeige und das Einstellen der eigenen Bewertung genutzt. Das Abgeben der Bewertung eines Bildpaares erfolgt über einen Klick auf die grafische Anzahl der Sterne, die der Nutzer vergeben möchte. Trifft er eine Auswahl, so wird der Wert in einem *HTML5*-Nummerneingabefeld gesetzt und das umschließende HTML-Formular abgeschickt. Auch hier enthält das Formular einen Token, der das *CSRF*-Risiko minimiert. Im Falle einer bereits abgegebenen Bewertung wird deren Wert zur Ladezeit in dem *Django*-Template gesetzt. Der Nutzer kann so sehen, ob und wie er ein Bildpaar bereits bewertet hat.

## 3.10 Stöbern: Server

Um dem Nutzer das Erkunden der Daten mit möglichst geringer Verzögerung zu ermöglichen, wurden alle dynamischen Teile dieser Ansicht mithilfe von *JavaScript* implementiert. Durch asynchrone Anfragen an die API (siehe Abschnitt 3) muss die Seite nicht für jede Änderung an den Parametern vollständig neu geladen werden. Die relevanten Teile der Ansicht werden nach Antwort über die API im Browser des Nutzers neu generiert.

[26] `http://fortawesome.github.io/Font-Awesome`
[27] `http://plugins.krajee.com/star-rating`

### 3.10.1 Django-Ansicht

Diese Funktion wird von der *Django*-Ansicht *CompilationBrowseView* realisiert. Diese Ansicht berechnet ein Template, das die Grundstruktur der Seite, sowie Verweise auf den für die Funktion benötigten *JavaScript*-Code enthält. Innerhalb des Templates werden *JavaScript*-Variablen gesetzt, die von dem nachgeladenen Code benötigt werden, beispielsweise die URL des API-Endpunktes, der für die Durchsuchung der Ergebnisse verantwortlich ist.

### 3.10.2 Geografische Filterung

Die Ansicht zum Durchsuchen der vorhandenen Bildpaare sollte auch die Filterung nach geografischen Parametern ermöglichen. Hierzu wurde die in Django integrierte Bibliothek *GeoDjango* genutzt. In Kombination mit der erwähnten *PostgreSQL*-Erweiterung *PostGIS*, erlaubt *GeoDjango* das performante Berechnen von (komplexen) geografischen Abfragen. Diese Bibliothek eignet sich auch hinsichtlich zukünftiger Erweiterungen des Projektes. So wäre es beispielsweise möglich, Ländergrenzen in die Datenbank zu importieren und dem Nutzer dann die Filterung der Bildpaare nach Ursprungsland zu ermöglichen.

Um die korrekte Filterung der Suchergebnisse nach einem bestimmten Kartenausschnitt zu implementieren, mussten die Daten, die *Google Maps* zurück gibt vorverarbeitet werden. Auf Funktionsaufruf von *getBounds* gibt die *Google Maps JavaScript API* die geografischen Koordinaten der Eckpunkte des sichtbaren Kartenbereichs zurück. Bei einer niedrigen Vergrößerungsstufe ist es hier möglich, dass die sichtbaren Längengrade den Anti-Meridian enthalten. In diesem Fall ist es möglich, dass der Längengrad auf der westlichen Seite des Ausschnitts positiv ist, während der Längengrad der östlichen Seite negativ ist. Ein Polygon mit diesen Eigenschaften ist in *GeoDjango* nicht definiert und resultiert in Rückgabe aller Bildpaare, die sich außerhalb des Polygons befinden.

Um sicher zu stellen, dass die richtigen Ergebnisse auf jede Filterungsanfrage zurück gegeben werden, wird ein Polygon mit den oben beschriebenen Eigenschaften an dem Antimeridian in zwei Polygone aufgeteilt (siehe Listing 4). *GeoDjango* erlaubt das Zusammenfassen mehrerer Polygone zu einem neuen Polygon, auf dem dann dieselben Abfragen ausgeführt werden können.

```
# Watching out for the antimeridian
if bounds[0] > bounds[2]:
    polygon1 = Polygon.from_bbox((bounds[0], bounds[1], 180, bounds[3]))
    polygon2 = Polygon.from_bbox((-180, bounds[1], bounds[2], bounds[3]))
    polygon = MultiPolygon(polygon1, polygon2)
else:
    polygon = Polygon.from_bbox(bounds)
```

