

# Konzeption und prototypische Entwicklung einer webbasierten Anwendung zur Aggregation und Analyse wissenschaftlicher Daten aus dem Argo-Projekt

Sebastian Schmid

S0543196

Prof. Dr. Christin Schmidt

Prof. Dr.-Ing. Hendrik Gärtner

Bachelorarbeit zur Erlangung des akademischen Grades

Bachelor of Science (B.Sc.)

Fachbereich Wirtschaftswissenschaften II

Studiengang Angewandte Informatik

an der Hochschule für Technik und Wirtschaft Berlin

## Erklärung der Urheberschaft

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Ort, Datum

Unterschrift

## Danksagung

An dieser Stelle möchte ich meiner Betreuerin Frau Prof. Dr. Christin Schmidt danken. Diese unterstützte mich stets tatkräftig. Auch Herrn Prof. Dr.-Ing. Gärtner möchte ich dafür danken, dass er sich dazu bereit erklärt hat, diese Arbeit zu betreuen. Allen Teilnehmern an der Usability-Umfrage möchte ich ebenso meinen Dank aussprechen.

Auch wenn diese heute nicht mehr unter uns sind, so waren es doch meine Eltern, die den Keim gelegt haben, der mich dazu befähigte, meinen Weg bis zu dieser Stelle zu gehen. Meine Freundin begleitete mich auf dem Weg dieser Ausarbeitung und stand mir stets mit Rat und Tat zur Seite. Da diese in ganz besonderer Weise an dieser ereignisreichen Zeit teilhaben durfte, ist ihr ganz besonders zu danken.

Zu danken habe ich auch Robert M. Pirsig. Sein Buch „Zen und die Kunst ein Motorrad zu warten“ begleitete mich bei der Entwicklung dieser Arbeit und inspirierte mich immer wieder aufs Neue. Ich könnte mir kein besseres Buch als Begleitung für eine schriftliche Ausarbeitung vorstellen.

# Inhalt

<b>Abbildungsverzeichnis</b>	<b>iv</b>
<b>Quellcodeverzeichnis</b>	<b>v</b>
<b>Tabellenverzeichnis</b>	<b>v</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Existierende Lösungen . . . . .	3
1.2 Alleinstellungsmerkmal & Abgrenzung . . . . .	3
<b>2 Grundlagen</b>	<b>4</b>
2.1 Argo-Programm . . . . .	4
2.2 Datengrundlage . . . . .	5
2.3 Objektrelationale Unverträglichkeit . . . . .	6
2.4 Geodäsie und Kartographie . . . . .	7
2.5 Network Common Data Format . . . . .	9
<b>3 Anforderungsanalyse</b>	<b>11</b>
3.1 Systembeschreibung . . . . .	11
3.2 Ermittlung der Anforderungen . . . . .	11
3.3 Funktionale Anforderungen . . . . .	12
3.3.1 Anwendende Perspektive . . . . .	12
3.3.2 Administrative Perspektive . . . . .	13
3.4 Nicht-Funktionale Anforderungen . . . . .	14
3.4.1 Anwendende Perspektive . . . . .	14
3.4.2 Administrative Perspektive . . . . .	14
3.5 Benötigte Daten . . . . .	14
3.6 Technische Anforderungen . . . . .	15
3.6.1 Verwendete Programmiersprachen . . . . .	15
3.6.2 Betriebssystem des Servers . . . . .	16
3.6.3 Web-Framework . . . . .	16
3.6.4 Datenbank . . . . .	17
<b>4 Systementwurf</b>	<b>18</b>
4.1 Modellierung der Datenbank . . . . .	18
4.2 Architektur . . . . .	18
4.2.1 Entwurf der Datenaggregation . . . . .	18

4.2.2	Entwurf der Webapplikation . . . . .	20
4.2.3	Ausarbeitung der Webrouten . . . . .	20
4.2.4	Aussehen der Webapplikation . . . . .	22
<b>5</b>	<b>Implementierung</b>	<b>23</b>
5.1	Datenaggregation . . . . .	23
5.1.1	Auslesen der Daten . . . . .	23
5.1.2	Schreiben der Daten . . . . .	24
5.2	Webapplikation . . . . .	27
5.2.1	Objektrelationales Mapping . . . . .	27
5.2.2	Controller . . . . .	34
5.2.3	Templates . . . . .	35
5.2.4	Kartendarstellung . . . . .	36
5.2.5	Zeichnen der Graphen . . . . .	40
<b>6</b>	<b>Testen</b>	<b>42</b>
6.1	Funktionstest . . . . .	42
6.1.1	Webapplikation . . . . .	42
6.1.2	Aggregation der Daten . . . . .	45
6.2	Usability-Umfrage . . . . .	47
6.2.1	Testaufbau . . . . .	47
6.2.2	Testauswertung . . . . .	47
6.2.3	Interpretation der Ergebnisse . . . . .	49
<b>7</b>	<b>Demonstration und Auswertung</b>	<b>51</b>
7.1	Geschaffene Lösung . . . . .	51
7.2	Erweiterungsmöglichkeiten . . . . .	52
	<b>Quellenverzeichnis</b>	<b>vi</b>
	<b>Glossar</b>	<b>viii</b>
	<b>Abkürzungsverzeichnis</b>	<b>ix</b>
	<b>Anhang</b>	<b>x</b>

## Abbildungsverzeichnis

1	Argos Messzyklus . . . . .	4
2	Qualitätszyklus der Argo-Daten . . . . .	5
3	Schematische Darstellung des Koordinatensystems WGS84 . . . . .	8
4	Use-Case Diagramm der Anforderungen . . . . .	12
5	Anforderungen und ihre Abhängigkeiten als gerichteter Graph . . . . .	13
6	Beschreibung der Entitäten der Datenaggregation . . . . .	18
7	Entwurf der Architektur der Aggregation der Daten . . . . .	19
8	Grafischer Grobentwurf der Webapplikation . . . . .	22
9	Architekturbeschreibung von ArgoData . . . . .	23
10	Sequenzdiagramm der Einbindung von SQLAlchemy . . . . .	32
11	<b>Funktionstest I</b> Der Aufruf der Webapplikation . . . . .	42
12	<b>Funktionstest II</b> Die Anzeige der Hilfe . . . . .	42
13	<b>Funktionstest III</b> Mousehover über Messstation . . . . .	43
14	<b>Funktionstest IV</b> Werte werden angezeigt . . . . .	43
15	<b>Funktionstest V</b> Metainformationen werden angezeigt . . . . .	44
16	<b>Funktionstest VI</b> Der Pfad der Messstation kann zurückverfolgt werden . . . . .	44
17	<b>Funktionstest VII</b> Fehlerhafte Daten werden markiert . . . . .	44
18	Ergebnis der Umfrage zur Usability . . . . .	50
19	Die Webpräsenz von ArgoData . . . . .	51

## Quellcodeverzeichnis

1	Die Verzeichnisstruktur der vom aoml bereitgestellten Daten . . . . .	6
2	Extraktion eines netCDF-Datensatzes zur Berechnung des Erstellungsdatums . . . . .	9
3	Webrouen der Datenrepresentation . . . . .	20
4	Webrouen der Darstellung . . . . .	21
5	Kontextmanager zur Sicherstellung der richtigen Dateibehandlung. . . . .	24
6	Die Verwendung des Kontextmanagers . . . . .	24
7	Implementierung des Iterators zur Steuerung der Aggregationssequenz . . . . .	25
8	Verwendung der Schnittstelle zur Steuerung der Datenaggregation . . . . .	25
9	Factory zur Extraktion der Datensätze . . . . .	26
10	Modellklasse für ein Argo-Float . . . . .	27
11	Modellklasse für eine Messung . . . . .	28

12	Das Schreiben der Daten eines Argo-Floats in die Datenbank . . . . .	29
13	Anfragen über SQLAlchemy zum Lesen von Daten . . . . .	30
14	Das Ausführen von SQL Anfragen über SQLAlchemy . . . . .	31
15	webapp.models.__init__.py . . . . .	32
16	webapp.__init__.py . . . . .	34
17	webapp.templates.map.html . . . . .	35
18	Das ol.Map Element aus der Kartendarstellung . . . . .	36
19	Gekürzte geoJSON zur Darstellung der Argo-Floats . . . . .	37
20	Die Funktion argoFloatsLayer . . . . .	38
21	Das Abfangen eines pointermove-Events . . . . .	38
22	Das Abfangen eines Klick-Events . . . . .	39
23	Ausliefern eines Funktionsplots Flask . . . . .	40
24	Erneuerung des Datensatzes . . . . .	45

## Tabellenverzeichnis

1	Beschreibung der ausgewählten Daten . . . . .	15
---	---	----

# 1 Einleitung

Die im Jahre 1965 von Gordon Moore vorhergesagte Gesetzmäßigkeit, die Komplexität, und damit die Speicherdichte integrierter Schaltkreise, würde sich regelmäßig verdoppeln (vgl. [Moo98]), hat sich seitdem bis zum heutigen Tag bewahrheitet. Damit sieht sich die Menschheit über 50 Jahre später in einer einmaligen Lage. Zum einen verfügen wir über eine noch nie dagewesene Ansammlung an Informationen, zum anderen wurden die Speichermedien durch die wachsende Komplexität immer flüchtiger und schwieriger in der technischen Handhabung. Werden kommende Zivilisationen in der Lage sein, diesen Pool an Informationen für sich zu nutzen oder sollten wir vielmehr annehmen, dass wir eine einmalige Chance haben, die wir nicht vergeuden sollten?

In diesem Kontext scheint es folgerichtig, dass Bewegungen wie Freie Software, Creative Commons und Open-Access die Hürden für den Zugang zu den Informationen weiter verringern. Viele der Informationen stehen heute frei zur Verfügung und können genutzt werden und doch werden, von einem gefühlt immer größeren Anteil unseres Kulturkreises, Prinzipien zur Bewertung von Quellen abgelehnt. In einer Zeit, in der es einfacher denn je ist, Fakten zu überprüfen, werden wissenschaftlich bewiesenen Aussagen wie dem Klimawandel nicht geglaubt. Es scheint, als wären viele Menschen der Flut an Informationen überdrüssig, als wende sich ein großer Teil überfordert davon ab. Die Frage ist, was kann dazu beitragen dieses Potential mehr zu nutzen? Über welche Mittel verfügt die Informatik, Daten in einen Kontext einzubetten? Wie muss der Kontext gestaltet sein, über den Menschen an wissenschaftliche Arbeit herangeführt werden können? Welche Daten sind geeignet, um die Brisanz und das Potential unserer Zeit einem breiteren Publikum zuzuführen? Mit diesen Fragestellungen motiviert sich die hier vorliegende Arbeit.

Im Rahmen dieser Abschlussarbeit wurde, über die Aggregation wissenschaftlicher Daten, ein exploratives Werkzeug geschaffen, um diese intuitiv erfahrbar zu gestalten. Dies geschieht mit dem Ziel, dass hier eine Identifikation mit den Messwerten und des wissenschaftlichen Prozesses zu erreicht werden kann. Als Datengrundlage wurden Daten des Argo-Programms verwendet. Unter dem Dach dieses Forschungsprojektes werden seit dem Beginn dieses Jahrtausends das Wasser der Weltmeere nach den Parametern Temperatur, Salzgehalt und Leitfähigkeit untersucht. Diese Daten stehen unter einer freien Lizenz zur Verfügung und können in eigene Projekte eingebunden werden. Die Messdaten dienen Wissenschaftlern, die Auswirkungen des globalen Klimawandels zu untersuchen. Die Herausforderung besteht darin, die hochkomplexen Messdaten in einer Form aufzubereiten, so dass diese einfach verstanden werden können.

Ein solches Werkzeug könnte zum Beispiel in Schulklassen eingesetzt werden. Die



Lernenden könnten, innerhalb einer Kontextvorgabe durch den Lehrenden, die Auswirkungen des Klimawandels auf die Weltmeere untersuchen. Hier könnte auch das Interesse und ein eventueller Berufswunsch im Wissenschaftssektor geweckt werden. Die Applikation richtet sich aber nicht explizit an Heranwachsende. Erwachsene Personen könnten sich hier, eigenes Interesse vorausgesetzt, weiterbilden. Die Applikation kann hier Relevanz der Forschung und die in diesem Programm erhobenen Messwerte erfahrbar machen.

Diese Arbeit handelt von der Konzeption und Entwicklung eines Prototypen mit der oben genannten Problemstellung. Zu Beginn wird das Argo-Programm näher vorgestellt. Es wird auf den Prozess der Datenerhebung und Aufbereitung eingegangen. Im Anschluss daran werden die Anforderungen der Software ausgearbeitet, um im Folgenden darauf gangbare Lösungsansätze und die geeignetsten Werkzeuge ausgearbeitet. Anschließend wird das Softwaresystem entworfen. Es werden die geplante Architektur und die wichtigsten Geschäftsprozesse ausgearbeitet und beschrieben. Über das Aufzeigen von gangbaren Alternativen werden die passendsten Werkzeuge und Entwicklungsmuster für diese Problemstellung ermittelt. Nach der Planung folgt eine Beschreibung der vorgenommenen Implementierung. Auch hier werden Implementierungen iterativ verfeinert, so dass die passendste Lösung gefunden werden kann. Um die Qualität der Software beschreiben zu können, folgt im darauf folgenden Kapitel eine Beschreibung der verwendeten Testverfahren. Hier werden Metriken ausgearbeitet, welche erlauben, funktionale als auch nicht-funktionale Anforderungen überprüfbar messen zu können. Im Abschluss findet eine Präsentation des entwickelten Prototypen statt. Dabei wird ein möglichst kritischer Blick auf Problemstellung und Lösungen geworfen. Abschließend wird ein Ausblick auf Alternativen und mögliche Weiterentwicklungen der Prototypen gegeben.

## **1.1 Existierende Lösungen**

Das Joint Technical Commission for Oceanography and Marine Meteorology (JCOMM) bietet mit [jcommops.org](http://jcommops.org) eine Grundlage für die wissenschaftliche Arbeit unter anderem mit den von Argo gesammelten Daten. Neben der Darstellung von Karten erfährt der Nutzer hier, von Sensordaten über den Bautyp der jeweiligen Boje alles, was die Bojen zu erzählen haben. Daneben werden über diese Plattform auch redaktionelle Reports veröffentlicht, um der Leserschaft ein Bild der aktuellen Lage unserer Weltmeere zu vermitteln.

Zwar bietet die Plattform durch ihre kartenbasierte explorative Darstellung ein ähnliches Angebot, wie es in der hier zu erstellenden Applikation geben soll. Die Fülle der Parameter erfordert vom Benutzer aber die Bereitschaft, sich vertieft in die Materie einzuarbeiten. Damit richtet sich das Angebot an Wissenschaftler und Journalisten. Diesen dient sie als eine hervorragende Datengrundlage für deren Veröffentlichungen. Ein Benutzer aus der für diese Arbeit definierten Zielgruppe wird von einem derartigen Angebot wohl eher abgeschreckt sein, bevor er dessen Vorteile für sich erarbeitet.

Das Global Data Access Committee (GDAC) der Coriolis (siehe [Arg17b]) ermöglicht den Download von, nach verschiedenen Filterkriterien ausgewählten, Datensätzen. Über einen Betrachter können Werte einer spezifischen Treibboje angesehen werden. Die primäre Aufgabenstellung dieser Plattform ist das Extrahieren der Daten, um diese in einer wissenschaftlichen Publikation verwenden zu können. Jenseits einer akademischen Verwendung müsste ein Benutzer hier viel Neugierde und Zeit für die Einarbeitung in das Thema mitbringen, um aus den angebotenen Daten einen Mehrwert für sich zu generieren.

## **1.2 Alleinstellungsmerkmal & Abgrenzung**

Das Ziel dieser Arbeit ist die Darstellung von wissenschaftlichen Daten aus dem Argo-Programm. Im Gegensatz zu den bereits vorhandenen Lösungen sollen die Daten aber nicht zur wissenschaftlichen Verwendung aufbereitet werden. Vielmehr sollen die hochkomplexen Daten auf ein einfach erfahrbares Maß herunter gebrochen werden. Zum Zeitpunkt dieser Arbeit existiert noch kein exploratives Tool für Daten des Argo-Programms für die hier definierte Zielgruppe.

## 2 Grundlagen

### 2.1 Argo-Programm

Zum Ende des vergangenen Jahrtausends verdichteten sich die Hinweise auf einen globalen und durch Menschen verursachten Klimawandel. Um dessen Auswirkungen auf die Weltmeere studieren zu können, wurde unter dem Dach des Global Ocean Observing System (GOOS) das Argo-Programm gegründet. Dieses untersucht, unterstützt durch das Satellitensystem Iason, die Wassersäule der oberen 2000m auf deren chemischen Eigenschaften. Dabei werden in ständigen Intervallen Salzgehalt, Druck, Temperatur und Leitfähigkeit gemessen. Die ermittelten Daten werden veröffentlicht, so dass diese durch Wissenschaftler ausgewertet werden können.

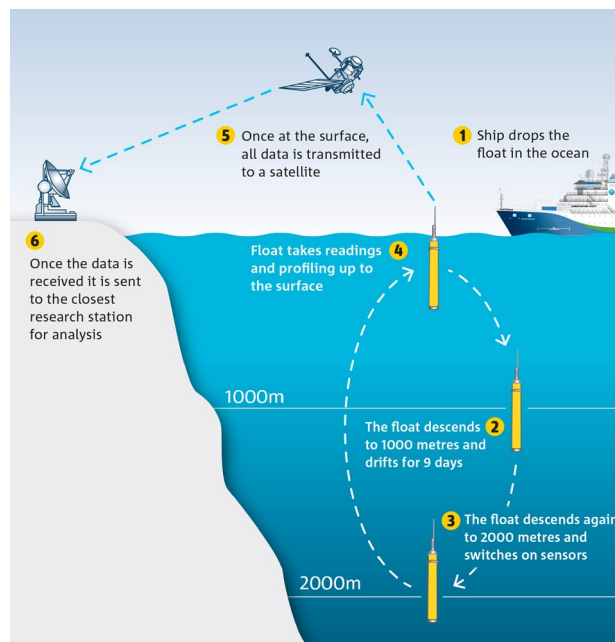


Abbildung 1: Argos Messzyklus

Bildquelle:

<https://www.csiro.au/en/Research/OandA/Areas/Marine-technologies/Argo-robotic-floats>

Die Argo Treibbojen werden mit Schiffen an spezifischen Punkten ausgesetzt. Ein Messzyklus beträgt 10 Tage. Die Boje taucht am Anfang des Zyklus auf 1000 Meter Tiefe herab. In dieser Tiefe verbringt die Sonde die nächsten 9 Tage. Anschließend sinkt sie auf die maximale Tiefe von 2000 Metern herab, um daraufhin wieder zur Oberfläche aufzusteigen. An der Oberfläche sendet die Boje innerhalb von 6 bis 12 Stunden die Daten über den Satellitenarray Iason an die Bodenstationen. Der oben genannte Messzyklus kann in Abbildung 1 nachvollzogen werden.

