Hochschule für Technik
und Wirtschaft Berlin
University of Applied Sciences

# Konzeption und prototypische Entwicklung einer Webanwendung zur Aggregation und Analyse wissenschaftlicher Daten des Argo-Projektes

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 und half mir, diese Abschlussarbeit auszuarbeiten. Auch Herrn Prof. Dr.-Ing. Gärtner möchte ich dafür danken, dass er sich dazu bereit erklärt hat, diese Arbeit zu betreuen.
Außerdem möchte ich auch meinen Eltern danken. Auch wenn diese heute nicht mehr unter uns sind, so waren sie es doch, die den Keim gelegt haben, der mich dazu befähigte den Weg bis zu dieser Stelle zu gehen.
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.
Zuletzt und doch ganz besonders richtet sich mein Dank an Denise Peth. Diese begleitete mich auf dem Weg dieser Ausarbeitung und stand mir stets mir Rat und Tat zur Seite.

# Inhalt

# Abbildungsverzeichnis



# Quellcodeverzeichnis



# Tabellenverzeichnis

# 1 Einleitung

Die im Jahre 1965 von Gordon Moore vorhergesagte Gesätzmäßigkeit, die Komplexität, und damit die Speicherdichte, integrierter Schaltkreise, würden sich regelmäßig verdoppeln, hat sich 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 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?

Was hält uns davon ab, Bildung und Wohlstand aus diesem Pool zu generieren? Viele der Informationen stehen frei zur Verfügung und können genutzt werden und doch werden gerade in dieser Zeit die Prinzipien der wissenschaftlichen Evidenz von einem, gefühlt immer größeren Anteil unseres Kulturkreises, abgelehnt. In einer Zeit in der es einfacher den je ist, Fakten zu überprüfen, werden wissenschaftlich bewiesenen Aussagen wie dem Klimawandel einfach 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, über die Menschen an die wissenschaftliche Arbeit herangeführt werden können? Welche Daten sind geeignet, um die Brisanz und das Potential unserer Zeit einem breiteren Publikum zuzuführen.
Hier bietet das Argoprogramm eine Möglichkeit die hierfür benötigten Daten bereitzustellen. Unter dem Dach dieses Programms, werden seit anfang dieses Jahrtausends die 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. Diese dienen Wissenschaftlern um die Auswirkungen des globalen Klimawandels zu untersuchen.
Ziel dieser Arbeit ist eine einfache Darstellung aus diese Messwerten zu erarbeiten. Über ein exploratives Werkzeug sollen die die wissenschaftlichen Daten intuitiv erfahrbar sein. Dies geschieht mit der Hoffnung, hier eine Identifikation mit den Messwerten und des wissenschaftlichen Prozesses zu erreichen.
Diese Anwendung wäre zum Beispiel für Schulklassen geeignet. Die Schüler können durch eine Kontextvorgabe des Lehrenden die Auswirkungen des Klimawandels auf die Weltmeere für sich explorieren. Zusätzlich könnte beim ein oder anderen Interesse an der wissenschaftlichen Arbeit geweckt werden. Das Angebot richtet sich aber nicht explizit an heranwachsende. Auch erwachsene Personen können sich hier bei Interesse weiterbil-

den. Die Applikation kann hier außerhalb der wissenschaftlichen Arbeit die Relevanz der Forschung und die in diesem Programm erhobenen Messwerte erfahrbar machen.
Am Anfang dieser Arbeit wird das Argo Programm vorgestellt. es wird der Prozess der Datenerhebung und -veröffentlichung eingegangen. Im Anschluss werden die Anforderungen für die zu entwickelte Software ausgearbeitet und damit die geeignetsten Werkzeuge ermittelt. Daraufhin wird das System entworfen. Es wird eine Architekturbeschreibung durchgeführt und die wichtigsten Geschäftprozesse beschrieben. Hier werden Alternativen aufgezeigt und versucht, die geeignetste zu ermitteln. Die Implementierung wird im darauf folgenden Kapitel beschrieben. Hier werden bestimmte Prozesse iterativ verfeinert und verbessert um die am besten geeignetste Lösung zu finden. Um die Qualität der Software beschreiben zu können, folgt im darauf folgenden Kapitel eine Beschreibung der Verwendeten Test-Verfahren. Es werden die durchgeführten Unit-tests sowie eine Umfrage zur Bestimmung der Usability beschrieben. Abschließend wird die fertige Software demonstriert. Hierbei wird versucht einen möglichst kritischen Blick auf das Projekt zu werfen und Alternativen und Verbesserungen vorzustellen.