Listing 4: Behandlung bei Überschreitung des Antimeridians

## 3.11 Stöbern: Client

Diese Ansicht bietet mehrere Wege, die Bildpaare nach eigenen Vorstellungen zu filtern und zu sortieren (siehe Abbildung 7). Alle Filter sind für den Nutzer gleichzeitig nutzbar und werden bei der Auswertung grundsätzlich und-verknüpft betrachtet. Auf diesem Wege kann der Nutzer die Ergebnismenge durch seine Auswahl einschränken. Die meisten Filter-Bedienelemente lösen bei Veränderung durch den Nutzer über *JavaScript*-Ereignisse automatisch eine neue Abfrage an die API aus. Bei deren Antwort wird die Ansicht ebenso automatisch aktualisiert.

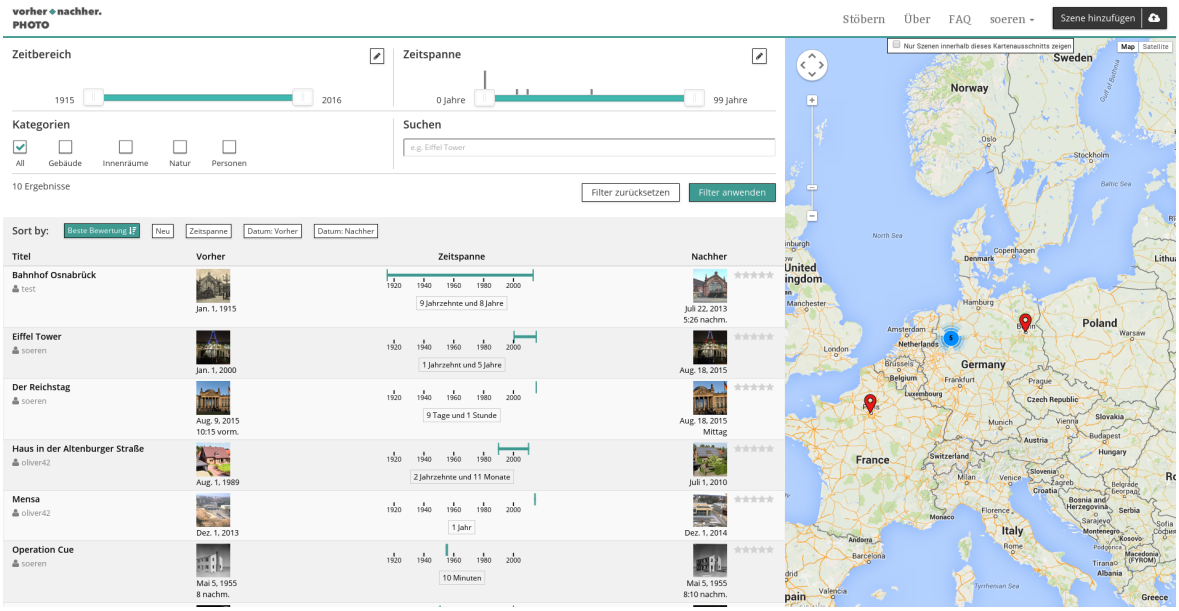


Abbildung 7: Stöbern-Ansicht

### 3.11.1 Filter

#### Filterung nach Karte

Zur Anzeige der Karte wurde die *Google Maps JavaScript API* genutzt. Über das Aktivieren einer Checkbox im Kartenelement kann die Filterung nach sichtbarem Kartenausschnitt durch den Nutzer zugeschaltet werden. In diesem Fall werden bei jeder

Anfrage an die API die Geo-Koordinaten des sichtbaren Kartenrechtecks zur Eingrenzung der Ergebnisse genutzt. Ist diese Funktion aktiviert, werden Bildpaare ohne zugeordnete Geodaten nicht mehr eingeblendet, da sie nicht Teil der Kartenansicht sind.