Im Anschluss an die Erhebung werden die Daten auf deren Plausibilität und Qualität überprüft (vgl. [SG] S. 3). Nach diesem Prozess werden die aufbereiteten Daten über

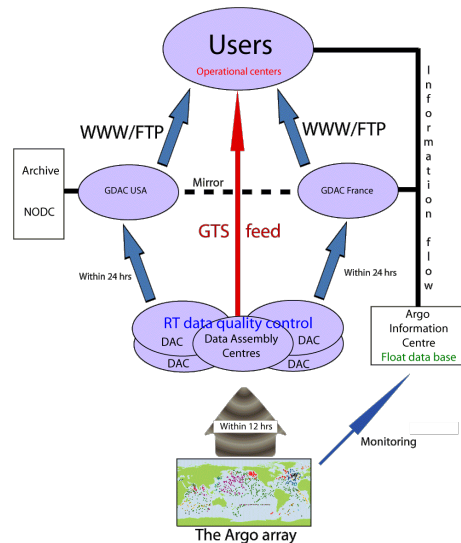


Abbildung 2: Qualitätszyklus der Argo-Daten  
 Bildquelle: [http://www.argo.ucsd.edu/Argo\\_data\\_and.html](http://www.argo.ucsd.edu/Argo_data_and.html)

die GDAC in monatlichen Releasezyklen veröffentlicht. (vgl. [Arg]). Der Werdegang der Daten ist in Abbildung 2 zu sehen.

## 2.2 Datengrundlage

Alle am Argo-Programm teilnehmenden Organisationen verpflichten sich auf eine gemeinsame Datenpolitik. So gibt es keine Herrschaft über die erhobenen Daten. Vielmehr stehen diese ab dem Zeitpunkt der Veröffentlichung transparent der Öffentlichkeit zur Verfügung. Nach der Übermittlung werden die Messdaten einer Qualitätskontrolle unterzogen. Sie werden auf Plausibilität und Abweichungen überprüft. Nach dieser Qualitätskontrolle werden die Daten nun über die GDAC in Frankreich und den USA veröffentlicht. Diese können über HTTP und FTP abgerufen werden.

Einmal im Monat werden die Daten als Snapshots, unter digital object identifier (DOI) zusammengefasst. Während und vor der Qualitätskontrolle liegen die Daten in den Formaten TESAC und BUFR vor. Die von den GDAC veröffentlichten Daten liegen im Format Network Common Data Format (netCDF) vor. Diese sind unter der Lizenz Attribution 4.0 International (CC BY 4.0) veröffentlicht und dürfen unter der Nennung der Lizenz frei verwendet und dabei auch verändert werden (vgl. [Arg17a]).

In Listing 1 ist die Ordnerstruktur der netCDF Dateien zu erkennen. Über den Ordnernamen aoml ist die Herkunft des DOI zu erkennen. In diesem Fall wurden die Dateien von einem Server der „Atlantic Oceanographic & Meteorological Laboratory“ erstellt. Hier befindet sich für jede Messboje ein Unterordner. Die Dateien meta, prof, Rtraj und tech sind eine Quelle für Metainformationen. Die Messprofile einer Boje finden sich im

Ordner profiles. Hier wird für jeden Messzyklus eine Datei angelegt. Dieser startet bei 1 und inkrementiert über jeden Messzyklus um 1.

```

1 ./aoml/1900200/
2 - 1900200_meta.nc
3 - 1900200_prof.nc
4 - 1900200_Rtraj.nc
5 - 1900200_tech.nc
6 - profiles
7   - D1900200_001.nc
8   ...
9   - D1900200_215.nc
10  - D1900200_216.nc
11 ./aoml/1900201/
12 ...

```

*Listing 1: Die Verzeichnisstruktur der vom aoml bereitgestellten Daten*

## 2.3 Objektrelationale Unverträglichkeit

In einem Softwareparadigma manifestiert sich ein bestimmtes Konzept in der Modellierung der Welt. Durch diese konzeptionelle Betrachtungsweise beeinflusst ein Paradigma den Erstellungsprozess sowie die Ergebnisse des Softwaredesigns und zwingt diesem seine Grenzen auf. Probleme bei der Kombination verschiedener Paradigmen werden dabei als Unverträglichkeit (impedance Mismatch) bezeichnet. Objektorientierte Programmierung und eine relationale Abfrage von Datensätzen sind weit verbreitete Paradigmen in der Softwareentwicklung. Damit ist die objektrelationale Unverträglichkeit eine häufig auftretende Herausforderung (vgl. [IBNW09] S. 36-38). Hier genannt sind unter anderem folgende Betrachtungsweisen als Ursachen für die objektrelationale Unverträglichkeit:

**Strukturelle Unterschiede** Die objektorientierte Programmierung erlaubt das Definieren beliebig komplexer Strukturen aus Methoden und Klassen. Durch Vererbungsstrukturen ist es möglich, Objekte zu spezialisieren und Grundkonzepte von Klassen zu generalisieren. Eine relationale Algebra wird durch Tupel, Mengen und Wahrheitswerte definiert. Inhärent wiederholbare Strukturen oder Hierarchien können mit diesen Mitteln nicht umgesetzt werden.

**Datenkapselung** In der objektorientierten Programmierung können die intrinsischen Attribute eines Objektes verborgen werden. Dieses Konzept ist als Kapselung bekannt und erlaubt es, den Zugriff auf die gekapselten Strukturen einzuschränken. In einer relationalen Algebra ist eine derartige Abstraktion über die Daten nicht vorgesehen.

**Objektidentität** Durch die Instanziierung eines Objektes aus einer Klasse erhält dieses eine eindeutige Identität. Somit unterscheiden sich zwei Objekte, auch wenn diese Träger eines identischen Datensatzes sind, durch ihre Repräsentation im Arbeitsspeicher. Relationen werden durch den Primärschlüssel, und damit über ihre Daten, definiert. Zwei eigenständige Relationen mit identischem Datensatz sind damit nicht möglich.

Als Hilfsmittel zur Überwindung der oben genannten Probleme werden objektrelationale Mapper (ORM bzw. O/R-Mapper) eingesetzt. Durch dieses Mapping wird eine Schnittstelle oder Abstraktionsebene zwischen Programmteilen aus den jeweiligen Sprachparadigmen definiert, um den impedance Mismatch möglichst transparent zu überwinden.

## 2.4 Geodäsie und Kartographie

In der vorliegenden Applikation werden geographische Daten über einen Kartendienst angezeigt. Aus diesem Grund erfolgt hier eine Zusammenfassung der wichtigsten geodätischen und kartographischen Grundlagen.

Die geographische Länge eines Punktes  $P_l$  bezeichnet den Winkel zwischen einer vom Nullmeridian durch den Erdmittelpunkt geführten Fläche und einer Meridianebene, die durch den Punkt  $P_l$  führt. Analog dazu ist die geographische Breite eines Punktes  $P_b$  der Winkel zwischen der Äquatorschnittfläche und der Flächennormalen an Punkt  $P_b$  (vgl. [WS11] S. 18).

Das „World Geodetic System 1984 (WGS84)“ ist ein geozentrisches, vom Erdschwerpunkt abgeleitetes Koordinatensystem. Dieses wird unter anderem von GPS zur Zuordnung von Positionen auf der Erde verwendet. Eine schematische Darstellung ist in Abbildung 3 zu sehen. Die Z-Achse führt durch den Erdschwerpunkt entlang der Rotationsachse durch den Nordpol, die X-Achse vom Zentrum der Erde durch einen Nullmeridian (Greenwich) und die Y-Achse wird als Orthogonale zur X-Achse ebenso durch den Erdschwerpunkt geführt (vgl. [WS11] S. 13-14).

Die Gauß-Krüger Projektion erlaubt die Darstellung des Globus auf einer Ebene. Um die Krümmung des Globus auszugleichen wird die Erde in  $3^\circ$  breite Meridianstreifen eingeteilt. Als Mittelpunkte dienen dabei die Meridiane  $[3^\circ, 6^\circ, 9^\circ, \dots]$ . Von den Mittelmeridianen aus wird eine Projektion an anliegende Zylinder vorgenommen. Auf diese Weise entstehen zu den Rändern der Meridianstreifen Längen- und Flächenverzerrungen, die Darstellung ist aber winkeltreu.

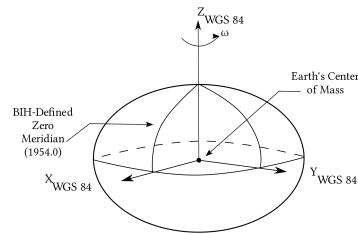


Abbildung 3: Schematische Darstellung des Koordinatensystems WGS84

Bildquelle:

[https://commons.wikimedia.org/wiki/File:WGS\\_84\\_reference\\_frame\\_\(vector\\_graphic\).svg](https://commons.wikimedia.org/wiki/File:WGS_84_reference_frame_(vector_graphic).svg)

Im UTM-System wird ein analoges Verfahren angewendet. Die Meridianstreifen haben hier aber eine Breite von  $6^\circ$ . Anstatt des Mittelmeridianes werden hier zwei Schnittkurven mit einem Abstand von 180km als längentreues Element für die Abbildung der Y-Achse verwendet. Der Bereich zwischen diesen Schnittkurven wird gestaucht (vgl. [WS11] S. 23-25).

## 2.5 Network Common Data Format

Das netCDF dient zum Austausch von wissenschaftlichen Daten. Es ist eine Weiterentwicklung des von der NASA entwickelten Common Data Format (CDF). Das Datenformat zeichnet sich dadurch aus, dass es selbstbeschreibend ist. Die Dokumentation ist Teil des Datensatzes und wird so immer mitgeführt. Dies soll die Portabilität des Datensatzes verbessern (vgl. [RGS<sup>+</sup>]).

Es wurde auch eine Bibliothek für Python entwickelt (vgl. [net]). Mit dieser ist es möglich, die Binärdaten zu öffnen und zu numpy-Arrays zu extrahieren. In Listing 2 ist die Verwendung exemplarisch an der Extraktion und der Umrechnung des Erstelldatums aufgezeigt und im Folgenden beschrieben.

```

1 from netCDF4 import Dataset
2 import datetime
3
4 dataset = Dataset('./4900442/profiles/D4900442_042.nc')
5 print(dataset.variables['JULD'])
6
7 # <class 'netCDF4._netCDF4.Variable'>
8 # float64 JULD(N_PROF)
9 #     long_name: Julian day (UTC) of the station relative to
10 #     ↪ REFERENCE_DATE_TIME
11 #     units: days since 1950-01-01 00:00:00 UTC
12 #     conventions: Relative julian days with decimal part (as parts of
13 #     ↪ day)
14 #     _FillValue: 999999.0
15 # unlimited dimensions:
16 # current shape = (1,)
17 # filling off
18
19 julian_date = dataset.variables['JULD'][:][0]
20 dataset.close()
21
22 print(julian_date)
23 # 20062.5483218
24
25 juld_zero = datetime.datetime.strptime('1950-01-01 00:00:00 UTC',
26                                       '%Y-%m-%d %H:%M:%S UTC')
27 date_creation = juld_zero + datetime.timedelta(days=int(julian_date))
28 print(date_creation)
29 # datetime.datetime(2004, 12, 5, 0, 0)

```

*Listing 2: Extraktion eines netCDF-Datensatzes zur Berechnung des Erstellungsdatums*

Um einen Datensatz zu öffnen, bildet man eine Instanz der Klasse netCDF4.Dataset. Die Auswahl des Profils gelingt durch die Pfadangabe als Parameter bei der Instanziierung. Über das Attribut dataset.variables werden die Datensätze als OrderedDict gehalten. Bei der Extraktion eines Parameters erhält man zunächst die Dokumentation des jeweiligen Datensatzes. In diesem Fall handelt es sich um den Zeitpunkt der Satellitenübertragung des betreffenden Messprofils. Aus der Dokumentation lassen sich für



die Weiterverarbeitung wichtige Parameter entnehmen. So ist der Datensatz in Form des C-Datentyps `float64` codiert. Das Datumsformat entspricht dem Julian Day ab dem Zeitpunkt *01. Januar 1950*.

Um die Werte eines Datensatzes zu extrahieren, wird über die numpy-slicing Operation `arr[:, :]` der Datensatz in vektorieller Form extrahiert. Da in diesem Array nur ein skalarer `float64` Wert enthalten ist, kann dieser als nulltes Element vom Array entnommen werden. Zur Überführung in das Format des Gregorianischen Kalenders wird ein Datumsobjekt des Referenzdatums benötigt. Durch die Verwendung von `timedelta` werden die Tage aus dem Feld `JULD` addiert. Da ein Messzyklus 10 Tage andauert, kann an dieser Stelle der Datensatz, durch das Überführen zu einem Integer, vereinfacht werden. Die Casting-Operation `int()` rundet in jedem Fall ab.

## 3 Anforderungsanalyse

### 3.1 Systembeschreibung

Die zu entwickelnde Anwendung verfolgt einen Datengetriebenen Ansatz. Daten aus dem Argo-Programm werden erhoben und eine Auswahl davon in ein Datenformat überführt. Zur Darstellung wird eine Webapplikation verwendet. Über diese werden unter Zuhilfenahme von Positionsmarkern auf einer Weltkarte, die letzten Positionen der Messstationen angezeigt. Über einen Klick auf die Repräsentation einer Messboje werden weitere Daten der Messstation präsentiert.

### 3.2 Ermittlung der Anforderungen

Ausgehend von einem ersten Prototypen wurden Anwendungsfälle entwickelt. Eine schematische Darstellung ist in Abbildung 4 zu sehen. Dabei wurden zwei Klassen von Akteuren herausgearbeitet. Darunter finden sich die Benutzenden der Applikation. Diese erwarten einen reibungslosen Betrieb der Applikation und die Darstellung der Messstationen in der gegebenen Form. Zum anderen existiert die Klasse der Administrierenden. Diese benötigen Werkzeuge, um die Daten zu erneuern und in die Datenhaltung einzuspielen. Beide Klassen haben verschiedene Sichten auf die Datenhaltung. Daraus entsteht ein Spannungsverhältnis, welches minimiert werden muss. Unter anderem ist zu erwarten, dass durch die Erneuerung der Datenhaltung eine Ausfallzeit in der Darstellung der Messwerte verursacht wird. Hier sind geeignete Mittel zu wählen, die die Funktionalität so wenig wie möglich einschränken.

Anhand dieses Modelles wurden Anforderungen ausgearbeitet. Diese wurden in Form eines, in Abbildung 5 ersichtlichen, gerichteten Graphen aufbereitet. Dabei werden die Anforderungen und deren Nachbedingungen über Knoten des Graphen repräsentiert. Finden sich in einer Anforderung Vorbedingungen zu den Nachbedingungen einer anderen Anforderung, so wird diese Abhängigkeit über eine Kante dargestellt. Diese Darstellung half dabei, Definitionslücken in den Beziehungen der Anforderungskette besser verstehen zu können und formale Lücken schließen zu können.

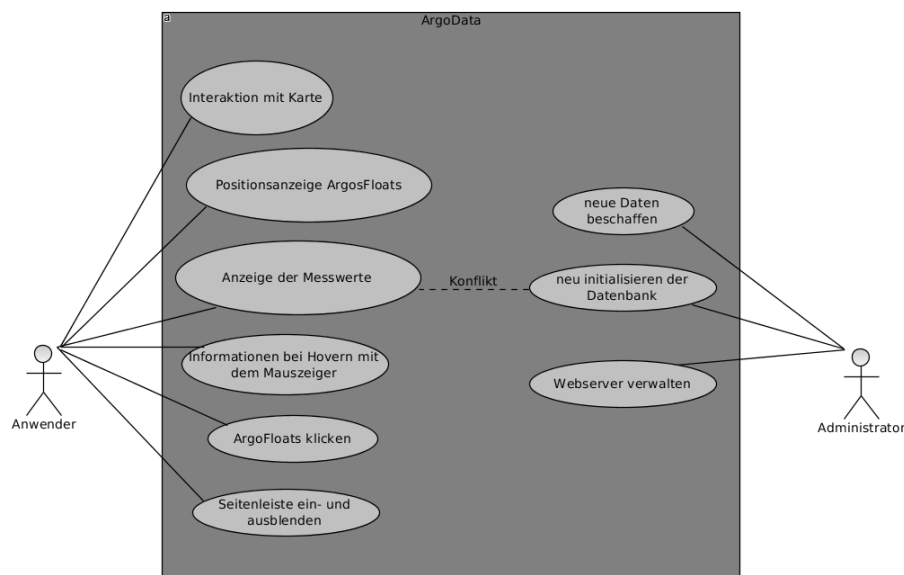


Abbildung 4: Use-Case Diagramm der Anforderungen

### 3.3 Funktionale Anforderungen

#### 3.3.1 Anwendende Perspektive

In der Anwendung wird die Darstellung einer Webapplikation erwartet. Angesteuert durch die zugehörige URL, sollen über eine Kartenapplikation die Grenzen der Ozeane und der umliegenden Kontinente erkennbar werden. Eine genauere Darstellung von Landmarken wie Straßen, Städten oder Gebirgen ist für den Aussagewert der Applikation nicht erforderlich. Die Kartendarstellung soll über Interaktionsmöglichkeiten, wie dem Einstellen der Zoomstufe, sowie des ausgewählten Kartenbereichs, verfügen. Über den Kartendienst ist die letzte Position jeder Messboje aus dem Datensatz ersichtlich. Um Streuung und Verdichtung an spezifischen Orten sehen zu können, werden alle Bojen des angezeigten Kartenausschnittes dargestellt. Es erfolgt keine Zusammenfassung oder Ausblendung. Wird die Maus über die visuelle Repräsentation einer Messboje geführt, so sollen grundlegende Daten der Messstationen dargestellt werden, gleichzeitig wird diese Station optisch hervorgehoben. Ein Mausklick auf die schematische Darstellung soll spezifische Messdaten anzeigen. Dazu werden in einem separierten Darstellungsfeld der Verlauf der Messdaten, sowie Werte, die eine Identifikation der Messboje ermöglichen, angezeigt. Zusätzlich soll der zurückgelegte Weg der Messstation ersichtlich sein. Eine Trendanalyse der Daten kann hier ein Maß für die Richtung der Messwerte ermöglichen. Können mehrere Messstationen gleichzeitig ausgewählt werden, so wäre es sinnvoll die akkumulierten Werte darzustellen.

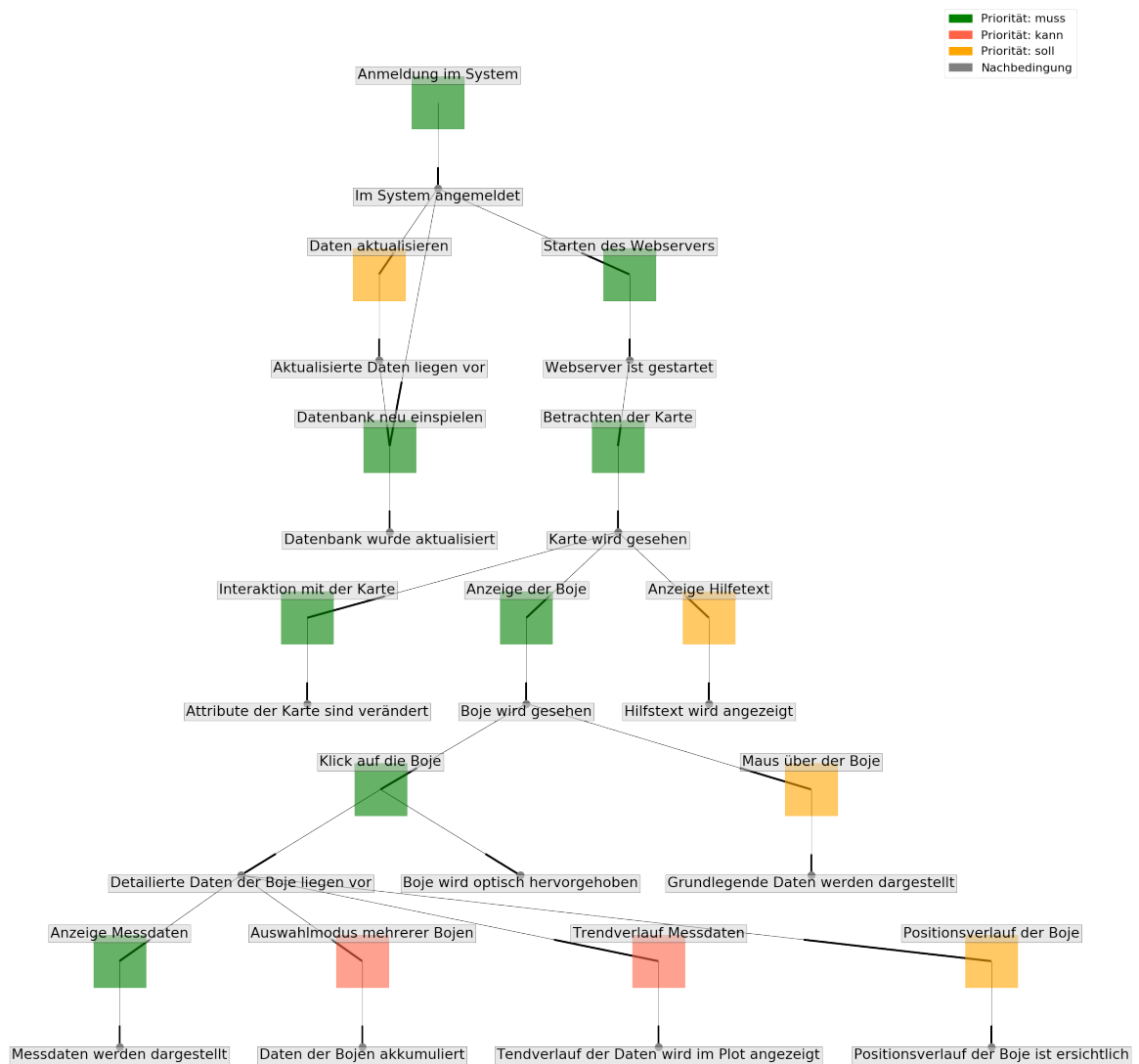


Abbildung 5: Anforderungen und ihre Abhängigkeiten als gerichteter Graph

### 3.3.2 Administrative Perspektive

Es ist notwendig, dass die Datensätze des Argo-Programms ausgelesen und in ein für die Datenbank aufbereitetes Format überführt werden. Neue Daten müssen hierfür aus den Quellen des Argo-Programms heruntergeladen werden. Um die Inhalte zu erstellen, wird statt einer journalistischen eine administrative Tätigkeit erwartet. Durch den Datengetriebenen Ansatz der Applikation müssen für das Erstellen der Inhalte keine Webmasken wie Texteingabefelder zur Verfügung gestellt werden. Vielmehr ist es sinnvoll, die hier benötigten Werkzeuge als Skripte zur Verfügung zu stellen, um die administrativen Aufgaben über die Kommandozeile ausführen zu können. Dies würde eine Abbildung der Prozesse bis hin zu einer Vollautomatisierung erlauben und eine

örtliche Trennung von Datenaggregation und -darstellung vereinfachen. Für die administrativen Tätigkeiten werden Werkzeuge benötigt, um neue Daten in die Datenbank überführen können.

## 3.4 Nicht-Funktionale Anforderungen

### 3.4.1 Anwendende Perspektive

Für die Benutzenden ist der Grad der Benutzbarkeit ein zentraler Aspekt der Applikation. Die Bedienungsmuster müssen klar ersichtlich und intuitiv erfahrbar sein. Die Darstellung der Applikation soll einfach und schlicht gehalten werden und dem gewohnten Erscheinen modernen Webapplikationen entsprechen. Es soll auf die wichtigsten Aspekte des Datenschutzes Wert gelegt werden. Insbesondere im Hinblick auf Datensparsamkeit sind das Einbinden von Trackern, Loginfunktionen und das Speichern von Logfiles zu bewerten. Sicherheitsaspekten muss aus der Anwendersicht nur eine normale Rolle zugewiesen werden. Dabei sollen die aktuell als Standard angesehenen Sicherheitsvorkehrungen getroffen werden.

### 3.4.2 Administrative Perspektive

Für die Aggregation der Daten ist die Sicherheit ein zentraler Aspekt. Dies umfasst die Bereiche Authentifizierung und Autorisierung der administrierenden Personen.

## 3.5 Benötigte Daten

Um die Darstellung zu vereinfachen und die Kommunikation mit der Datenbank zu beschleunigen, ist im ersten Schritt eine Auswahl aus den Daten der Argo-Bojen vorzunehmen. Hierbei soll darauf geachtet werden, nur diejenigen Daten zu verwenden, welche für die Erbringung des Dienstes notwendig sind. Aus diesem Grund findet sich in Tabelle 1 eine Auswahl aus dem Datenkatalog (vgl. [CKT<sup>+</sup>15] S. 19 ff.) des Argo-Programms.

Das Feld PLATFORM\_NUMBER wird benötigt um Messprofile eindeutig der Plattform zuordnen zu können, während die CYCLE\_NUMBER eine Messung innerhalb ihrer Plattform eindeutig macht. Das Datum der Messdatenübertragung kann aus dem Feld JULD ermittelt werden. Dieses erlaubt es, die Messreihe in ihrem zeitlichen Kontext zu sehen. Die Anzahl der Sensoren lässt sich aus dem Feld N\_PARAM herleiten. Dies wird benötigt, da einige Messstationen nicht alle Sensoren eingebaut haben. Über die Geokoordinaten wird die lokale Zuordnung der Datenübertragung erreicht.

Datenfeld	Beschreibung	Datentyp
PLATFORM_NUMBER	Eindeutige Identifikationsnummer einer Messstation	<string8>
CYCLE_NUMBER	Fortlaufende und innerhalb einer Messboje eindeutige Identifikationsnummer eines Messprofils	<int32>
JULD	In Julian Date codierter Zeitpunkt der Übertragung einer Messung	<float64>
N_PARAM	Anzahl der Messsensoren	<string16>
LATITUDE & LONGITUDE	Die Positionsdaten einer Messung zum Zeitpunkt der Übertragung der Werte	<float64>
PRES	Messvektor des Wasserdrucks	<float32>
TEMP	Messvektor der Wassertemperaturen	<float32>
PSAL	Messvektoren des Salzgehaltes	<float32>
FLOAT_OWNER	Besitzer der Messboje	<string64>
LAUNCH_DATE	Aussetzdatum	<date_time>
PROJECT_NAME	Federführendes Forschungsprojekt	<string8>

Tabelle 1: Beschreibung der ausgewählten Daten

Dies wird benötigt um in der Kartendarstellung der Messung einen Ort zuzuweisen. Aus den Messwerten werden die Felder PRES, TEMP und PSAL ausgewählt. Neue Modelle der Messbojen besitzen noch das Feld COND, über das sich die Leitfähigkeit des umliegenden Wassers herleiten lässt. Dieses ist aber noch nicht bei vielen Messstationen implementiert und wird hier deswegen nicht verwendet.

Die verwendeten Messwerte PRES, TEMP und PSAL eines Messprofils liegen in vektorieller Form vor. Für die Darstellung über einen univariaten Graphen wird nur ein skalarer Wert pro Messprofil benötigt. Die Daten sollen zusammengefasst werden, bevor diese in die Datenstruktur überführt werden, um die Größe der Daten zu verringern.

## 3.6 Technische Anforderungen

### 3.6.1 Verwendete Programmiersprachen

Die Darstellung der Webapplikation wird mit HTML und Javascript umgesetzt. Diese Sprachen gelten in der Entwicklung von Webseiten als Standard und werden von den gängigen Browsern unterstützt. Für die weiteren Teile der Applikation ist eine Programmiersprache zu finden, die alle benötigten Teilbereiche möglichst gut abbilden kann.

Das Öffnen der netCDF Dateien und die numerische Berechnung kann mit C, Java/Scala, R und Python erfolgen. Insbesondere die zuletzt genannten sind in der Datenverarbeitung und Numerik als etablierte Werkzeuge zu sehen.

Die Darstellung der Seite soll durch ein Web-Framework unterstützt werden. Hier gibt es in beinahe allen Hochsprachen entsprechende Werkzeuge. Wählt man aus der Problemstellung der numerischen Verarbeitung R und Python heraus, so ist die Auswahl der geläufigen Web-Frameworks bei Python höher anzusehen. Diese Sprache besitzt eine große Anzahl von Bibliotheken für die Anbindung von Datenbanken, sowie einige etablierte Bibliotheken für die Schaffung von Webapplikationen.

Python besitzt in Hinsicht auf Laufzeitkosten und einer nicht strikten Typisierung einige Nachteile gegenüber Sprachen wie C und Java. In Hinsicht die auf die breite Auswahl der Programmbibliotheken, sowie der numerischen Berechnung besitzt Python aber Vorteile und wird deswegen hier eingesetzt.

#### 3.6.2 Betriebssystem des Servers

Für die Laufzeitumgebung der Applikation wird ein GNU/Linux eingesetzt. Die Software wird so entwickelt, dass sie unter den gängigen Distributionen lauffähig sein wird. Für den Betrieb dieser Applikation wird sich hier für die Distribution Debian-Stable entschieden. Debian ist sehr verbreitet und stark auf Stabilität ausgelegt, sie zeichnet sich insbesondere gegenüber weiteren Distribution mit einer ähnlichen Verbreitung wie RHEL und Ubuntu durch ihre nicht kommerzielle Ausrichtung aus. Als Nachteil ist der fehlende kommerzielle Support zu sehen. Diese Arbeit deckt aber keine Bereiche ab, welche einen kostenpflichtigen Support rechtfertigen würden.

#### 3.6.3 Web-Framework

Für Python gibt es eine große Anzahl an Werkzeugen für die Entwicklung von Webseiten. Hier werden exemplarisch 3 herausgegriffen und für die hier vorliegende Aufgabe bewertet.

**Django** wird beworben als „The web framework for perfectionists with deadlines“.

Es wurde 2005 unter einer BSD Lizenz released und entwickelt, um die News-Seite des *Lawrence Journal-World* umzusetzen und zu verwalten. Django ist ein etabliertes und häufig verwendetes Web-Framework. Es ist dynamisch einsetzbar und für eine große Anzahl von Anwendungen verwendbar. Die Bibliothek ist außerdem durch Module erweiterbar. Die Software folgt dem „*batteries included*“ Ansatz und liefert in der Grundausstattung bereits alles Nötige mit, um eine Webseite inklusive Login und Eingabemasken für journalistische Tätigkeiten auszubauen.

**Flask** ist ein sogenanntes Micro-Framework. Es verwendet die Toolsammlung *Werkzeug* um Webseiten über WSGI darstellbar zu machen. Das Framework folgt

dem Keep It Simple, Stupid (KISS) Ansatz und liefert im Grundumfang nur diejenigen Werkzeuge, die man für die Verwaltung von einfachen Webseiten benötigt. Die Software lässt sich über Module erweitern. Einzelne Teilprojekte von Applikationen lassen sich in sogenannte Blueprints modularisieren.

**Falcon** ist ebenso als Micro-Framework zu sehen. Es wurde insbesondere in Hinblick auf Geschwindigkeit entwickelt. Das Framework erlaubt es Requests asynchron zu verarbeiten und lässt die auf Geschwindigkeit spezialisierte Python-Laufzeitumgebung pypy zu. Falcon ist ein relativ neues Framework und erfährt in letzter Zeit zur Schaffung von REST-APIs immer mehr Aufmerksamkeit. Es ist aber auch möglich, mit diesem Framework Webapplikationen mit einer Anzeige über HTML-Elementen zu gestalten.

In dieser Applikation wird die Schaffung von journalistischen und redaktionellen Inhalten eine sehr geringe Rolle spielen. Unter diesem Gesichtspunkt ist die Frage der Komplexität der verwendeten Software zu klären. Unter diesem Aspekt erscheint Django nicht als die ideale Wahl. Flask und Falcon verfolgen beide den Ansatz eines Micro-Frameworks. Die Laufzeitgeschwindigkeit des Controllers erscheint an dieser Stelle nicht als limitierender Faktor der Applikation. Zwar wäre es wünschenswert, die Datenbeschaffung der Webapplikation bereits im Backend asynchron zu erledigen, doch überwiegt das breitere Spektrum an Modulen und Dokumentationen für die Webentwicklung von Flask. Damit erscheint Flask in diesem Kontext als das geeignetste Werkzeug für diese Aufgabe und wird deswegen für die hier ausgearbeitete Lösung verwendet.

#### 3.6.4 Datenbank

Das Datenbank-Management-System (DBMS) soll über Schnittstellen in den Sourcecode des Programmes eingebunden werden. Die Datenbank soll einem relationalen Schema folgen. Die Verwendung sollte kostenfrei möglich sein und die benötigte Software über die Quellen des Betriebssystems verfügbar sein.

In dieser Applikation wurde sich für die Verwendung von PostgreSQL entschieden. Für die Verwendung in Flask wird SQLAlchemy als O/R-Mapper empfohlen (vgl. [Ron11] S.33).



## 4 Systementwurf

### 4.1 Modellierung der Datenbank

Um die Daten zu modellieren ist es sinnvoll, sich diese im Kontext der Erhebung zu betrachten. In (1) ist dieser Prozess vereinfacht dargestellt.

$$\text{Boje} \rightarrow (\text{misst}_{\text{an Ort}}) \rightarrow \text{Messprofile} \quad (1)$$

Dabei ist erkennbar, dass dieses Modell eine Verkettung von Entitäten und Ereignissen darstellt. Eine Boje misst über ihre Lebensdauer eine Anzahl von Messzyklen. Jeder dieser Messzyklen besteht aus einer gewissen Anzahl von Messwerten.

Um das Modell weiter fortzuführen, wurde diese Ereigniskette in ein Entitätsschema überführt. In Abbildung 6 ist die hier verwendete Modellierung des Prozesses zu sehen.

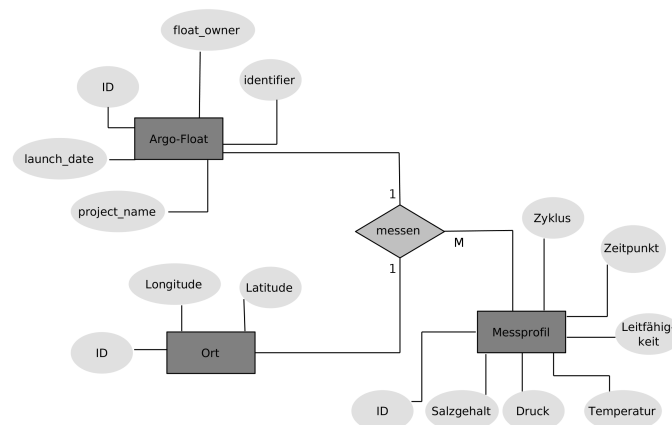


Abbildung 6: Beschreibung der Entitäten der Datenaggregation

### 4.2 Architektur

Die Applikation besteht aus zwei Grundkomponenten. Der erste Teilbereich ist für die Beschaffung und Aufbereitung der Daten zuständig, während der zweite Teilbereich für die Darstellung der Daten zuständig ist.

#### 4.2.1 Entwurf der Datenaggregation

Die Datenaggregation erfüllt zwei Funktionen. Zum einen muss sichergestellt sein, dass die Daten aus den vom Argo-Programm bereitgestellten Strukturen gelesen und in ein logisches Format überführt werden. Des Weiteren müssen diese Daten in die Datenbank der Applikation überführt werden.