### Filterung nach Zeit

Der Nutzer sollte laut Vorgaben die Möglichkeit haben, sowohl den Zeitraum, in dem beide Bilder eines Bildpaares entstanden sein sollen, als auch die Zeitspanne zwischen beiden Bildern einzugrenzen. Für eine solche Eingrenzung bestimmt der Nutzer jeweils Minimum und Maximum der gewünschten Spanne. Dies wurde mithilfe eines Schieberegler-Elements realisiert, das zwei Griffe hat und damit das Einstellen beider Werte ermöglicht. Dieses Element wird von der *JavaScript*-Bibliothek *noUiSlider*[28] generiert.

Über dem Schieberegler der Zeitspanne wurde ein Histogramm der aktuellen Suchergebnisse unter Einsatz der *JavaScript*-Bibliothek *D3.js*[29] implementiert. Da dieser Graph keine Funktion der eingesetzten Bibliothek war, wird auf diesem Wege auf Nutzerseite ein *SVG*-Objekt generiert, das auf die Schrittweite des Reglers angepasst ist. Es wird eine maximale Anzahl von Balken generiert, um die Lesbarkeit der Grafik zu erhalten. Die Daten zur Erstellung des Histogramms sind in jeder Antwort der API auf eine Bildpaar-Suche enthalten.

Statt der Schieberegler kann der Nutzer auf Knopfdruck auch Textfelder einblenden, die eine direktere Eingabe der Suchdaten ermöglichen. Die Filterung des gesamten Zeitraums ist auf diesem Wege auch tageweise möglich, ohne dass die Schieberegler aufgrund einer sehr kleinen Schrittweite schlecht nutzbar würden.

### Filterung nach Kategorien

Es ist dem Nutzer möglich, über das Aktivieren von Checkboxen eine Auswahl der Bildpaar-Kategorien zu treffen, die in die Ergebnismenge einbezogen werden sollen. Diese Auswahl ist oder-verknüpft, da ein Bildpaar derzeit nur eine Kategorie besitzen kann. Um das Verhalten dieser Elemente zu verdeutlichen wird eine zusätzliche Checkbox eingeblendet, deren Aktivierung alle verfügbaren Kategorien mit in die Suche einbezieht. Wird diese Checkbox durch den Nutzer aktiviert, werden alle sonstigen Kategorie-Checkboxen deaktiviert. Umgekehrt führt die Auswahl einer oder mehrerer Kategorien zu der Deaktivierung dieses Bedienelements.

### Filterung nach Zeichenkette

Ein Textfeld ermöglicht das Filtern von Bildpaaren nach enthaltenen Zeichenketten in Titel und Beschreibung. Dieses Feld löst nicht automatisch bei einer Änderung eine neue Anfrage an die API aus, um Anfragen zu vermeiden, die nicht zur Darstellung

[28] `http://refreshless.com/nouislider`
[29] `http://d3js.org`

kommen, da der Nutzer bereits ein weiteres Zeichen eingegeben hat. Die entsprechende Anfrage kann in diesem Fall durch das Betätigen der Eingabetaste oder des Knopfes zur Aktualisierung der Ergebnisse gestellt werden. Die Suche umfasst sämtliche verfügbare Sprachen der Felder. So kann ein englischsprachiger Nutzer auch Stichworte in deutschen Beschreibungen auffinden.

### 3.11.2 Sortierung

Die Sortierung der Ergebnisse wird über mehrere Knöpfe gesteuert. Mehrfacher Klick auf einen Knopf erlaubt das Wechseln der Sortierrichtung. Es kann derzeit stets nur ein Sortierkriterium gesetzt sein. Der aktuelle Status eines solchen Knopfes wird durch seine *HTML*-Klasse gespeichert und ausgelesen. Per *CSS* werden entsprechende, erklärende Symbole für Elemente der jeweiligen Klasse eingeblendet. Per *JavaScript* werden bei einer neuen Auswahl die Klassen der anderen Knöpfe entfernt und für den aktivierten Knopf entsprechend neu gesetzt.