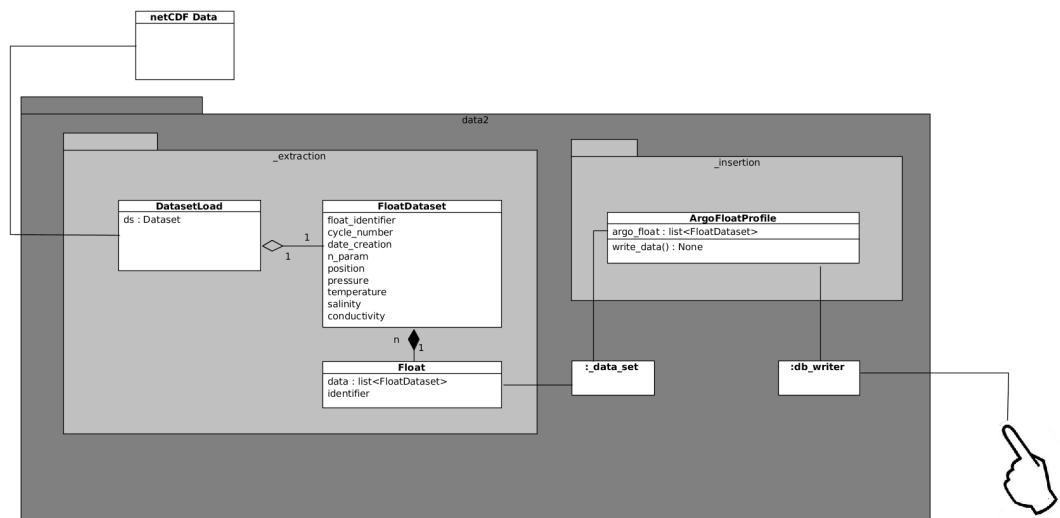


Abbildung 7: Entwurf der Architektur der Aggregation der Daten

Auch in diesem Modell ist die Ereigniskette aus (1) abzubilden. Jede Messstation wird über ein Objekt abgebildet. Die hier gespeicherten Daten sind für eine Messstation eindeutig. Jeder Messzyklus der Boje wird über ein weiteres, zur Messstation gehöriges Objekt abgebildet. Dieses ist innerhalb des Kontextes eindeutig und trägt die Informationen zur jeweiligen Messung.

Für die Persistierung der objektorientierten Strukturen wird ein Objektrelationaler Mapper (ORM) verwendet. Die Datenstruktur teilt sich das Aggregationsmodul mit der Webapplikation. Die Behandlung der Daten sowie eine Steuerung des Prozesses sind als weitere Schnittstellen zu definieren.

Wie in Abbildung 7 ersichtlich ist, wird die Aggregation der Daten über zwei Teilbereiche abgebildet. Zum einen müssen die benötigten Parameter aus den Dateien ausgelesen und modelliert werden. Als zweite Ebene der Aggregation ist der Prozess des Schreibens in eine Datenbank zu sehen. Diese verwendet die zuvor ermittelten modellierten Messprofile und schreibt sie anhand der dort enthaltenen Daten in eine Datenbank.

Es ist an dieser Stelle zu erkennen, dass zwei Stellen existieren, die die intrinsischen Eigenschaften des Modules an dieser Stelle beeinflussen. Im Folgenden werden die Schnittstellen aufgelistet, die eine richtige Handhabung festsetzen:

## Schnittstellen

### 1. Daten

- a) Die Datenstruktur ist normiert. Es ist sicherzustellen, dass die bekannte Daten- bzw. Ordnerstruktur abgearbeitet wird. Mit Änderungen in dieser Struktur muss nicht gerechnet werden.
- b) Es muss sicher gestellt werden, dass geöffnete Dateien wieder geschlossen werden.

### 2. Steuerung

- a) Daten separiert in die Datenbank zu schreiben, könnte zu Inkonsistenzen führen und soll vermieden werden.
- b) Der Prozess sollte als Sequenz modelliert werden.
- c) Es muss sicher gestellt werden, dass die Daten schrittweise in die Datenbank überführt werden. Würden zu Beginn alle Dateien geöffnet und gemeinsam im flüchtigen Speicher vorgehalten, könnte es zu Problemen führen.

## 4.2.2 Entwurf der Webapplikation

Die Webapplikation besteht aus zwei Teilkomponenten. Die erste Komponente ist für die Darstellung der Webseite zuständig (APP). Durch diese werden Elemente in HTML, Javascript und als Bilder ausgeliefert, welche im Webbrowser den Benutzenden angezeigt werden können. Die Darstellung erfolgt dabei dem Singlepage-Prinzip.

Die zweite Teilkomponente ist dafür zuständig, benötigten Daten bereitzustellen (API). Alle Daten aus der Datenhaltung werden über diese Schnittstelle angefordert. Die Daten werden über JSON codiert.

## 4.2.3 Ausarbeitung der Webrouten

Die API ist dafür zuständig, die benötigten Daten der Applikation bereitzustellen. Über definierte Webrouten wird über einen GET-Request ein JSON angefordert. Die hierfür ausgearbeitete Struktur ist in Listing 3 zu sehen und ist im Folgenden beschrieben:

1	(1.1)	GET	/last_seen
2	(1.2)	GET	/last_seen/[force_reload]
3	(1.3)	GET	/argo_float/[identifizier]
4	(1.4)	GET	/positions/[identifizier]

*Listing 3: Webrouten der Datenrepresentation*

- (1.1) / (1.2) Über diese Route werden die Daten für die letzte Position der Messstationen aufgerufen. Dieser Datensatz trägt die Daten, die zur Anzeige der Positionen der Messstationen auf der Weltkarte benötigt werden und die Zusatzinformationen zur Darstellung eines Tooltips der einzelnen Argo-Floats. Die Daten werden bei jedem Besuch der Webseite ausgeliefert. Da sich diese erst verändern, wenn neue Daten vorliegen, werden diese über einen Caching-Mechanismus vorgehalten. Der optionale Übergabeparameter `force_reload` ermöglicht das Neuanlegen des Caches. Um Vandalismus vorzubeugen, sollte es sich dabei um einen nicht zu erratenden Token handeln.
- (1.3) Über diese Route werden Mess-, wie auch Metadaten einer Messstation angefordert. Die Auswahl der Boje erfolgt über den Übergabeparameter `identifizier`. Hier wird die eindeutige Identifikationsnummer (siehe `PLATTFORM_NUMBER` in Tabelle 1) der Messstation verwendet.
- (1.4) Um den Positionsverlauf einer Boje anzufordern, dient diese Route. Auch hier wird zur Identifikation der Messstation deren Identifikationsnummer als Parameter übergeben.

Die APP liefert die Teile aus, die vom Benutzer gesehen werden. Es handelt sich dabei um HTML/Javascript und Bilddaten. Die hierfür ausgearbeitete Struktur ist in Listing 4 zu sehen und im Folgenden beschrieben:

```

1 (2.1) GET      /
2 (2.2) GET      /chart/[identifizier]
3 (2.3) GET      /info/[identifizier]
```

*Listing 4: Webrouen der Darstellung*

- (2.1) Über diese Seite ist die eigentliche Applikation zu sehen. Dies umfasst die Karte sowie die angezeigten Messwerte.
- (2.2) Diese Route stellt den Plot der darzustellenden Messwerte als Bild zur Verfügung.
- (2.3) Über diese Route wird ein HTML für die Infoanzeige einer Messstation bereitgestellt.

## 4.2.4 Aussehen der Webapplikation

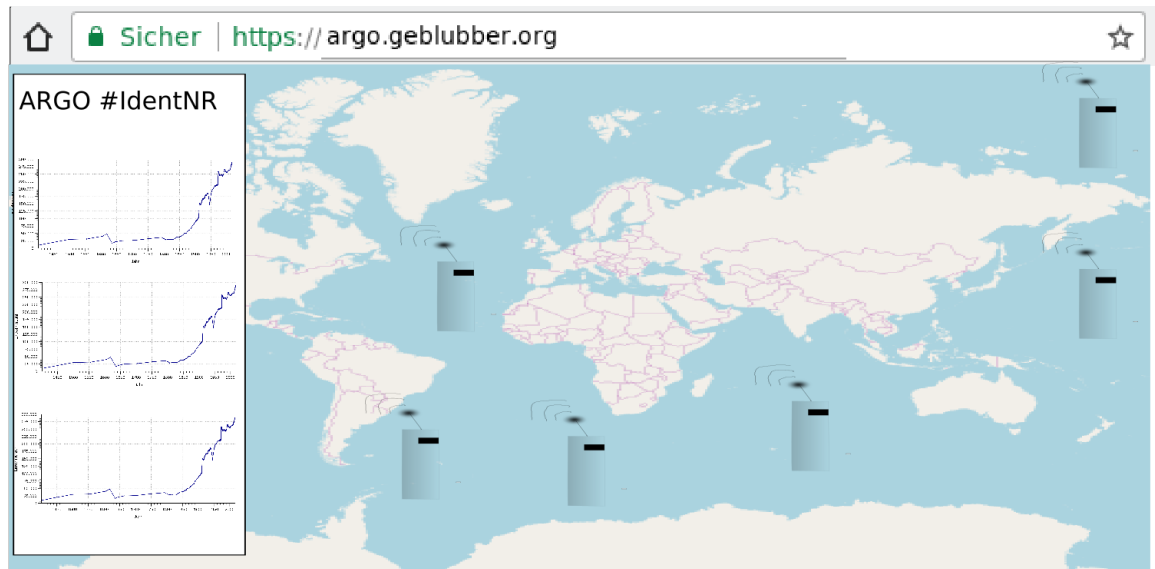


Abbildung 8: Grafischer Grobentwurf der Webapplikation

Das Aussehen der Applikation soll dem Muster von gängigen Webapplikationen entsprechen. In Abbildung 8 ist ein erster Grobentwurf der Applikation zu sehen. Zentrales Element der Applikation ist die Darstellung einer Karte. Über diese sollen die letzten Positionen der Karte ersichtlich sein. Klickt man eine Messstation an, so wird auf der linken Seite ein weiteres Element in die Seite eingefügt. Dieses dient zur Darstellung der Messwerte des jeweiligen Argo-Floats.

## 5 Implementierung

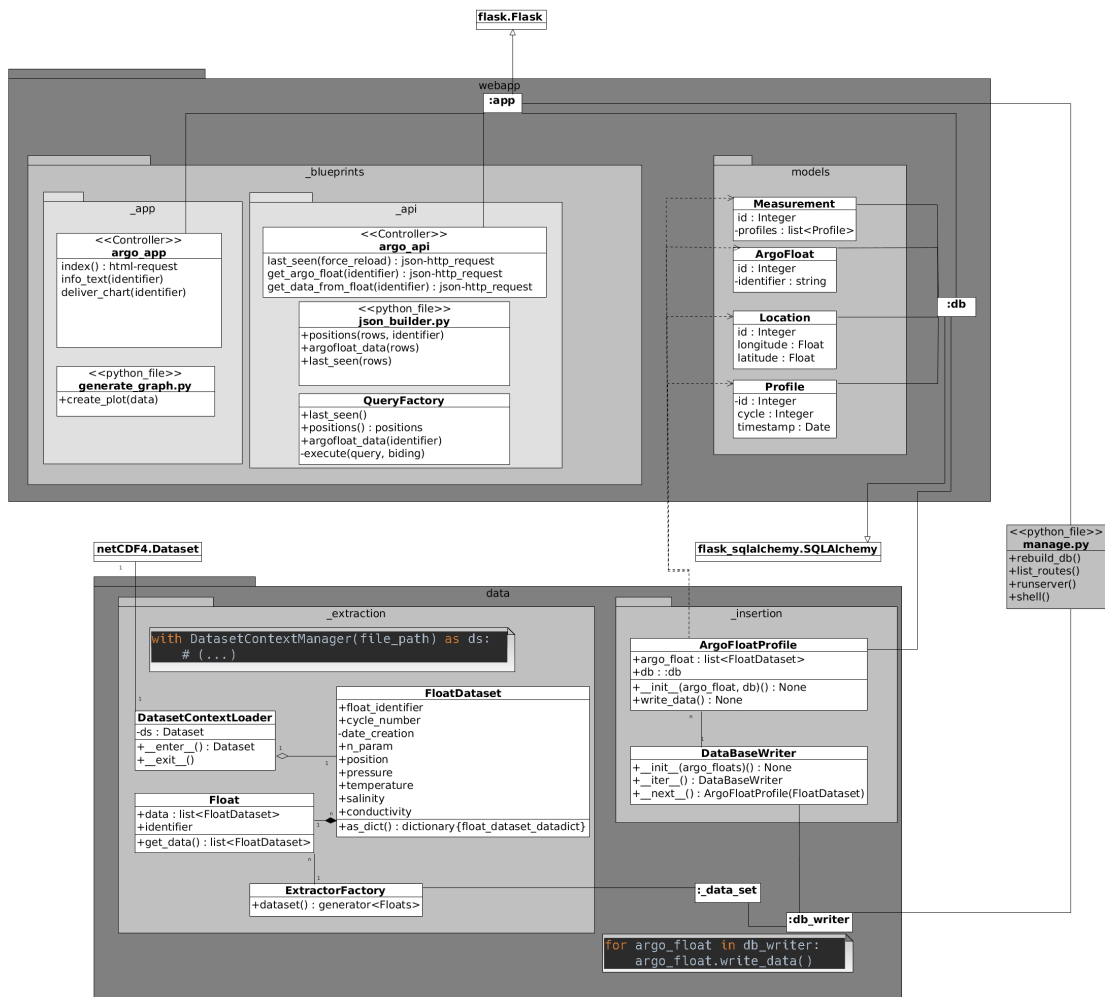


Abbildung 9: Architekturbeschreibung von ArgoData

### 5.1 Datenaggregation

#### 5.1.1 Auslesen der Daten

Der Datensatz besteht vor der Verarbeitung aus einer Ansammlung von Verzeichnissen und Dateien. Da die Gefahr besteht, dass geöffnete Dateien nicht wieder ordnungsgemäß geschlossen werden, muss eine Schnittstelle geschaffen werden, die unsachgemäße Verwendung der Dateien verhindert. Python sieht für diesen Zweck den Kontextmanager vor. In Listing 5 ist die hier verwendete Implementierung zu sehen.

```

1 from netCDF4 import Dataset
2
3 class DatasetContextManager(object):
4     def __init__(self, file_path):
5         self.ds = Dataset(file_path)
6
7     def __enter__(self):
8         return self.ds
9
10    def __exit__(self, exc_type, exc_val, exc_tb):
11        self.ds.close()

```

*Listing 5: Kontextmanager zur Sicherstellung der richtigen Dateibehandlung.*

In Listing 6 ist die Verwendung des Kontextmanagers aufgezeigt und im Folgenden beschrieben.

```

1 with DatasetContextManager(file_path) as ds:
2     juld = ds.variables['JULD'][0]

```

*Listing 6: Die Verwendung des Kontextmanagers*

Durch die Verwendung der Struktur über das Schlüsselwort `with` wird die Objektmethode `__enter__` aufgerufen. Diese gibt den objekteneigenen `netCDF`-Datensatz zurück. Dieses wird in der Verwendung im gleichnamigen Objekt `ds` gespeichert. Im darauf folgenden Operationsblock kann das Objekt nun verwendet und dessen Daten extrahiert werden. Nachdem der Operationsblock verlassen wurde, wird die Methode `__exit__` des Kontextmanagers aufgerufen. Innerhalb dieser Methode wird die Datei nun geschlossen. Durch dieses Entwurfsmuster ist die richtige Verwendung der Datei also transparent sichergestellt. Ein Entwickler kann somit nicht durch Unachtsamkeit vergessen, den Datensatz zu schließen. Gleichzeitig erhöht sich die Lesbarkeit. Durch den Operationsblock ist jederzeit ersichtlich, ab welcher Stelle im Quelltext der Datensatz nicht mehr verfügbar ist. Diese Schnittstelle wird in den Objektrepräsentationen der `Argo-Floats` und der Datensätzen verwendet, um auf die Daten zuzugreifen. In diesen wird die in Kapitel 4.2.1 gezeigte Datenstruktur vorgehalten.

### 5.1.2 Schreiben der Daten

Um die Verarbeitung der Daten sowie den Prozess der Aggregation in der Datenbank richtig zu modellieren, ist es sinnvoll, den Prozess als Sequenz zu modellieren. Dies erlaubt es, die Datenstruktur über einen Generator vorzuhalten. Python sieht dafür den Iterator vor. Die hier verwendete Implementierung ist in Listing 7 zu sehen. Das Interface wird über die Funktion `def __next__(self)` realisiert. Dieses delegiert die Iteration zum Objekteigenen Generator `self.argo_floats`. In dem Moment, in dem ein Objekt

aus dem Generator angefordert wird, findet die Verarbeitung der Datenstruktur statt. Zum Schluss wird der Datensatz über ein Objekt ausgeliefert, das es erlaubt, die Daten in die Datenbank zu überführen. Damit ist sichergestellt, dass der Prozess nur als Sequenz verwendet werden kann.

```

1 from ..argo_float_profile_writer import ArgoFloatProfile
2
3
4 class DataBaseWriter:
5     def __init__(self, argo_floats, db, app):
6         """
7         Interface to write the data as sequence.
8
9         :param argo_floats: generator - Graph of the representation of
10         ↪ the Data.
11         :param db: ORM - Flask-SQLAlchemy
12         :param app: Flask-app
13         """
14         self.argo_floats = argo_floats
15         self.db = db
16         self.app = app
17
18     def __iter__(self):
19         return self
20
21     def __next__(self):
22         """
23         Follow the sequence.
24
25         :return: ArgoFloatProfile
26         """
27         self.argo_float = next(self.argo_floats)
28         return ArgoFloatProfile(self.argo_float, self.db, self.app)

```

*Listing 7: Implementierung des Iterators zur Steuerung der Aggregationssequenz*

Die Verwendung der Schnittstelle ist in Listing 8 aufgezeigt.

```

1 from data import db_writer
2
3 for argo_float in db_writer:
4     argo_float.write_data()

```

*Listing 8: Verwendung der Schnittstelle zur Steuerung der Datenaggregation*

Eine zentrale Problemstellung im Prozess der Überführung der Daten in das relationale Schema ist die effektive Verwendung des flüchtigen Speichers. Eine klassische, sequenzielle Verarbeitung würde die Daten initial auslesen und diese in Gänze im Arbeitsspeicher vorhalten, bevor diese über den Mapper in die Datenbank überführt werden. Dieser Prozess wurde hier durch die Verwendung von Generatoren aufgebrochen. In Listing 9 ist die Implementierung dieser Schnittstelle zu sehen. Da in der Comprehension runde statt eckige Klammern verwendet werden, wird initial nur die



logische Struktur als Sequenz vorgehalten. Die inhärenten Float-Objekte des Generators werden zu dem Zeitpunkt erzeugt, wenn diese durch eine Iteration aufgerufen werden. Damit werden die Daten erst zu dem Zeitpunkt ausgelesen, wenn diese für die weitere Verarbeitung benötigt werden.

Die Methode `get_data_sets()` erzeugt aus jedem Unterordner im definierten Arbeitsverzeichnis ein Float-Objekt und gibt dieses über das Schlüsselwort `yield` zurück. Durch diese Klasse wird somit ein Generator definiert, der es erlaubt, alle im Arbeitsverzeichnis definierten Argo-Float-Datenobjekte bereitzustellen, ohne diese bereits bei der Instantiierung kennen und abarbeiten zu müssen.

```

1 import os
2
3 from ._float import Float
4
5
6 class ExtractorFactory:
7     def __init__(self, data_directory):
8         """
9         Interface to get a extractor-generator-object.
10
11         :param data_directory: Directory to the data of all ArgoFloats.
12         """
13         self.extractor_generator = (
14             Float(os.path.join(data_directory,
15                               profile_directory))
16             for profile_directory in os.listdir(data_directory))
17
18         self.__sum_floats = sum([os.path.isdir(os.path.join(
19             ↪ data_directory, d)) for d in os.listdir(data_directory)])
20
21     def get_data_sets(self):
22         """
23         Getting data.
24
25         :return: generator - LazyLoading Graph of all Datasets of all
26         ↪ ArgoFloats.
27         """
28         return self.extractor_generator
29
30     def float_count(self):
31         """
32         Counting the ArgoFloats.
33
34         :return: int - Number of directories in the Data-folder
35         """
36         return self.__sum_floats

```

*Listing 9: Factory zur Extraktion der Datensätze*

## 5.2 Webapplikation

### 5.2.1 Objektrelationales Mapping

Als objektrelationaler Mapper wird SQLAlchemy eingesetzt. Diese Software gilt als erprobt und wird bereits in einer Vielzahl von Softwaresystemen eingesetzt. So verwendet unter anderem reddit oder die Mozilla Foundation (vgl. [Org]) SQLAlchemy als Schnittstelle zur Datenhaltung. SQLAlchemy wird für die Verwendung in Flask empfohlen und es existiert eine Erweiterung, um die Verwendung des Mappers in Flask zu vereinfachen (vgl. [Ron11] S. 33).

In SQLAlchemy werden die Tabelleneinträge in Modellklassen abgebildet. In Listing 10 ist die Implementierung der Modell-Klasse für eine ArgoFloat-Messstation zu sehen.

```

1 from . import db
2
3
4 class ArgoFloat(db.Model):
5     """Representation of one ArgoFloat."""
6     __tablename__ = 'argo_floats'
7
8     measurements = db.relationship('Measurement', backref='argo_floats',
9     ↪ lazy='dynamic')
10
11     id = db.Column(db.Integer, primary_key=True)
12     identifier = db.Column(db.String(10))
13     project_name = db.Column(db.String(100))
14     launch_date = db.Column(db.Date)
15     float_owner = db.Column(db.String(100))
16
17     def __init__(self, identifier, project_name, launch_date,
18     ↪ float_owner):
19         self.identifier = identifier
20         self.project_name = project_name
21         self.launch_date = launch_date
22         self.float_owner = float_owner
23
24     def __repr__(self):
25         return f'<Argo Float {self.id!r}>'
26
27 class ArgoFloatTmpTable(ArgoFloat):
28     __bind_key__ = 'data_input'

```

Listing 10: Modellklasse für ein Argo-Float

Die Registrierung der Modell-Klasse erfolgt über die Vererbung der Metaklasse `db.Model`. Die Eigenschaften der jeweiligen Entitäten werden über Attribute des Objektes implementiert. Diese werden in Instanzen von `db.Column` transferiert. Dies erlaubt das Festsetzen des Datentyps. So lassen sich auch weitere Datenspezifikationen definieren. Die Attribute werden durch eine Parameterübergabe in den Initiator der Klasse mit

Werten versehen. Um das Binding umzusetzen, wurden Nachfahren der Modell-Klassen erzeugt. Diese benötigen für die Zuordnung das Attribut `__bind_key__`.

Über das Schlüsselwort `db.relationship` werden Beziehungen beschrieben. In diesem Fall besteht eine 1 – N Beziehung zu `measurements` (`argo_float ← measurements`) (siehe auch Abbildung 6). Als Übergabeparameter wird ein Name für die Beziehung erwartet. Der Parameter `backref` gibt den Namen der Klasse auf dem Mapper an. Diese Einstellung erlaubt den Aufruf der Beziehung auch in die entgegengesetzte Richtung. Der Parameter `lazy` definiert die verwendete Strategie für das Lazyloading. In diesem Fall wurde sich für `'dynamic'` entschieden. Diese Einstellung gibt bei Lesezugriffen ein vorkonfiguriertes Query-Objekt zurück. Dies erlaubt das Hinzufügen weiterer Filter vor dem Zugriff auf die Tabellen. Die durch diese Beziehung verbundene Modell-Klasse `Measurements` ist in Listing 11 zu sehen.

```

1 from . import db
2
3
4 class Measurement(db.Model):
5     """Representation of one Measurement."""
6
7     __tablename__ = 'measurements'
8
9     id = db.Column(db.Integer, primary_key=True)
10
11     profiles = db.relationship('Profile', backref='measurements', lazy='
↪ dynamic')
12
13     argo_float_id = db.Column(db.Integer, db.ForeignKey('argo_floats.id'
↪ ))
14     argo_float = db.relationship('ArgoFloat')
15
16     location_id = db.Column(db.Integer, db.ForeignKey('locations.id'))
17     location = db.relationship('Location')
18
19     def __init__(self, argo_float, location):
20         self.argo_float = argo_float
21         self.location = location
22
23     def __repr__(self):
24         return f'<Measurement {self.id!r}>'
25
26
27 class MeasurementTmpTable(Measurement):
28     __bind_key__ = 'data_input'

```

Listing 11: Modellklasse für eine Messung

Auf dieser Seite der Beziehung (`measurement → argo_float`) unterscheidet sich deren Implementierung. Für die eindeutige Zuordnung des übergeordneten Argo-Floats wird deren `id` als Fremdschlüssel registriert. Die Beziehung benötigt neben dem Namen der anderen Seite keine weiteren Parameter, da die Beziehung bereits konfiguriert worden

ist.

Das Schreiben von Daten über den Mapper SQLAlchemy ist Teil der Datenaggregation. Der dort implementierte Programmcode ist in Listing 12 zu sehen.

```

1 from webapp.models import ArgoFloat, Location, Measurement, Profile
2
3
4 class ArgoFloatProfile:
5     def __init__(self, argo_float, db, app):
6         """
7         Writes the Profile of one ArgoFloat into the database.
8
9
10        :param argo_float: Representation of the extracted Data of one
11        ↪ ArgoFloat.
12        :param db: ORM - Flask-SQLAlchemy
13        :param app: Flask-app
14        """
15
16        self.argo_float = argo_float
17        self.db = db
18        self.app = app
19
20    def write_data(self, bind=None):
21        """
22        Triggers the data-transfer.
23        :param bind: str - Define a binding.
24        """
25        session = self.db.create_scoped_session(
26            options={
27                'bind': self.db.get_engine(self.app, bind),
28                'binds': {}
29            }
30        )
31        try:
32            argo_float_ = ArgoFloat(
33                identifier=self.argo_float.identifier,
34                project_name=self.argo_float.project_name,
35                launch_date=self.argo_float.launch_date,
36                float_owner=self.argo_float.float_owner
37            )
38            for profile_data in self.argo_float.data:
39                location_ = Location(
40                    latitude=profile_data.position['latitude'],
41                    longitude=profile_data.position['longitude']
42                )
43                measurement_ = Measurement(
44                    argo_float=argo_float_,
45                    location=location_
46                )
47
48                session.add(Profile(
49                    cycle=int(profile_data.cycle_number),
50                    timestamp=profile_data.date_creation,
51                    measurement=measurement_,
52                    salinity=profile_data.salinity,
53                    pressure=profile_data.pressure,
54                    temperature=profile_data.temperature,
55                    valid_data_range=profile_data.valid_data_ranges

```

```

56         ))
57
58         session.commit()
59     except Exception as err:
60         session.rollback()
61         raise err

```

*Listing 12: Das Schreiben der Daten eines Argo-Floats in die Datenbank*

Die hier implementierte Klasse `ArgoFloatProfile` ist für das Schreiben aller Datensätze eines `ArgoFloats` zuständig. Diese verwendet für die Zuordnung der Daten die Modell-Klassen aus der Webapplikation. Durch die Methode `write_data` werden die Daten des durch `__init__` übergebenen Datensatzes in die Datenbank überführt. Um ein dynamisches Binding realisieren zu können, wird der betreffende Parameter `bind` übergeben. Mithilfe dieses Parameters wird eine Session aufgebaut, welcher das Schreiben in die Datenbank erlaubt, die über das Binding definiert ist.

Für jeden Datensatz wird eine Instanz der betreffenden Modell-Klasse erstellt. Die dafür benötigten Daten werden aus den Datensätzen des klasseneigenen Generator-Objektes `argo_float` angefordert und als Parameter übergeben.

Der zuvor definierten Session werden nur die Instanzen der Profile-Klassen übergeben. Da alle zum Messprofil gehörenden Modell-Klassen in diesem Kontext eindeutig sind, kann SQLAlchemy diese selbstständig zuordnen. Zum Abschluss werden die Daten über `session.commit()` in die Datenbank überführt. `session.rollback()` erlaubt es, die Veränderungen, die innerhalb dieser Session an der Datenbank herbeigeführt wurden, wieder auf den Urzustand zurückzuführen.

SQLAlchemy bietet eine eigene Abfrage-Sprache an. In Listing 13 ist eine vereinfachte Implementierung der Abfrage zur Bestimmung der Positionshistorie einer Argo-Float zu sehen, wie sie auch in der Web-API implementiert ist.

```

1 from webapp import db
2 from webapp.models import Measurement, Location, Profile, ArgoFloat
3
4 query = db.session.query(ArgoFloat, Location, Profile) \
5         .join(Measurement) \
6         .join(Location) \
7         .join(Profile) \
8         .filter(ArgoFloat.identified == '1900037') \
9         .order_by(Profile.timestamp)
10
11 result = db.session.execute(query, None, bind=db.get_engine(app, None))
12
13 data = [row for row in result]

```

*Listing 13: Anfragen über SQLAlchemy zum Lesen von Daten*

In diesem Beispiel werden über eine Projektion auf Attribute von ArgoFloat, Location und Profile Daten angefordert. SQLAlchemy erlaubt die Definition von Joins über mehrere Tabellen. Diese werden durch die in den Modell-Klassen definierten Beziehungen aufgelöst. Die Selektion erfolgt durch die Methode `query.filter()`. In diesem Fall werden nur Relationen der Argo-Float mit dem Identifier '1900037' ausgewählt. Die Methode `query.order_by()` erlaubt das Sortieren der Ergebnisse. In diesem Fall werden die Datensätze durch das Attribut `Profile.timestamp` chronologisch sortiert.

SQLAlchemy ist auch in der Lage, in SQL-Sprache definierte Anfragen zu verarbeiten. In Listing 14 ist eine vereinfachte Implementierung aus der Web-API zu sehen. Diese extrahiert die letzten Positionen mit den dazugehörigen Zeitpunkten aus der Datenbank.

```

1 from webapp import db
2
3 sql_template = """SELECT DISTINCT ON (f.id)
4     f.identifizier,
5     l.longitude,
6     l.latitude,
7     p.timestamp
8 FROM argo_floats f
9     JOIN measurements m
10        ON f.id = m.argo_float_id
11     JOIN profiles p
12        ON m.id = p.measurement_id
13     JOIN locations l
14        ON l.id = m.location_id
15 ORDER BY f.id, p.cycle DESC;"""
16
17 result = db.engine.execute(sql_template)

```

*Listing 14: Das Ausführen von SQL Anfragen über SQLAlchemy*

Über die Methode `db.engine.execute()` können SQL-Queries direkt verarbeitet werden. Dies hat aber zwei entscheidende Nachteile. Der Query-Dialekt beschränkt die Anfrage auf ein bestimmtes DBMS. Diese Anfrage ist für PostgreSQL entworfen und würde auf einem anderen DBMS nicht funktionieren. Zum anderen würden Eingaben nicht gegen SQL-Injections gesichert werden. Eine Abfrage wie `db.engine.execute(f'↪ SELECT * FROM argo_floats WHERE argo_floats.identifizier={user_input})` würde also unabschätzbare Sicherheitsrisiken mit sich bringen.

Flask bietet für die Einbindung von SQLAlchemy eine Erweiterung an. Durch diese Erweiterung wird eine SQLAlchemy-Instanz durch ein zentrales Objekt (`db`) repräsentiert. Dies ermöglicht eine einfache Datenabstraktion und ist für Flask typisch. Dieser Mechanismus wird durch die zeitkritische und in ihrem Ablauf fest vorgeschriebene Er-

stellung von Objekten und Submodulen erkaufte. Dieser Prozess ist bei der Einbindung von flask-sqlalchemy gut sichtbar. Aus diesem Grund wird die Initialisierung dieser Erweiterung hier gesondert dargestellt.

In Abbildung 5.2.1 ist eine sequentielle Darstellung des Prozesses zu sehen und wird im Folgenden beschrieben.

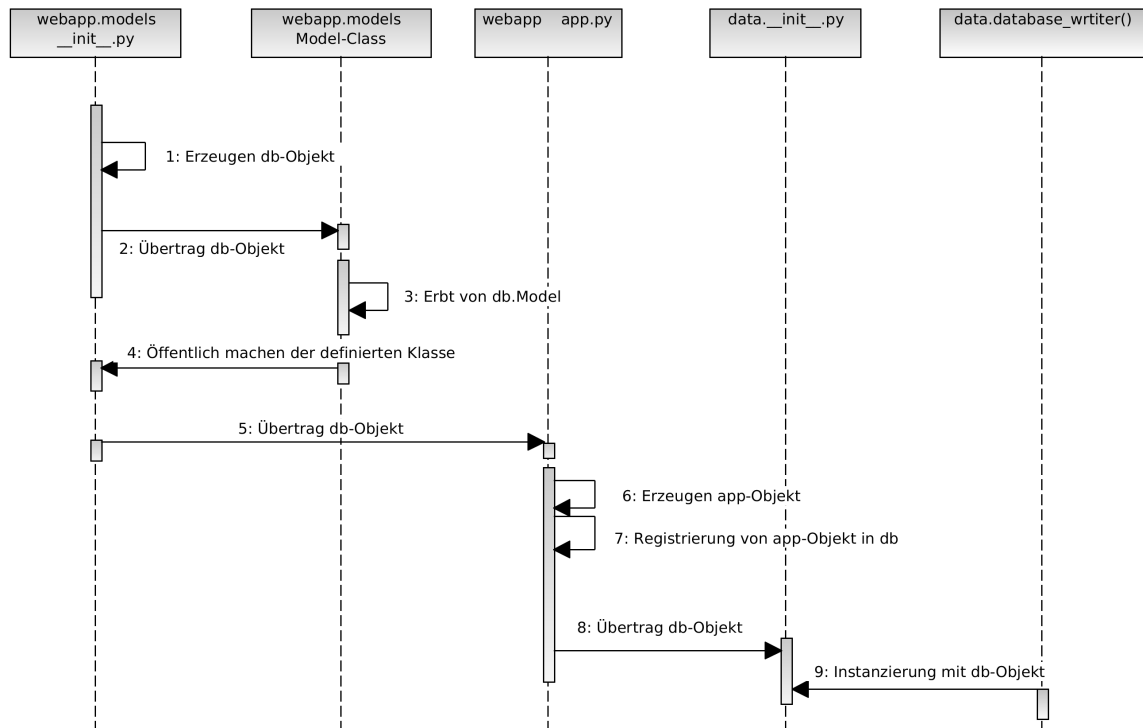


Abbildung 10: Sequenzdiagramm der Einbindung von SQLAlchemy

1. In der Datei `webapp.models.__init__.py` muss sichergestellt werden, dass die SQLAlchemy-Instanz noch vor der Initiierung der Modell-Klassen erzeugt wurde.
2. Nun kann das db-Objekt in die Modellklassen übertragen werden. Dafür importieren diese db in ihren Scope.
3. Die Modell-Klassen werden in db registriert. Dafür erben diese von der abstrakten Klasse `db.Model`. Siehe auch Listings 10 und 11.

```

1 from flask_sqlalchemy import SQLAlchemy
2
3 db = SQLAlchemy()
4
5 from ._measurement import Measurement
6 from ._argo_float import ArgoFloat
7 from ._location import Location
8 from ._profile import Profile
  
```

---

*Listing 15:* webapp.models.\_\_init\_\_.py

4. Die Modell-Klassen dürfen erst nach der Instantiierung von db über den Scope des Submodules heraus bekannt gemacht werden, um sicherzustellen, dass dieses bereits existiert. Die Implementierung des Ablaufs ist in Listing 15 ersichtlich.
5. Die Instanz db kann nun mit den in ihr registrierten Modell-Klassen aus dem Submodul in die Datei webapp.\_\_init\_\_.py importiert werden.
6. Hier wird nun das globale Objekt der Flask-Instanz (app) erstellt und konfiguriert.
7. Anschließend wird app in db registriert. Die Schritte 5 bis 7 können über das Listing 16 nachvollzogen werden.
8. Die Instanz db aus dem Scope webapp.db ist nun fertig konfiguriert und kann für das Lesen und Schreiben in die Datenbank verwendet werden.



### 5.2.2 Controller

```

1 from flask import Flask, url_for
2 from flask_twisted import Twisted
3
4 from .models import db
5
6 app = Flask(__name__, static_url_path='/static')
7 app.config.from_pyfile('./argo.cfg')
8
9 db.init_app(app)
10
11 from .blueprints import argo_api, argo_app
12
13 app.register_blueprint(argo_api)
14 app.register_blueprint(argo_app)
15
16 from . import models
17
18 twisted = Twisted(app)

```

Listing 16: webapp.\_\_init\_\_.py

Über die globale Instanz des Controllers (app) kann auf Daten zugegriffen und die Konfiguration vorgenommen werden. In Listing 16 ist die Initialisierung des Controllers zu sehen.

Über Blueprints wird die in Kapitel 4.2.3 ausgearbeitete logische Trennung zwischen app und api realisiert. Die hier definierten Module umfassen auch die in diesem Kontext benötigten Hilfsprogramme. Die hier implementierte Modulstruktur von argo\_api und argo\_app ist im Folgenden ersichtlich.

#### argo\_api

**query\_factory** Alle lesenden Zugriffe zur Datenbank werden über eine QueryFactory zentral zusammengefasst. Die hier ausgearbeitete Implementierung ist in Kapitel 5.2.1 ausführlich beschrieben.