### 3.11.3 Abfrage der Ergebnisse

Falls Filter- oder Sortierparameter geändert wurden, wird eine HTTP-Anfrage an die API gesendet. Diese enthält die jeweiligen gewünschten Einschränkungen der Ergebnismenge als *GET*-Parameter. Hier werden die Werte der jeweiligen Filter nur beachtet, falls diese die Ergebnismenge einschränken. Standardmäßig sind alle Filter so eingestellt, dass alle Bildpaar umfasst werden. Die Antwort der API erfolgt in dem Datenformat *JSON* und enthält sowohl serialisierte Bildpaar-Objekte, als auch weitere Metadaten. Teil der API ist eine automatische Paginierung. Sollten sehr viele Ergebnisse auf eine Abfrage zutreffen, werden die Ergebnisse in Seiten von festgelegter Länge ausgeliefert. Diese Funktion wurde aktiviert, um die Ladezeiten der Ergebnisse gering zu halten und somit die Nutzeroberfläche schnell auf Änderungen reagieren lassen zu können. Die *JSON*-Antwort enthält für die Paginierung Links zu der jeweils vorherigen und nächsten Seite. Der Nutzer kann diese Funktion kontrollieren, indem er zwei Knöpfe zum Vor- und Zurückschalten betätigt.

### 3.11.4 Anzeige der Ergebnisse

#### Tabelle

Anhand der Antwort der API wird die *HTML*-Tabelle, die die Ergebnisse enthält, zunächst geleert und dann dynamisch mit den aktuellen Ergebnisse neu generiert. Dies wurde über eine unsichtbare Tabellenzeile realisiert, die bei Bedarf geklont, neu positioniert und mit Daten gefüllt wird. Für jedes Bildpaar werden unter anderem serverseitig generierte Thumbnails der Bilder sowie eine Zeitleiste präsentiert. Die Zeitleiste wird für jedes Bildpaar mithilfe der *JavaScript*-Bibliothek *D3.js* generiert und visualisiert

die Zeitspanne zwischen Vorher- und Nachher-Bild auf einer Zeitachse, die mit den aktuell dargestellten Bildpaaren skaliert.

#### Karte

Bildpaare, die in der aktuellen Ergebnismenge enthalten sind und denen ein Aufnahmeort zugeordnet wurde, werden auf der Karte mithilfe von Markern visualisiert. Die Marker werden mit den korrespondierenden Tabellenzeilen über *JavaScript*-Ereignisse so verknüpft, dass die jeweiligen Gegenstücke hervorgehoben werden, falls der Nutzer mit dem Mauszeiger über sie fährt. Per Klick auf einen Marker wird außerdem eine kleine Einblendung generiert, die den Titel und einen Link zu der Detail-Ansicht des Bildpaares enthält.

Da die Darstellung von vielen Karten-Markern unübersichtlich werden kann, wird die Bibliothek *Marker Clusterer*[30] eingesetzt, die mehrere nah beieinander liegende Marker zu einem neuen Symbol zusammenfasst. Bei Klick auf einen solchen Cluster wird die Vergrößerungsstufe der Karte angepasst bis die zusammengefassten Elemente erkennbar sind.

## 3.12 Application Programming Interface: Server

Um sowohl die Erstellung von Bildpaaren aus der *iOS-App* heraus, als auch das asynchrone Nachladen der Suchergebnisse per *JavaScript* zu ermöglichen, müssen Teile dieses Projektes über eine definierte, maschinennutzbare Schnittstelle verfügen. Das hierzu erstellte *Application Programming Interface* (API) ist wie das Projekt selber per HTTP erreichbar und erlaubt nur definierte Abfragen oder Änderungen auf dem Datenbestand.

Die API wurde unter Einsatz des *Django REST Frameworks*[31] realisiert. Dieses Framework unterstützt einerseits bei der Gestaltung der De-/Serialisierung von Objekten und bietet andererseits mit sinnvollen Standardeinstellungen versehene *Django*-Ansichten. Es erlaubt unter anderem die Nutzung als „Browsable API“: hier ist die API über ein Web-Interface direkt durch einen Menschen nutz- und testbar.

Zur Filterung der Bildpaare wurde die Applikation *django-filter* eingesetzt, die sich mit dem *Django REST Framework* verknüpfen lässt. Sie ermöglicht das Definieren und Ausführen von Filtern und Sortierungen über *Django QuerySets*, welche eine Abstraktion von Datenbankergebnissen in *Django* darstellen. Die Filter- und Sortierparameter werden hier von der API übergeben.

Die API wurde durch eine einzelne *Django*-Ansicht realisiert, die sowohl das Abfragen (inklusive Filterung und Sortierung), als auch das Erstellen von Bildpaaren unter einem URL-Endpunkt erlaubt. Die verschiedenen Funktionen werden über un-

[30] `https://github.com/googlemaps/js-marker-clusterer`
[31] `http://www.django-rest-framework.org`

terschiedliche HTTP-Anfragetypen angesteuert. Die Antwort erfolgt im produktiven Einsatz stets über das Datenformat *JSON*, im testweisen Betrieb allerdings auch über das Webinterface der API.

### 3.12.1 Abfrage

Zur Abfrage wird eine *GET*-Anfrage an den Endpunkt gestellt. Per *GET*-Parameter können hier die angewandten Filter, Sortierungen und Suchen gesteuert werden. Sind keine *GET*-Parameter in der Abfrage enthalten, werden alle veröffentlichten Bildpaare als Ergebnismenge angesehen und absteigend nach dem Bewertungs-Sortierkriterium sortiert ausgegeben.

Die Daten in der Antwort einer solchen Abfrage sind lokalisiert, um diese Aufgabe nicht im Browser des Nutzers lösen zu müssen. Übersetzbare Felder werden in der aktuellen Sprache der Sitzung ausgegeben. Ebenso werden die Zeitangaben auf Sprache und eventuell gesetzte Zeitzone des Nutzers angepasst. Die zu verwendende Sprache wird auf demselben Wege bestimmt wie für die anderen Bereiche des Portals auch (siehe Abschnitt 3.1.3).

### 3.12.2 Erstellung

Zur Erstellung eines Bildpaares wird eine authentifizierte *POST*-Anfrage, die sämtliche benötigte Daten enthält, genutzt. Die Authentifizierung gegenüber der API erfolgt über einen nutzerspezifischen Token im *Authorization* HTTP-Header. Dieser Token kann über einen zweiten Endpunkt unter Angabe von Nutzernamen und Passwort abgefragt werden. Um zu verhindern, dass Login-Daten oder der Token Dritten bekannt werden können, sollte dieses Projekt im Einsatz deshalb ausschließlich über eine verschlüsselte HTTPS-Verbindung erreichbar sein.

Nach Absprache mit dem Entwickler der *iOS-App* wurde festgehalten, dass die Applikation die Ausrichtung der Bilder bereits vorgenommen hat, wenn diese in das Portal geladen werden können. In der POST-Anfrage werden deshalb zwei originale und zwei ausgerichtete Bilddateien erwartet. Die ausgerichteten Bilder müssen pixelweise dieselbe Größe aufweisen.

Titel und Beschreibung werden in der Sprache der Abfrage abgelegt, die in dem HTTP-Header *Accept-Language* bestimmt werden kann.

Hier angegebene Zeitpunkte werden standardmäßig als UTC-Zeiten interpretiert. Falls die entsprechenden Bilder in einer anderen Zeitzone entstanden sein sollten, müssen die Zeitangaben vorher in der Applikation in die koordinierte Weltzeit umgerechnet werden..

### 3.12.3 Serialisierung

Um Objekte wie gewünscht im *JSON*-Format beschreiben oder aus einer *POST*-Anfrage erstellen zu können, muss die De-/Serialisierung für diese Objekte definiert werden. In dem *Django REST Framework* geschieht dies durch das Erben der Klasse *Serializer* und dem Definieren von zu verarbeitenden Datenfeldern, sehr ähnlich dem Festlegen von Datenfeldern in *Django*-Modellen.