**json\_builder** Durch diese Programmabschnitte werden die Daten in Listen und Dictionaries überführt. Die Variablen und Datenstrukturen werden dabei so angelegt, dass diese direkt als JSON bzw. geoJSON ausgeliefert werden können.

Dieses Controller-Blueprint fordert die Daten über die query\_factory an und bringt sie über den json\_builder in das vorgesehene Format. Die Umwandlung in JSON und die Erstellung des Requests erfolgt über flask jsonify.

**argo\_app**

**generate\_graph** Dieser Programmteil erzeugt einen Funktionsplot über die Bibliothek `matplotlib`. Eine nähere Beschreibung der Erstellung findet sich in Kapitel 5.2.5.

In diesem Blueprint werden Templates zu HTML-Dateien zusammengesetzt. Dafür benötigt dieser Programmteil Zugriff auf die Verzeichnisse für die jeweiligen Templates und statischen Dateien. Diese Daten werden über `flask.render_template` zusammengesetzt und ausgeliefert.

Die so erstellten Blueprints werden im zentralen Controller (app) über das Schlüsselwort `app.register_blueprint` registriert und zusammengefasst. Als zentrale Steuerungseinheit für den gesamten Programmablauf dient die Datei `manage.py`. Die Parameter dieser Datei werden im Folgenden kurz beschrieben:

**runserver** Durch diesen Parameter wird der Server der Applikation gestartet.

**rebuild\_db** Dies stößt den Prozess der Datenaggregation an. Streng genommen ist dies nicht Teil des Controllers. Aus Gründen der Einfachheit wurde dieser Programmablauf aber in die Steuerungseinheit integriert.

**shell** Durch diesen Parameter wird eine IPython-Shell gestartet. In diesem repl ist die Webapplikation mit all ihren Umgebungsvariablen initiiert. Diese Shell dient für das Debugging der Applikation.

**list\_routes** Dieser Parameter listet alle Routen, die durch den Controller definiert sind auf der Kommandozeile auf.

**5.2.3 Templates**

Für das Erstellen der HTML-Dateien wurde die Template-Engine Jinja2 verwendet. Diese ist gut in Flask integriert. Jinja2 erlaubt das Modularisieren von HTML-Dateien, den Zugriff auf Variablen und übergebenen Daten sowie die Abarbeitung einfacher Strukturen und Wahrheitswerte.

Die Templatestruktur wurde in dieser Applikation stark modularisiert. In Listing 17 ist der Einstieg in die Modulstruktur zu sehen und im Folgenden beschrieben.

```

1 {% extends '_base.html' %}
2
3 {% block map %}
4     {% include '_render_map.html' %}
5 {% endblock %}

```

Listing 17: `webapp.templates.map.html`

Die Datei `_base.html` ist für die Darstellung der Webseite verantwortlich. Diese wurde über Twitter Bootstrap realisiert. Als Vorlage diente dabei ein bereits ausgearbeitetes 2-Column Layout aus [Ng14]. Der Beispielcode wurde an die Anforderung der Applikation angepasst. Der Code wurde dabei in die logischen Bestandteile zerlegt und in die modulare Struktur der Webseite überführt. Der Code ist unter einer MIT Lizenz veröffentlicht. Damit kann der Code unter Nennung der Lizenz verwendet werden. Deswegen wurde ein Tag in das HTML Element eingesetzt, um Autor und Lizenz zu nennen.

#### 5.2.4 Kartendarstellung

OpenLayers ist eine Programmbibliothek um interaktive Geoapplikationen zu entwickeln. Das Framework ist in Javascript entwickelt und nimmt alle benötigten Berechnungen auf Clientseite vor. Die Bibliothek erlaubt es, Kartenmaterial aus verschiedenen Quellen zu rendern. Dabei können Kachel- oder auch Vektorbasierte Materialien eingebunden werden. Die Elemente der Karte setzen sich, wie in Listing 18 zu sehen, zusammen.

```

1  var map = new ol.Map({
2    layers: [mapVectorLayer, argoFloatsLayer],
3    target: 'map',
4    controls: ol.control.defaults({
5      attributionOptions: false,
6      zoom: false,
7    }),
8    view: new ol.View({
9      center: [0, 0],
10     zoom: 3.5,
11     minZoom: 3
12   })
13 });

```

*Listing 18: Das ol.Map Element aus der Kartendarstellung*

Die Darstellung basiert auf zwei Layern, einer Vektordarstellung der Kontinent- und Landesgrenzen (`mapVectorLayer`), sowie der Darstellung der ArgoFloats (`argoFloatsLayer`). Die Darstellung der Steuereinheiten wurde deaktiviert. Als Startposition wurden Längen- und Breitengrad mit jeweils 0 gewählt. Die Zoomstufe ist mit 3 initialisiert und es ist eine maximale Zoomstufe von 3.5 gewählt. Die Werte für die Zoomstufe wurden bei einer Auflösung von 1920x1080 ermittelt und getestet. Die Karte bettet sich in das HTML-Element `map` ein und wird mit diesem zur Anzeige gebracht. Steuergesten mit Maus und Tastatur nimmt die Karte an dieser Stelle bereits entgegen.

Der `mapVectorLayer` wurde aus der GeoJSON-Datei `countries.geo.json` erstellt. Die

Datei wurde von [Sun16] heruntergeladen. Diese steht unter der Lizenz UNLICENSE und kann damit ohne Einschränkungen verwendet werden. Die Darstellung der Argo-Floats wird ebenso über eine GeoJSON realisiert. Diese wird auf dem Webserver über die API bereitgestellt und wird über der URL /last\_seen (siehe Listing 3) angefordert. Die so angeforderte Datenstruktur hat ein spezifisches Format. In Listing 19 ist das Format abgekürzt ersichtlich.

```

1  {
2  "crs": {
3    "properties": {
4      "name": "EPSG:4326"
5    },
6    "type": "name"
7  },
8  "features": [
9    {
10     "geometry": {
11       "coordinates": [
12         -52.02875,
13         12.92591
14       ],
15       "type": "Point"
16     },
17     "properties": {
18       "feature_type": "latest_position",
19       "identifier": "1901673",
20       "last_seen": "Sun, 10 Feb 2013 00:00:00 GMT"
21     },
22     "type": "Feature"
23   },
24   // (feature_2, ..., feature_n)
25 ]
26 }
```

*Listing 19: Gekürzte geoJSON zur Darstellung der Argo-Floats*

Der Parameter crs (Coordinate Reference System Objects) bestimmt die Art der Kartenprojektion. Durch das property EPSG:4326 wurde der GPS-Standard WGS84 - World Geodetic System 1984 eingesetzt. Dies deckt sich auch mit den durch die Messstationen bereitgestellten Geo-Koordinaten. Für die Darstellung werden diese Koordinaten in eine Merkatorarstellung (EPSG:3857) projiziert.

Die Repräsentationen der Messstationen finden sich im Array features. Im Feld geometry werden Position und die Form des Features eingestellt. Über properties werden die zusätzlichen Daten feature\_type, identifier und last\_seen in diesem Format gespeichert. Openlayers kann ein derartig formatiertes GeoJson selbstständig zur Anzeige bringen. Wie die Datei geladen wird, ist in Listing 20 zu sehen.

```

1 var argoFloatsLayer = new ol.layer.Vector({
2   style: FloatStyle,
3   source: new ol.source.Vector({
4     format: new ol.format.GeoJSON(),
5     url: "/last_seen"
6   })
7 });

```

Listing 20: Die Funktion *argoFloatsLayer*

Als Quelle für die Anzeigedaten wird hierbei ein `ol.source.Vector`-Objekt gewählt. Das Format und die `url` für die spezifizierte Datenquelle werden als Parameter übergeben. Damit werden die ArgoFloats auf einer vektorbasierten Karte angezeigt. Was zu diesem Zeitpunkt noch fehlt, ist die Möglichkeit mit den ArgoFloats zu interagieren. Beim Hovern mit dem Mauszeiger über einer ArgoFloat soll ein Tooltip mit einigen Informationen erscheinen und die Messstation optisch hervorgehoben werden. Die Hervorhebung wird durch den Befehl `map.addInteraction(hoverInteraction)` in der Karte registriert. `overInteraction` ist eine Instanz von `ol.interaction.Select` und erlaubt die Auswahl von Features und des Event-Typs. In diesem Fall werden alle Features des `argoFloatsLayer` ausgewählt, die unter dem Mauszeiger liegen. Openlayers wählt mit dieser Konfiguration einen Vergrößerungseffekt um das Feature hervorzuheben. Um den Tooltip zur Anzeige zu bekommen, wurde eine weitere Funktion definiert. In Listing 21 ist die Funktion für das Fangen eines `pointermove`-Events zu sehen.

```

1 map.on('pointermove', function (evt) {
2   if (evt.dragging) {
3     info.tooltip('hide');
4     return;
5   }
6   displayFeatureInfo(map.getEventPixel(evt.originalEvent));
7 });

```

Listing 21: Das Abfangen eines *pointermove*-Events

Die Funktion wartet auf definierte Eingaben. In diesem Fall wird der Funktionskörper durch das Bewegen des Mauszeigers über der Kartendarstellung ausgelöst. Der Event wird als `evt` in die innere Funktion überführt. Dort wird überprüft, ob es sich um einen `dragging`-event handelt. Ist dies der Fall, wird der DIV-Container mit der `id #info` (`info`) ausgewählt und der darin liegende Tooltip über den CSS-Parameter `hide` versteckt. Diese Einstellung wurde gewählt, um störende Tooltips bei der Navigation mit der Karte zu unterbinden. In jedem anderen Fall wird der Pixel an der Spitze des Mauszeigers extrahiert und der Funktion `displayFeatureInfo` übergeben.

`displayFeatureInfo` positioniert das `info`-Element in der Nähe des Mauszeigers und extrahiert etwaige unter dem Pixel liegende Features. Anschließend werden die At-

tribute des Features untersucht. Über den Befehl `feature.get("feature_type") === '↪ latest_position'` werden Features aus dem `last_seen`-GeoJson identifiziert. Ist die Identifikation positiv, werden weitere Attribute extrahiert und in den Text des Tooltips überführt. Zum Abschluss wird der Tooltip über seine CSS-Attribute zur Anzeige gebracht.

Die Funktion kann auch die Features der Positionshistorie einer Boje über einen Tooltip anzeigen. Da sich die hier darzustellenden Parameter unterscheiden, werden dessen Daten gesondert abgefragt.

Die Interaktion über einen Klick passiert analog. In Listing 22 ist die hier verwendete Implementierung der Klick-Event-Abfrage gezeigt.

```

1 map.on('click', function (evt) {
2     displayArgoData(map.getEventPixel(evt.originalEvent));
3     displayPositions(map.getEventPixel(evt.originalEvent))
4 });

```

*Listing 22: Das Abfangen eines Klick-Events*

Auch an dieser Stelle wird die `map.on`-Funktion der Karte verwendet, um den Event abzufangen. Ist dieser vom Typ `'click'`, so werden die darauf folgenden Funktionen mit dem Pixel an der Position des Mauszeigers aufgerufen. Im Folgenden wird der Programmablauf der hier aufgerufenen Funktionen erläutert:

**displayArgoData** Diese Funktion ist für die Darstellung der Messdaten, sowie des Informationstextes der ausgewählten `ArgoFloat` verantwortlich. Aus diesem Grund besteht sie aus zwei Unterfunktionen.

**display\_chart** Diese Funktion stellt den Funktionsplot der Messwerte der ausgewählten `ArgoFloat` dar. Dafür wird am Anfang der `div`-Container mit der ID `#chart-picture` ausgewählt. In diesem Container wird später das Bild platziert. Um bereits bestehende Bilder zu löschen, werden alle HTML-Elemente innerhalb des Containers erstellt. Anschließend wird das Bild der Messdaten über die Webroute `/chart/[identifizier]` (siehe Listing 3) angefordert. Das Bild wird anschließend auf die Breite des Containers gebracht und der Bootstrap-Klasse `img-responsive` hinzugefügt. Damit wird das Bild formatiert über die Sidebar dargestellt.

**display\_info** Diese Funktion stellt einen Info-Block zu den Metadaten der Argo-Float dar. Analog zum oberen Prozess werden hier Container aus dem DOM-Tree selektiert und das sich darin befindliche HTML manipuliert. Die Daten wurden hier als HTML-Objekte über einen Ajax-Befehl angefordert und direkt in den umliegenden Container injiziert.

Die oben genannten Funktionen werden aufgerufen, wenn es sich bei dem angeklickten Element um ein Feature vom Typ Point handelt. Der Bereich des Graphen wird an dieser Stelle sichtbar gemacht und der Info-Bereich eingeklappt. Die Sichtbarkeit der Sidebar wird ebenso an dieser Stelle eingestellt.

**displayPositions** Diese Funktion stellt eine Positionshistorie einer spezifischen Messstation dar. Dafür wird ein GeoJSON über die URL `/positions/[identifizier]` angefordert. Das Sichtbar-Machen, der hier kodierten Daten geschieht analog zu der Darstellung aller Argo-Floats. Das GeoJson trägt zusätzlich noch ein `LineString`-Element, um den zeitlichen Verlauf der Positionen zu verdeutlichen.

### 5.2.5 Zeichnen der Graphen

Für die Darstellung der Messwerte wurde sich in dieser Applikation für die Python-Bibliothek `matplotlib` entschieden. Diese Plotting-Engine ist in der Entwicklung von wissenschaftlichen Applikationen in Python sehr verbreitet. Als Renderer wurde ein `AGG`-Renderer eingesetzt. Dieser erlaubt das Vorhalten der Bilddaten im Arbeitsspeicher, sowie deren Überführung in ein `Canvas`-Element.

In Listing 23 ist die Funktion des Controllers zu sehen, über die das Bild bereit gestellt wird.

```

1 @argo_app.route("/chart/<identifizier>")
2 def deliver_chart(identifizier):
3     url = url_for('argo_api.get_argo_float', identifizier=identifizier,
4     ↪ _external=True)
5     data = requests.get(url).json()
6
7     fig = create_plot(data)
8     canvas = FigureCanvas(fig)
9     png_output = BytesIO()
10    canvas.print_png(png_output)
11    response = make_response(png_output.getvalue())
12    response.headers['Content-Type'] = 'image/png'
13
14    return response

```

*Listing 23: Ausliefern eines Funktionsplots Flask*

Die Funktion wird im Blueprint `argo_app` der Flask-Applikation mit der Route registriert. Als Parameter `identifizier` wird die eindeutige Identifikationsnummer der Messstation erwartet. Die zum Erstellen des Graphen benötigten Daten werden aus der API angefordert. Dafür wird die URL der Datenschnittstelle der Methode `get_argo_float` in der API ermittelt. Anschließend wird unter Zuhilfenahme der Python-Bibliothek `requests` der Datensatz über einen GET-Request angefordert. Die Methode `.json()` überführt die Daten von `Json` in eine Python-Datenstruktur.

**create\_plot** Diese Funktion ist für die Erstellung des Funktionsplots zuständig. Als Theme wurde sich hier für `'seaborn-whitegrid'` entschieden. Eine der damit vorgegebenen Grundeinstellungen wurden durch das Ändern der `rcParams` noch weiter angepasst.

Für die Darstellung der drei Funktionsplots wurden jeweils ein Achsen-Objekt erstellt. Diese Achsen werden über ein `Gridspec` untereinander angeordnet.

Die Datensätze werden über die Y-Achse aufgetragen, während die X-Achse die Zeitstempel aufzeigt. Enthält ein Datensatz Werte keiner numerischen Zuordnung (`NaN`), so wird davon ausgegangen, dass dieser Datensatz nicht vorhanden ist und diese Achse nicht gezeichnet. Überschreitet ein Datensatz den ihm zugeschriebenen Wertebereich, so wird der Wertebereich der Y-Achse fest eingestellt. Die hierfür benötigten Daten werden über die zentrale Flask-Instanz aus der Konfiguration angefordert.

Um den Platz besser nutzen zu können, wird die Beschriftung der Y-Achse nur auf dem untersten Graphen aufgetragen. Dieser enthält die Werte für den Druck des umliegenden Wassers. Dieser Messwert sollte in jeder Boje enthalten sein, es ist also nicht davon auszugehen, dass dieser Wert nicht gezeichnet wird.

Die Achsen werden über das `Gridspec` in einer `Figure` zusammengefasst und damit aus dem Funktionskontext zurückgegeben.

Der so erstellte Funktionsgraph wird als `Canvas-Element` in einen `response` eingebunden und über diesen ausgeliefert.



## 6 Testen

### 6.1 Funktionstest

#### 6.1.1 Webapplikation

Um die Webapplikation zu testen, wurde ein Funktionstest durchgeführt. Dieser umfasst die in Kapitel 3.3.1 festgesetzten Anforderungen und ist im Folgenden bebildert beschrieben.

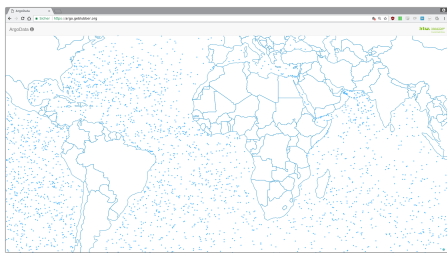


Abbildung 11: **Funktionstest I**  
Der Aufruf der Webapplikation

Wie in Abbildung 11 zu erkennen ist, werden die Grenzen der Ozeane und Kontinente über eine Weltkarte dargestellt, wenn die Webpräsenz der Applikation unter der für ArgoData registrierten Url aufgerufen wird. Die Karte ist über eine Vektorrepräsentation dargestellt, Landmarker wie Straßen und Städte sind darauf nicht zu erkennen. Die letzte Position der Messstationen ist über die jeweilige Position auf der Karte ersichtlich.

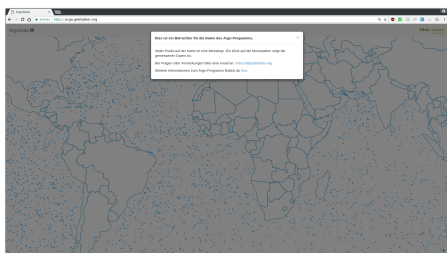


Abbildung 12: **Funktionstest II** Die Anzeige der Hilfe

Durch Anklicken des Infobuttons erhält der Benutzer über ein Modal eine Kurzhilfe angezeigt. Diese erklärt die grundlegende Funktion der Webseite, stellt Kontaktdaten bereit und stellt über einen Weblink eine Verbindung zum Argo-Programm her. In Abbildung 12 ist die Webseite mit aktiviertem Modal zu sehen.

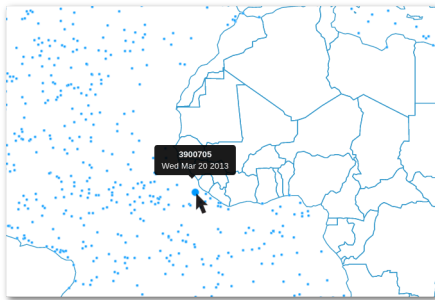


Abbildung 13: **Funktionstest III** Mousehover über Messstation

Wird der Mauszeiger über die Repräsentation einer Messstation geführt, so erscheinen grundlegende Daten der Argo-Boje. Dies umfasst den eindeutigen Identifier sowie das Datum des letzten Messvorgangs. Zu erkennen in Abbildung 13.

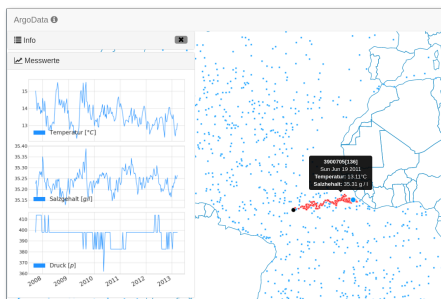


Abbildung 14: **Funktionstest IV** Werte werden angezeigt

Durch einen Mausklick auf die Kartendarstellung der Argo-Boje werden weiterführende Daten angefordert. Wie in Abbildung 14 zu sehen ist, werden Messwerte über Funktionsplots an der linken Seite der Webseite dargestellt. Dabei sind über die X-Achse der zeitliche Verlauf und über die Y-Achse der jeweilige Messwert kodiert. Die gemessenen Werte umfassen dabei Temperatur, Salzgehalt sowie Druck. Der Ort der Messwerterhebung wird durch einen Punkt auf der Karte repräsentiert.

Wird der Mauszeiger über diese Repräsentation geführt, werden neben dem Datum der Messung auch die dort erfassten Werte dargestellt.

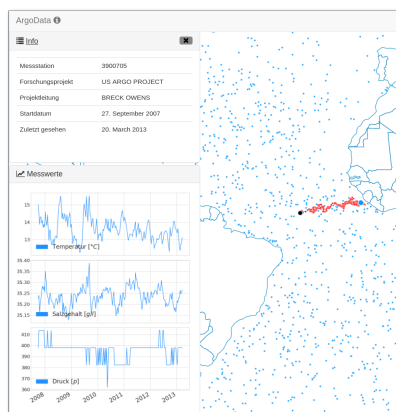


Abbildung 15: **Funktionstest V** Metainformationen werden angezeigt

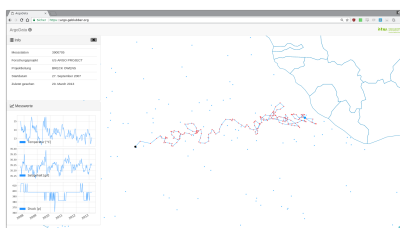


Abbildung 16: **Funktionstest VI** Der Pfad der Messstation kann zurückverfolgt werden

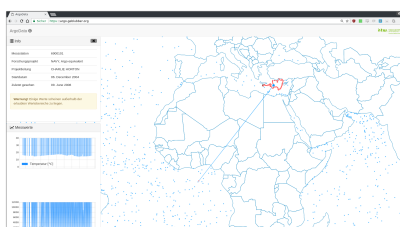


Abbildung 17: **Funktionstest VII** Fehlerhafte Daten werden markiert

Wird der Reiter „Info“ angeklickt, so werden weitere Informationen zur Messstation sichtbar. Wie in Abbildung 15 zu sehen ist, umfassen diese den Identifier der Messstation, das verantwortliche Forschungsprojekt, den Besitzer der Boje sowie das Startdatum und das Datum der letzten Messung.

Wie in Abbildung 16 zu erkennen ist, wird der zwischen zwei Messpunkten der zurückgelegte Weg über eine Linie dargestellt. Somit kann der Weg, den eine Messstation zurückgelegt hat um die Daten zu erheben, klar nachvollzogen werden.

In Abbildung 17 ist ersichtlich, dass fehlerhafte Daten durch eine Warnung hervorgehoben werden. Diese informiert die Benutzenden, dass die Daten, die gerade angezeigt werden, zum Teil die Wertebereiche überschritten haben. Die Achsen der Funktionsplots werden auf die Wertebereiche beschränkt und Plots von mit NaN kodierten Messwerten nicht dargestellt.

### 6.1.2 Aggregation der Daten

In Listing 24 ist ein Proof of Concept des Prozesses der Datenerneuerung zu sehen. Dieser umfasst die in Kapitel 3.3.2 festgelegten Anforderungen. Der Ablauf ist im Folgenden beschrieben.

```

1  -> # bash /root/argo_proto/renew_data.sh
2  (!) Starting renew process at Tue Mar  6 12:41:24 CET 2018.
3  >> Downloading new data...
4      done.
5  Datafolder: /root/aoml/
6  ||||| 6711/6711 [2:02:33<00:00, 1.10s/it]
7  >> Dumping tmpdb to /tmp/argo_db.sql ...
8      done. [164601410 Mar  6 14:44 /tmp/argo_db.sql]
9  >> Renew production database ...
10     done.
11  >> Renew Argos cache...
12     done.
13
14  Process finished.
15  This took 7381 seconds.
16  We had a downtime of 11 seconds.
17  So long and thanks for all the fish.
18
19  EOS

```

Listing 24: Erneuerung des Datensatzes

**Daten herunterladen** Die Daten des Argo-Programms werden heruntergeladen. Es handelt sich um die Livedaten nach der Qualitätskontrolle aus der Quelle aoml. Für das Herunterladen wird rsync verwendet. In der Quelle nicht mehr vorhandene Daten werden gelöscht.

**Daten aggregieren** Der Aggregationsteil der hier entwickelten Software wird verwendet, die soeben erneuerten Daten in die Datenbank zu überführen. Um Ausfallzeiten der Webapplikation möglichst gering zu halten, werden die Daten zunächst in eine temporäre Datenbank geschrieben. Über einen Ladebalken wird der derzeitige Stand dieses Prozesses sichtbar. Es ist außerdem ersichtlich, dass die Daten von 6711 Messstationen in 2 Stunden, 2 Minuten und 30 Sekunden in die Datenbank überführt wurden.

**Erstellen eines Datenbankdumps** Um die Daten in die Produktionsdatenbank zu überführen, werden die Tabellen der temporären Datenbank gedumpt. Es ist ersichtlich, dass der Dump eine Größe von ca. 160 Megabyte umfasst.

**Einspielen des Datenbankdumps** Über die Funktionalitäten des Postgresql-Clients psql werden die Daten in die Produktionsdatenbank überführt. Dieser Prozess erfordert eine Ausfallzeit der Webapplikation von 11 Sekunden.

**Erneuern des Caches zur Anzeige der Argo-Floats** Um die Daten in der Anzeige zu erneuern, wird der Cache neu geschrieben. Für diesen Zweck wird ein GET-Request an die hierfür festgelegte URL gesendet. Die Webapplikation erneuert daraufhin den Cache. Damit stehen der Webapplikation die erneuerten Daten zur Verfügung.

## 6.2 Usability-Umfrage

### 6.2.1 Testaufbau

Um die Usability der Webapplikation zu testen, wurde eine Umfrage durchgeführt. Als Grundlage diente der System Usability Scale (SUS). Für diese Umfrage waren insgesamt 10 Fragen innerhalb einer Skala von 1 (gar nicht) bis 5 (sehr) zu bewerten (vgl. [Rau11]).

An der Umfrage nahmen insgesamt 12 Personen teil. Diese setzen sich aus Bekannten- und Kollegenkreis zusammen. Die Daten wurden anonym erhoben, ein Rückschluss auf die Personen ist damit nicht mehr möglich.

In Abbildung 6.2.3 ist eine grafische Zusammenfassung der Ergebnisse zu sehen. In dieser findet sich zu jeder Frage ein Balkendiagramm der akkumulierten Häufigkeit, kombiniert mit einem Box-Whisker-Plot. Dieser setzt sich aus folgenden Elementen zusammen:

**Box** Dies markiert den Bereich in welchem sich 50% der Daten finden. Der Median wird durch eine vertikale Linie gezeigt. Die hier vorliegende Darstellung ist um den Durchschnitt (mean) erweitert. Die schmale Linie stellt das arithmetische Mittel und die fett gedruckte den Median dar.

**Whisker (Antennen)** Dieser Bereich stellt die äußeren Quantile dar. Hier wird die Grundeinstellung von maximal dem 1.5-fachen des Interquartilabstandes für den Whiskerbereich verwendet.

### 6.2.2 Testauswertung

Im Folgenden findet sich eine detaillierte Auswertung der hier erhobenen Daten:

**Statement 1: „Ich kann mir gut vorstellen, ArgoData regelmäßig zu nutzen.“** Diese Frage zielt auf die Wiederbesuchsbereitschaft ab. Die Antworten weisen eine breite Streuung auf. Die am häufigsten gegebene Antwort liegt bei 4 (4 von 12 Personen). Der Median liegt bei 3, das arithmetische Mittel befindet sich bei 2.67. Die Box umfasst die Antworten 2 bis 4. Der Whisker schließt Antwort 1 mit ein.

**Statement 2: „Ich empfinde ArgoData als unnötig komplex.“** In dieser Frage bildet sich die Reduktion der Komplexität ab. Die am häufigsten gegebene Antwort liegt bei 1 (7 von 12 Personen). Der Median liegt bei 1, das arithmetische Mittel bei 1.67. Die Box umfasst die Antworten 1 bis 2. Der Whisker schließt Antwort 3 mit ein.

**Statement 3:** „Ich empfinde ArgoData als einfach zu nutzen.“ Durch diese Frage wird die Einfachheit der Darstellung und die Reduktion der Benutzungskomplexität abgebildet. Die am häufigsten gegebene Antwort liegt bei 5 (6 von 12 Personen). Der Median liegt bei 4.5, das arithmetische Mittel bei 4.25. Die Box umfasst Antworten 4 und 5. Der Whisker schließt Antwort 3 mit ein.

**Statement 4:** „Ich denke, dass ich Hilfestellung bei der Benutzung brauchen würde um ArgoData zu benutzen.“ Diese Frage ermittelt ob genügend Hilfestellung zur Verfügung gestellt wurde. Die am häufigsten gegebene Antworten liegen bei 1 und 3 (jeweils 4 von 12 Personen). Der Median liegt bei 2.5, das arithmetische Mittel liegt bei 2.33. Die Box umfasst Antworten 1 bis 3, der Whisker schließt Antwort 4 mit ein.

**Statement 5:** „Ich finde, dass die Funktionen (Karte, Steuerung, Anzeige von Daten) gut integriert sind.“ Diese Frage ermittelt die Integration der Steuerungselemente. Die am häufigsten gegebene Antwort liegt bei 4 (6 von 12 Personen). Der Median liegt bei 4, das arithmetische Mittel bei 4.33. Die Box umfasst die Antworten 4 und 5. Der Whisker schließt Antwort 3 mit ein.

**Statement 6:** „Ich finde, dass es in ArgoData zu viele Inkonsistenzen gibt.“ Diese Frage zielt auf die Integration der Komponenten als Gesamtkonzept ab. Die am häufigsten gegebene Antwort liegt bei Antwort 2 (5 von 12 Personen). Der Median liegt bei 2, das arithmetische Mittel bei 2.08. Die Box umfasst Antworten 2 und 3. Der Whisker schließt Antwort 1 mit ein.

**Statement 7:** „Ich kann mir vorstellen, dass die meisten Leute ArgoData schnell zu beherrschen lernen.“ Diese Frage versucht über eine externe Perspektive die Lernkurve bei der Benutzung abzufragen. Die am häufigsten gegebene Antwort liegt bei 5 (6 von 12 Personen). Der Median liegt bei 4.5, das arithmetische Mittel bei 4.42. Die Box umfasst die Antworten 4 und 5. Der Whisker schließt Antwort 3 mit ein.

**Statement 8:** „Ich empfinde die Bedienung als sehr umständlich.“ Diese Frage versucht das Erleben des Bedienungskonzeptes zu erfragen. Die am häufigsten gegebene Antwort liegt bei 2 (7 von 12 Personen). Der Median liegt bei 2, das arithmetische Mittel bei 1.75. Die Box umfasst die Antworten 1 und 2. Der Whisker schließt Antwort 3 mit ein.

**Statement 9:** „Ich musste eine Menge Dinge lernen, bevor ich mit ArgoData arbeiten konnte.“ Diese Frage ermittelt die Lernkurve die zur Benutzung notwendig ist. Die am häufigsten gegebene Antwort ist die 1 (9 von 12 Personen). Der Median liegt bei Antwort 1, das arithmetische Mittel bei 1.5. Die Box umfasst Antwort 1, der Whisker schließt keine weiteren Antworten mit ein, diese sind damit als Ausreißer zu betrachten.

### 6.2.3 Interpretation der Ergebnisse

Die Antworten sind durchwegs als positiv zu betrachten. Die Fragen zur Komplexität lassen erkennen, dass es gelungen ist, die wissenschaftlichen Daten ohne Vorkenntnisse verstehbar darzustellen. Die Bedienungsmuster wurden angenommen und als konsistent und wenig umständlich beschrieben. Die Frage zur Wiederbesuchsbereitschaft weist eine starke Streuung auf. Da es sich um eine Seite mit einem Bildungsangebot handelt, ist dies erklärbar. Ein klarer Verbesserungswunsch lässt sich aus der Frage zu den angebotenen Hilfestellungen ableiten. Hier gilt es zu überlegen, ob und in welcher Form den benutzenden Personen zur Hand gegangen werden kann.



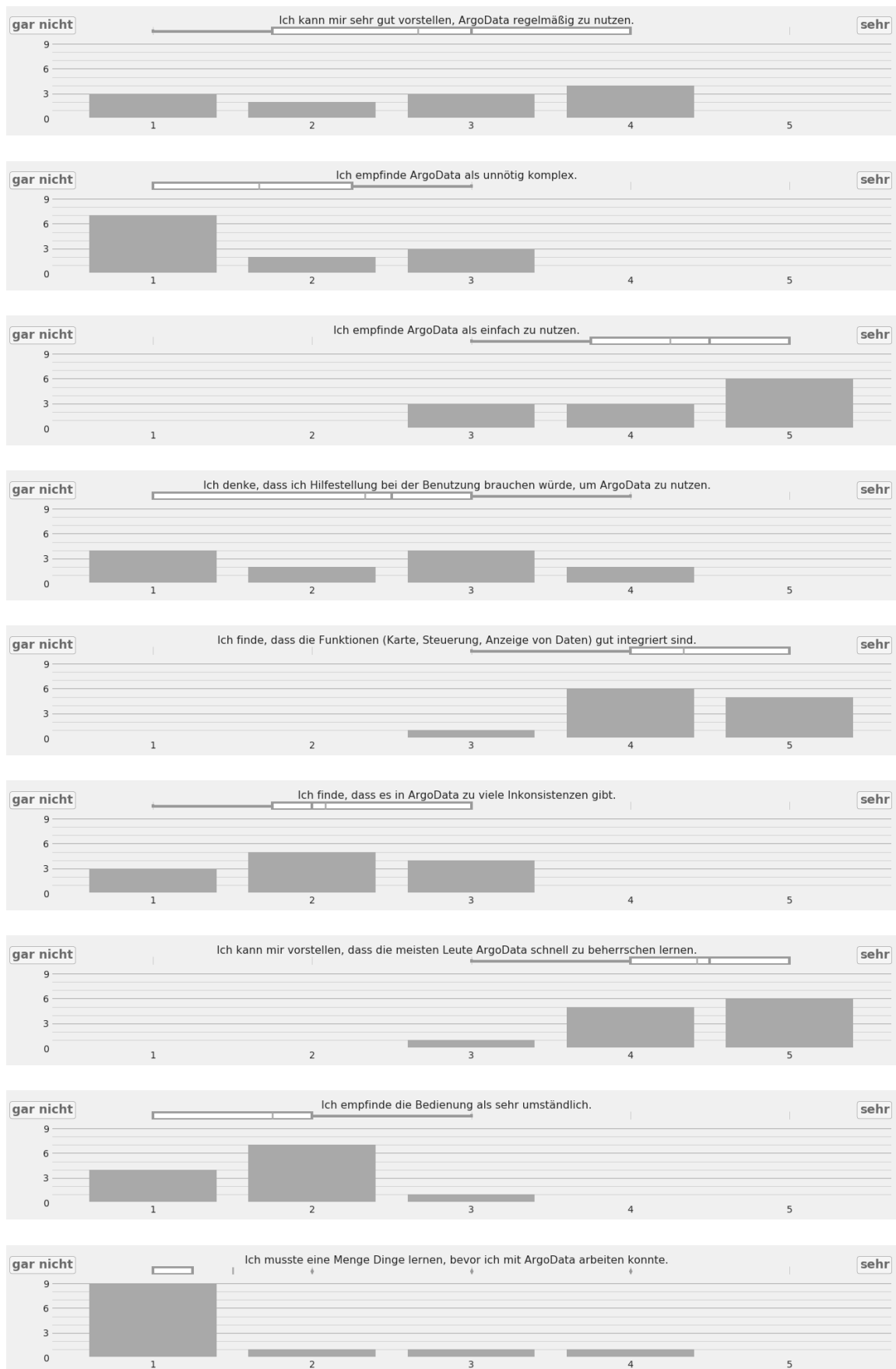


Abbildung 18: Ergebnis der Umfrage zur Usability

## 7 Demonstration und Auswertung

### 7.1 Geschaffene Lösung

Im Rahmen dieser Arbeit entstand eine Lösung zur Aggregation und Betrachtung der Daten des Argo-Programms. Unter dem Argo-Programm werden Messwerte über den Klimazustand unserer Weltmeere gemessen und öffentlich gemacht. Diese Daten wurden heruntergeladen, angepasst und in ein Format überführt, dass durch eine Webapplikation zur Anzeige gebracht werden konnte. Die Webdarstellung bediente sich moderner Frameworks wie Flask, SQLAlchemy und OpenLayers um die Daten über eine Weltkarte zur Darstellung zu bringen. Durch die Kartenrepräsentation der Messstation ist es den Anwendern möglich, mit den Datensätzen zu interagieren. Durch einen Mausklick werden die Messwerte über Funktionsplots angezeigt. Fehlerhafte Daten werden markiert, so dass es den Anwendenden möglich ist, diese zu erkennen. Durch eine Umfrage konnte ermittelt werden, dass die hier implementierten Schnittstellen gut integriert sind und die Bedienung als intuitiv und leicht zu erlernen empfunden wurde.