#### Compilation

```
class CompilationSerializer(serializers.ModelSerializer):
    creator = serializers.ReadOnlyField(source='creator.username')
    creation_time = AwareDateTimeField(read_only=True)
    description = TruncatedTextField()
    image_before = ImageSerializer()
    image_after = ImageSerializer()
    rating_average = serializers.IntegerField(read_only=True)
    timespan = serializers.ReadOnlyField(source='get_humanized_timespan')
    url = serializers.SerializerMethodField()
    is_public = serializers.BooleanField(write_only=True)

    class Meta:
        model = Compilation
        fields = (
            'creator', 'title', 'description', 'category', '
                rating_average', 'position', 'creation_time', '
                image_before', 'image_after', 'timespan', 'url', '
                is_public'
        )
```

Listing 5: Serialisierer des Compilation-Modells (Auszug)

Der Serialisierer des *Compilation*-Modells (siehe Listing 5) beschreibt, welche Daten einer Compilation-Instanz les- und beschreibbar sind und wie diese auszugeben sind. Die Felder *creator*, *creation_time* und *rating_average* werden nur bei lesenden Abfragen beachtet, da sie von der Model-Instanz selbst verwaltet werden. Die Beschreibung des Bildpaares wird über einen eigenen Feldtypen gekürzt ausgegeben (kann aber vollständig beschrieben werden). Das erlaubt es, die Datenmengen bei der Übermittlung von Suchergebnissen zu minimieren, da die Beschreibungen in dieser Ansicht ohnehin nicht vollständig sichtbar sind. Das Feld *is_public* ist bei lesenden Zugriffen nicht sichtbar (da es dort eh stets aktiv wäre), dafür aber bei schreibenden Zugriffen setzbar.

**Image**

```
class ImageSerializer(serializers.ModelSerializer):
    original_file = serializers.ImageField(write_only=True, required=True
        )
    aligned_file = serializers.ImageField(write_only=True, required=True)
    original_file_thumb = serializers.ImageField(read_only=True)
    creation_datetime = serializers.DateTimeField(required=True)
    creation_date = AwareDateField(source='creation_datetime')
    creation_time = AwareTimeField(source='creation_datetime')

    class Meta:
        model = Image
        fields = (
            'original_file', 'aligned_file', 'original_file_thumb', '
                creation_datetime', 'creation_date', 'creation_time', '
                creation_time_set'
        )
```

Listing 6: Serialisierer des Image-Modells

Der Serialisierer des *Image*-Modells (siehe Listing 6) legt fest, dass die Bilddateien der Original- und der ausgerichteten Dateien nicht lesbar, aber beschreibbar sind. Beide Dateien werden in der Ergebnisansicht nicht angezeigt, und daher nicht ausgegeben. Das Feld, das eine verkleinerte Ansicht des Originalbildes (Thumbnail) enthält, wird in der Ergebnisansicht präsentiert und deshalb auch in der API ausgegeben. Da dieses Feld die Thumbnails automatisch generiert, ist der Schreibzugriff hier deaktiviert.

Zeitinformationen werden über eigens angelegte Feldtypen lokalisiert ausgegeben, da diese Informationen ansonsten in der koordinierten Weltzeit ausgegeben würden. Die Feldtypen *AwareDateField* und *AwareTimeField* (siehe Listing 7) erben von *ReadOnlyField* und erlauben somit nur einen lesbaren Zugriff. Die Methode *to_representation* lest fest, wie die Serialisierung des Modell-Datenfeldes vorgenommen werden soll. In diesem Fall wird das Modul *timezone* aus dem Paket *django.utils* genutzt. Dessen Methode *localtime* stellt die aktuell genutzte Zeitzone fest und lokalisiert dann das übergebene Zeit-Objekt. Die weitere Serialisierung wird daraufhin derselben Methode in der Elternklasse überlassen.

```
class AwareDateField(serializers.ReadOnlyField):
    def to_representation(self, value):
        value = timezone.localtime(value)
        return super(AwareDateField, self).to_representation(
            defaultfilters.date(value))

class AwareTimeField(serializers.ReadOnlyField):
    def to_representation(self, value):
        value = timezone.localtime(value)
        return super(AwareTimeField, self).to_representation(
            defaultfilters.time(value))
```

Listing 7: Serializer-Feldtypen AwareDateField & AwareTimeField

# Kapitel 4

# Ausblick

Zusätzlich zu den gegebenen Vorgaben wurden im Rahmen dieser Bachelor-Arbeit zahlreiche weitere Ideen zur Verbesserung des Projekts implementiert. Dennoch gibt es noch einige weitere vielversprechende Entwicklungsmöglichkeiten, die im folgenden kurz dargestellt werden sollen.