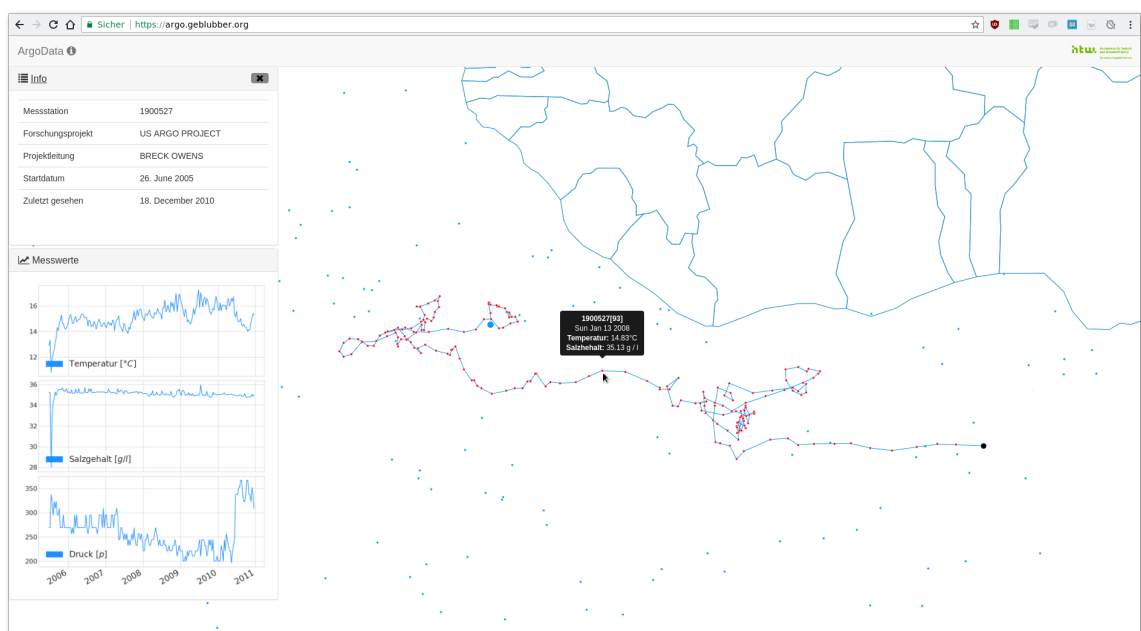


Abbildung 19: Die Webpräsenz von ArgoData

Bei der Aggregation wurde stark auf die Modularität der Anwendung geachtet. Objektorientierte Programmierung und die saubere Ausarbeitung von Schnittstellen wurde als zentrales Entwicklungsparadigma betrachtet. Auch in der Webanwendung wurde auf eine feinteilige Modularisierung geachtet. Die bei der Entwicklung mit Flask empfohlenen Entwurfsmuster wurden umgesetzt.

## 7.2 Erweiterungsmöglichkeiten

Der hier ausgearbeitete Prozess der Datenüberführung ist stark monolytisch und statisch ausgelegt. Die Erneuerung wird immer über den gesamten Datensatz ausgeführt. Dies ist aus zwei Gesichtspunkten als verbesserungswürdig anzusehen. Zum einen finden redundante Schreibzugriffe statt. Zum anderen wird durch den Schreibvorgang in die produktive Datenhaltung eine kurzzeitige Ausfallzeit des Webservices hervorgerufen. Ein Microservice, der die Daten auf Veränderungen überwacht und nur alternierende Daten erneuert, wäre deswegen eine sinnvolle Weiterentwicklung der Plattform.

Durch die Usability Umfrage wurde ermittelt, dass sich die Benutzenden durch den stark minimalistischen Designansatz verloren fühlten, bzw. eine Hilfestellung erwartet hätten. Hier könnte man durch ein Interface-Overlay oder der initialen Anzeige eines Hilfsmodals eine bessere Hilfestellung leisten.

Die Anforderung der Messdaten und das Erstellen des Funktionsplots erfordert nach jedem Klick einige Sekunden, bis die Daten zur Anzeige gebracht werden. Dadurch wirkt die Darstellung schwerfällig und träge. Diesem könnte man durch einen Caching-Prozess entgegenwirken. Ein weiterer spannender Ansatz wäre das proaktive Laden von Datensätzen durch Ermittlung der Mauszeigerposition noch bevor der Datensatz angeklickt wurde. Dies könnte durch maschinelles Lernen sogar noch weiter verfeinert werden. Besonders die Kombination von Caches und dem proaktiven Laden der Daten könnte hier starke Verbesserung in der Seitengeschwindigkeit bringen.

Es hat sich gezeigt, dass es schwierig ist, zwischen den Messpunkten auf der Karte und den Positionen der Funktionsplots eine geistige Verbindung herzustellen. Aus diesem Gesichtspunkt wäre es eine sinnvolle Erweiterung der Applikation, die Graphen über eine auf JavaScript aufbauende Lösung auszutauschen. Damit wäre es möglich, die Position des Mauszeigers in Karte und Graph zu ermitteln und die Position im Messverlauf im jeweils anderen Element hervorzuheben.

„Der echte Zug des Wissens ist nichts Statisches, das man anhalten und in Teile zerlegen kann. Er ist immer in Fahrt. Auf einem Gleis namens Qualität. Und die Lok und die 120 Güterwagen fahren nie woanders hin, als wo das Gleis der Qualität sie hinführt.“

**Robert M. Pirsing** – *Zen und die Kunst ein Motorrad zu warten*

## Quellenverzeichnis

- [Arg] Argo float data and metadata from global data assembly centre (argo gdac). <http://www.seanoe.org/data/00311/42182/>. (Aufgerufen am 12.12.2017).
- [Arg17a] Documentation - Argo Data Management. <http://www.argodatamgt.org/Documentation>, Dec 2017. (Aufgerufen am 12.12.2017).
- [Arg17b] Argo data selection - Argo Data Management. <http://www.argodatamgt.org/Access-to-data/Argo-data-selection>, Dec 2017. (Aufgerufen am 12.12.2017).
- [CKT<sup>+</sup>15] Thierry Carval, Robert Keeley, Yasushi Takatsuki, Takashi Yoshida, Claudia Schmid, Roger Goldsmith, Annie Wong, Ann Thresher, Anh Tran, Stephen Loch, and Rebecca Mccreadie. Argo user manual, 2015.
- [IBNW09] Christopher Ireland, David Bowers, Michael Newton, and Kevin Waugh. A classification of object-relational impedance mismatch. In *Advances in Databases, Knowledge, and Data Applications.*, pages 36–43. IEEE, 2009.
- [Moo98] Gordon E Moore. Cramming more components onto integrated circuits. *Proceedings of the IEEE*, 86(1):82–85, 1998.
- [net] Python cdf4. <http://unidata.github.io/netcdf4-python/>.
- [Ng14] Jackie Ng. bootstrap-viewer-template. <https://github.com/jumpinjackie/bootstrap-viewer-template>, 2014.
- [Org] Organizations - sqlalchemy. <https://www.sqlalchemy.org/organizations.html>. (Aufgerufen am 12.03.2018).
- [Rau11] Matthias Rauer. Quantitative usability-analysen mit der system usability scale (sus). <https://blog.seibert-media.net/blog/2011/04/11/usability-analysen-system-usability-scale-sus/>, April 2011. (Aufgerufen am 06.03.2018).
- [RGS<sup>+</sup>] Rew Russ, Davis Glenn, Emmerson Steve, Davies Harvey, Hartnett Ed, Heimbigner Dennis, and Ward Fisher. Netcdf: Introduction and overview. <https://www.unidata.ucar.edu/software/netcdf/docs/index.html>. (Aufgerufen am 13.12.2017).

- [Ron11] Armin Ronacher. Opening the flask. <http://mitsuhiko.pocoo.org/flask-pycon-2011.pdf>, 2011. (Aufgerufen am 13.01.2017).
- [SG] Megan Scanderbeg and John Gould. A beginners' guide to accessing argo data. [http://www.argo.ucsd.edu/Argo\\_data\\_guide.pdf](http://www.argo.ucsd.edu/Argo_data_guide.pdf). (Aufgerufen am 17.01.2017).
- [Sun16] Johan Sundstroem. world.geo.json. <https://github.com/johan/world.geo.json>, 2016.
- [WS11] B. Witte and P. Sparla. *Vermessungskunde und Grundlagen der Statistik für das Bauwesen*. Wichmann, 2011.

# Glossar

## API

Application-Programming-Interface. Programmschnittstelle, die Daten in maschinenlesbarer Form darstellt 20, 37, 41

## BUFR

Binäres Datenformat zur Übertragung meteorologischer Daten 5

## Controller

Element zur Programmsteuerung der Webanwendung 17, 34, 35, 40

## Framework

Erweiterbare Softwarebibliothek zur Erstellung von Systemen. Funktioniert nach dem Hollywood-Prinzip (don't call us, we call you) 16, 17, 36

## FTP

File Transfer Protocol. Protokoll zur Übermittlung von Daten 5

## HTML

Hypertext Markup Language. Beschreibungssprache die häufig für die Inhaltsbeschreibung von Webseiten verwendet wird 15, 21, 36, 40

## HTTP

Hypertext Transfer Protocol. Weit verbreitetes Protokoll um beispielsweise das WWW in den Webbrowser zu laden 5

## Julian Day

(Julianisches Datum) In der Wissenschaft gebräuchliches Datumsformat über die Tageszählung ab einem fixen Startdatum (meist 1. Januar 4713 v. Chr.) 10

## NaN

Not a Number. Ist ein numerischer Wert, der undefiniert ist, oder keinem Wert entspricht 41, 44

## TESAC

Wissenschaftliches Datenformat zur Übertragung von Sensordaten 5

## **WSGI**

Web Server Gateway Interface. Schnittstellenspezifikation zur Protokollstandardisierung der Kommunikation von Python-Programmen und Webservern 16

# **Abkürzungsverzeichnis**

## **CDF**

Common Data Format 9

## **DBMS**

Datenbank-Management-System 17, 31

## **DOI**

digital object identifier 5

## **GDAC**

Global Data Access Committee 3, 5

## **GOOS**

Global Ocean Observing System 4

## **JCOMM**

Joint Technical Commission for Oceanography and Marine Meteorology 3

## **KISS**

Keep It Simple, Stupid 17

## **netCDF**

Network Common Data Format 5, 9, 15, 24

## **ORM**

Objektrelationaler Mapper 19



# Anhang

## Resultate

### Daten aus der SUS Umfrage

User-Index	0	1	2	3	4	5	6	7	8	9	10	11
Statement 1	4	3	2	3	1	4	3	2	1	4	1	4
Statement 2	1	2	2	1	3	1	1	3	3	1	1	1
Statement 3	4	3	5	5	3	5	5	3	4	5	5	4
Statement 4	1	4	1	3	2	4	1	3	3	3	1	2
Statement 5	4	4	4	5	3	5	5	4	4	5	4	5
Statement 6	3	2	3	2	2	2	1	3	3	2	1	1
Statement 7	4	4	5	5	4	5	5	3	4	4	5	5
Statement 8	2	2	2	2	1	2	1	2	3	1	2	1
Statement 9	4	5	5	4	4	4	5	3	2	5	4	4
Statement 10	1	1	1	3	1	1	1	2	4	1	1	1

### Erzeugung der Umfragegraphen

```
1 import numpy as np
2 import seaborn as sns
3 import matplotlib.pyplot as plt
4 import matplotlib.patches as patches
5 from matplotlib.ticker import AutoMinorLocator
6 from matplotlib.font_manager import FontProperties
7
8 plt.style.use('fivethirtyeight')
9
10 def subplot(data):
11     bins=[1,2,3,4,5]
12     y = [data.value_counts()[x] if x in data.value_counts() else 0 for x
13     ↪ in bins]
14
15     f, (ax_box, ax_hist) = plt.subplots(2, sharex=True, gridspec_kw={"
16     ↪ height_ratios": (.1, .86)})
17     f.set_figwidth(20)
18     f.set_figheight(2.3)
19
20     whisker = sns.boxplot(data.values,
21     ↪ ax=ax_box, showmeans=True,
22     ↪ meanline=True, color="white",
23     ↪ meanprops=dict(color='darkgrey', linestyle='- '
24     ↪ , linewidth=2.5,))
25     ax_hist.bar(bins, y, color="darkgrey")
26     ax_box.set(yticks=[])
27
28     sns.despine(ax=ax_hist)
```

```

26 sns.despine(ax=ax_box, left=False)
27
28 f.suptitle(data.name, fontsize=16)
29
30 left, width = -.06, 1
31 bottom, height = -0.09, .33
32 right = left + width
33 top = bottom + height
34 p = patches.Rectangle(
35     (left, bottom), width, height,
36     fill=False, transform=ax_hist.transAxes, clip_on=False
37 )
38
39 font = FontProperties()
40 font.set_weight('bold')
41 font.set_size(18)
42 bbox = dict(facecolor='whitesmoke', edgecolor='grey', pad=0.2,
↳ boxstyle='round')
43 ax_hist.text(right, height, 'sehr',
44     horizontalalignment='center',
45     verticalalignment='top',
46     transform=ax.transAxes,
47     fontproperties=font,
48     color = 'dimgray',
49     backgroundcolor='whitesmoke',
50     bbox=bbox)
51 ax_hist.text(left, height, 'gar nicht',
52     horizontalalignment='center',
53     verticalalignment='top',
54     transform=ax.transAxes,
55     fontproperties=font,
56     color = 'dimgray',
57     backgroundcolor='whitesmoke',
58     bbox=bbox)
59
60 plt.yticks(np.arange(0, 9+1, 3.0))
61 ax_hist.set_ylim([0,9])
62 ax_hist.minorticks_on()
63 ax_hist.xaxis.grid(False)
64 ax_hist.yaxis.grid(b=False, which='minor', color='lightgray',
↳ linestyle='-')
65 ax_hist.yaxis.set_minor_locator(AutoMinorLocator(3))
66 ax_hist.yaxis.grid(b=True, which='major', color='darkgray',
↳ linestyle='-', markersize=1)
67 ax_box.xaxis.grid(True)
68
69 for col in df:
70     subplot(df[col])

```

## Erzeugen des Anforderungsgraphen

```

1 import networkx as nx
2 from networkx.drawing.nx_agraph import write_dot, graphviz_layout
3 import pandas as pd
4 import numpy as np
5 import matplotlib.pyplot as plt

```

```

6 from matplotlib.patches import Rectangle
7 import matplotlib.patches as mpatches
8 import matplotlib.patches
9 from matplotlib import pylab
10 URL_DOC = "https://docs.google.com/spreadsheets/d/e/2PACX-1
    ↳ vQfFMHK_XVtNXex5pAPh6boF4qdBH0iB6_ndGWV"\
11         "-FANyBXh1TB9MuIQ_Ex7gBlcrGOM02Tn133NxVf1/pub?gid=0&single=
    ↳ true&output=csv"
12
13 color_map = {
14     'kann': 'tomato',
15     'soll': 'orange',
16     'muss': 'green'
17 }
18
19 anforderungen = pd.read_csv(URL_DOC)
20
21 unpack = lambda cell: [c[1:-1] for c in cell.split("|")]
22
23 anforderungen_ = [row['Titel'] for i, row in anforderungen.iterrows()]
24 nachbedingungen_ = [unpack(row['Nachbedingungen']) for i, row in
    ↳ anforderungen.iterrows()]
25 vorbedingungen_ = [" " if pd.isnull(row['Vorbedingungen']) else unpack(
    ↳ row['Vorbedingungen']) for i, row in anforderungen.iterrows()]
26
27 flat_nachbedingungen = [item for sublist in nachbedingungen_ for item in
    ↳ sublist]
28 flat_vorbedingungen = [item for sublist in nachbedingungen_ for item in
    ↳ sublist]
29
30
31 G = nx.DiGraph()
32 G.add_nodes_from(anforderungen_)
33 G.add_nodes_from(flat_nachbedingungen)
34
35 A = G.subgraph(anforderungen_)
36 N = G.subgraph(flat_nachbedingungen)
37
38 for i, a in enumerate(anforderungen_):
39     for n in nachbedingungen_[i]:
40         G.add_edge(n,a, weight=1000)
41     for v in vorbedingungen_[i]:
42         G.add_edge(a,v, weight=0.0001)
43
44 G = G.reverse()
45 A = A.reverse()
46 n = N.reverse()
47
48 pos=graphviz_layout(G, prog='dot')
49
50 plt.figure(3,figsize=(36,40))
51 plt.axis('off')
52
53 color = [color_map[row['Prioritaet']] for i, row in anforderungen.
    ↳ iterrows()]
54
55 nx.draw_networkx_nodes(A, pos, node_shape='s', node_size=10000,
    ↳ node_color=color, alpha=0.6, label=True)
56 nx.draw_networkx_nodes(N, pos, node_shape='o', node_size=200, node_color
    ↳ ='grey')
57 nx.draw_networkx_edges(G, pos, node_shape='o')
58

```

```

59 x_centr = np.average([p[0] for p in pos.values()])
60
61 fontsize = 10
62
63 def shift_x(x):
64     align_map = {
65         -1: 'left',
66         1: 'right'
67     }
68     s = 1 if x < x_centr else -1
69     return (align_map[s], x - (s*7))
70
71 def shift_y(y):
72     return (y + 20)
73
74 def position_string(node):
75     x,y = pos[node]
76     align, x = shift_x(x) if False in N else ('center', x)
77     y = y-10 if node in N else y
78     y = shift_y(y) if node in A else y
79     return (align,x,y)
80
81 for n in G:
82     align,x,y = position_string(n)
83
84     plt.text(x,y,s=n, bbox=dict(facecolor='lightgray', alpha=0.5),
85             horizontalalignment=align,
86             color='black', fontsize=27)
87 plt.rcParams["legend.fontsize"] = 22
88 plt.legend(handles=[
89     mpatches.Patch(color=color_map['muss'], label='Prioritaet: muss'),
90     mpatches.Patch(color=color_map['kann'], label='Prioritaet: kann'),
91     mpatches.Patch(color=color_map['soll'], label='Prioritaet: soll'),
92     mpatches.Ellipse((1,1), 4, 0, fill=True, label='Nachbedingung',
93     ↪ color='grey')
94 ])
95 plt.show()

```