## 4.1 Nutzung auf Mobilgeräten

Aufgrund der Nutzung des *Bootstrap*-Frameworks wäre es grundsätzlich möglich, die Darstellung des Portals responsiv, das heißt auf mehrere Gerätetypen angepasst, zu gestalten. Während der Konzeption und Entwicklung dieses Projektes wurde hierzu darauf geachtet, dass alle verwendeten Interaktionselemente kompatibel mit der Bedienung per Touchscreen sind. Die notwendigen Anpassungen und Tests für ein responsives Design waren im Rahmen dieser Arbeit nicht vorgesehen, lassen sich aber zukünftig aber nachrüsten.

## 4.2 Nutzer-Tracking

Sollte die Aufzeichnung und Auswertung von Besuchs- und Nutzungsdaten des Portals gewünscht sein, kann dies ohne weitere Probleme implementiert werden. Die Einbindung einer solchen Analyse-Software erfolgt meist über das Laden einer zusätzlichen JavaScript-Datei im Browser des Nutzers. Durch eine Änderung des *Django*-Templates 'compilations/base.html' könnte diese Datei für jeden Seitenaufruf innerhalb des Projektes geladen werden. Es existieren mehrere Lösungen in diesem Bereich, die jeweils mit Vor- und Nachteilen einhergehen. Die beiden bekanntesten Lösungen, die an dieser Stelle beispielhaft zu nennen sind, sind *Piwik*[32] (Open Source, eigene Instanz), sowie *Google Analytics*[33] (proprietär, Drittanbieter). *Piwik* erlaubt dem Betreiber Kontrolle über die gesammelten Daten und überlässt dem Administrator mehr Rechte, während

[32] `http://piwik.org`
[33] `http://www.google.com/intl/de_de/analytics`

*Google Analytics* ein ausgereifteres und übersichtlicheres Nutzer-Interface bietet, dafür aber Zugriff auf die Nutzerdaten hat und diese zu vermarkten sucht.

## 4.3 Content Delivery Network

Zur Verbesserung der Server-Sicherheit und der Ladezeiten des Nutzers bietet sich die Migration der Bild-Verarbeitung und -Auslieferung in ein *Content Delivery Network* (CDN) an. Der Upload, sowie die Bildbearbeitung würden in diesem Fall ausschließlich in dem CDN statt finden. Hierdurch wäre der Applikations-Server nicht mehr dem Sicherheitsrisiko der hochgeladenen Dateien ausgesetzt. Da ein CDN seine Inhalte geografisch verteilt, wären die Ladezeiten der Bilder für weiter entfernte Nutzer außerdem verkürzt. Es existieren Anbieter, die *Django*-kompatible Bibliotheken zur Nutzung ihrer Dienstleistungen anbieten. Diese erlauben derzeit allerdings nur simplere Bildbearbeitungen auf ihrer Seite und nicht das Anwenden von perspektivischen Transformationen.

Um nur den Vorteil kürzerer Ladezeiten aufgrund der geografischen Verteilung eines CDN zu erreichen, könnte die Bildverarbeitung weiterhin auf einem eigenen Server stattfinden, während die Auslieferung über das CDN läuft. Das beschriebene Sicherheitsrisiko würde in diesem Fall weiter bestehen.

## 4.4 Bildformate

Die unterstützten Bildformate dieses Projekte beschränken sich derzeit auf die weiter verbreiteten Formate PNG und JPEG. Diese Beschränkung wurde programmatisch umgesetzt, da *OpenCV* nur eine kleinere Auswahl von Formaten in der Form unterstützt, die für das Projekt benötigt wird. Es wäre möglich, das Projekt dahingehend zu erweitern, dass hochgeladene und eigentlich mit *OpenCV* inkompatible Bilddateien automatisch in ein unterstütztes Format überführt werden. Dies könnte beispielsweise über die *Python*-Anbindung der Bildbearbeitungsbibliothek *ImageMagick*[34] umgesetzt werden.

## 4.5 Lizenzen der Bilder

Um Besuchern des Portals ihre Rechte an den Bildern zu vermitteln, könnte im Erstellunsgprozess zusätzlich die urheberrechtliche Lizenzsituation abgefragt werden. So könnten in der Detail-Ansicht die Nutzer darüber aufgeklärt werden, ob und in welchem Umfang sie die Bilder weiter verbreiten oder verändern dürfen Diese Rechte könnten in Piktogrammen dargestellt werden (vergleiche *Creative Commons*[35]), um sie

[34] `http://www.imagemagick.org`
[35] `http://de.creativecommons.org`

zugänglicher zu machen.

Diese Funktion könnte beispielsweise zur Wiederverwendung alter Aufnahmen oder Zeichnungen in eigenen Bildpaaren anregen.

## 4.6 Meldemöglichkeit

Durch die Möglichkeit einen Regelverstoß eines Bildpaares zu melden, könnte die Moderation erleichtert werden und potentiell schneller erfolgen. In bestimmten Konstellationen kann der Betreiber einer Internetseite auch für die Inhalte haftbar gemacht werden, die von Dritten erstellt wurden. Hier ist beispielsweise auf urheberrechtlich geschützte Werke, pornografisches Material und persönliche Beleidigungen zu achten.

# Kapitel 5

# Fazit

Alles in allem kann festgehalten werden, dass die verpflichtenden Vorgaben des Projektes erfolgreich umgesetzt werden konnten. In einem dreiteilligen Formular können Nutzer zunächst zwei Bilder einer Szene auswählen und hochladen, diese danach mit Meta-Daten (z.B. einem Ort, den Aufnahme-Daten sowie einer Kategorie) versehen und sie anschließend mit Hilfe von Markern ausrichten. Eine ausführliche Stöbern-Ansicht bietet weiterhin die Möglichkeit, sämtliche Bildpaare nach Zeitbereich, Zeitspanne, Kategorie und Stichworten zu filtern. Gleichzeitig werden hier die Ergebnisse als Marker beziehungsweise als Cluster auf einer Karte dargestellt und auf Wunsch des Nutzers auch auf den aktuellen Kartenausschnitt begrenzt. Die implementierten Sortiermöglichkeiten der Ergebnistabelle (z.B. nach Erstellungsdatum, Bewertung oder Zeitspanne) erlauben eine gezielte Auswahl bestimmter Bildpaare. Eine Detailansicht stellt ein ausgewähltes Bildpaar schließlich vergleichend dar. Der produktive Einsatz des Projektes ist somit in derzeitiger Form bereits möglich.

Neben dem Umsetzen dieser Vorgaben wurden weitere Funktionen implementiert, die die Benutzbarkeit des Internetportals verbessern. Ein Beispiel hierfür ist die Darstellung der vorhandenen Ergebnisse als Histogramm - positioniert oberhalb des Schiebereglers der Zeitspanne - auf der Stöbern-Ansicht. Diese Funktion gibt dem Nutzer ein intuitives Feedback und vereinfacht die Suche und das Filtern von Bildpaaren. Auch die Visualisierung der Zeit zwischen den beiden Bildern in der Ergebnistabelle trägt dazu bei, dass Nutzer die vorhandenen Daten schneller erfassen und verstehen können.

Selbstverständlich hat jedoch auch dieses Projekt noch einige Schwachstellen. Der Algorithmus zum Beschneiden der ausgerichteten Bilder ist momentan nicht optimal, da er für stark verzerrte Bilder kein nutzbares Resultat liefern kann und nicht grundsätzlich das größtmögliche Rechteck wählt. Außerdem wurde das Internetportal bislang nur für den Browser Chrome ausführlich getestet. Weitere Browser wie Firefox oder Safari sollten zwar alle grundlegenden Funktionen darstellen können, aber zeigen das Design eventuell an einigen Stellen falsch an. Bevor das Portal für Nutzer öffentlich zugänglich gemacht werden kann, sollten weitere Unterseiten (zum Beispiel eine Seite, die die Funktionsweise des Portals erklärt) sowie die Startseite mit Inhalt gefüllt

werden.

# Kapitel 6

# Declaration of Authorship

I hereby certify that the work presented here is, to the best of my knowledge and belief, original and the result of my own investigations, except as acknowledged, and has not been submitted, either in part or whole, for a degree at this or any other university.

---

Osnabrück, den 28.08.2015