## 1.1 Existierende Lösungen

### 1.1.1 Das JCOMMops

Das Joint Technical Commission for Oceanography and Marine Meteorology (JCOMM) bietet mit `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. Über einen Twitter Account werden Änderungen an der Plattform und neu veröffentlichte Reports veröffentlicht.
Zwar bietet die Plattform durch ihre kartenbasierte explorative Darstellung ein ähnliches Angebot, wie es in ArgoData gemacht wird. Die schiere Fülle der Parameter erfordert vom Benutzer aber die Bereitschaft sich vertieft in die Matherie einzuarbeiten. Damit richtet sich das Angebot des Global Ocean Observing Systems an Wissenschaftler und Journalisten. Diesen dient sie als eine hervorragende Datengrundlage für deren Veröffentlichungen. Ein Benutzer aus der hier genannten Zielgruppendefinition wird von einem derartigen Angebot wohl eher abgeschreckt sein, bevor er dessen Vorteile für sich erarbeiten konnte.

### 1.1.2 Data Selection and Visualization Tool des Coriolis GDAC

Das Data selection and visualization tool der Coriolis GDAC [1] ermöglicht den Download und von nach verschiedenen Filterkriterien ausgewählten Datensätzen. Ein Betrachter erlaubt zusätzlich werte einer spezifischen Treibboje anzusehen und ihren bisher zurückgelegten Weg nachzuvollziehen.
Auch wenn die Darstellung hier mit weniger Parameter auskommt, so ist die primäre Aufgabenstellung dieser Plattform wohl das extrahieren der Daten um diese in einem wissenschaftlichen Paper verwenden zu können. Neben der akademischen Verwendung müsste ein Benutzer aber auch 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 mit diesem Ansatz.

[1] siehe [Arg17b]

# 2 Datengrundlage

## 2.1 Das 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 das Argo Programm gegründet. Dieses sollte, unterstützt durch das Satellitensystem Iason die Wassersäule der oberen 2000m auf deren chemischen Eigenschaften untersuchen. Dabei werden in ständigen Intervallen Salzgehalt, Druck, Temperatur und (...) gemessen. Die ermittelten Daten werden veröffentlicht, so das diese durch Wissenschaftler ausgewertet werden können. Zu diesem Zeitpunkt existieren bereits XXYYZZ Publikationen, die sich mit diesen Daten befassen.

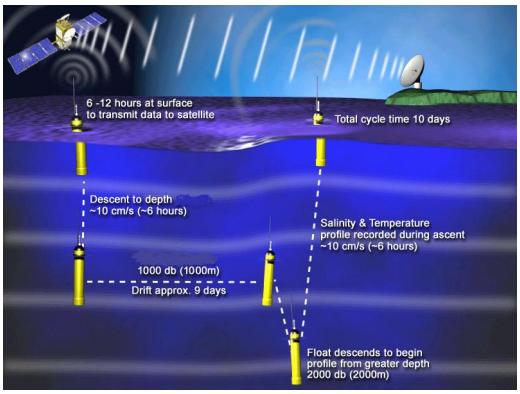



*Abbildung 1: Argos Messzyklus*

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 anschließend wieder zur Oberfläche aufzusteigen. An der Oberfläche sendet die Boje innerhalb von 6-12 Stunden die Daten über Iason an die Bodenstationen.
Nach der Erhebung werden die Daten auf deren Plausibilität und Qualität überprüft. [2] Nach diesem Prozess werden die aufbereiteten Daten über die Globalen Data Assembly Center in monatlichen Releasezyklen veröffentlicht. [3]. Der Werdegang der Daten ist in Abbildung 2 zu sehen.

[2] Siehe [SG] Seite 3
[3] Siehe [Arg]

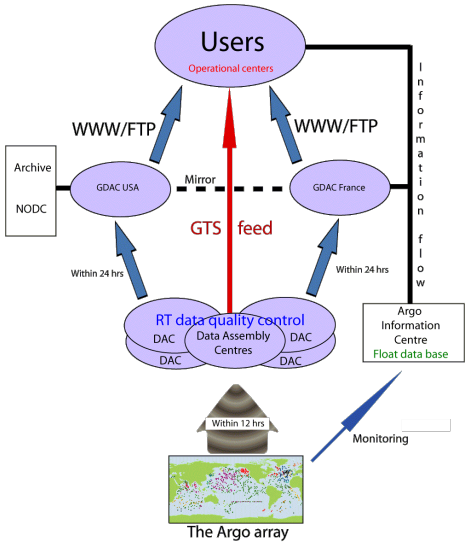



*Abbildung 2: Der Werdegang der Argo-Daten*

### 2.1.1 Daten des Argo Programmes

Alle am Argo 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.
Nachdem die Daten von Iason aus dem Argo Array an die Data Assembly Centers übermittelt hat, werden diese einer Qualitätskontrolle unterzogen. Die Daten werden auf Plausibilität und Abweichungen überprüft. Nach dieser Qualitätskontrolle werden die Daten nun über die Global Data Assembly Centers in Frankreich und den USA released. Diese können dann über HTTP und FTP abgerufen werden.
Ein mal Im Monat werden die daten als Snapshots, den sogenannten DOI (digital object identifier) released.

### 2.1.2 Datenformate

Während und vor der Qualitätskontrolle liegen die Daten im TESAC und BUFR Format vor.
Die von den DGACs veröffentlichten Daten stehen dann im 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.
[Arg17a]

### 2.1.3 Datenstruktur

In Listing 1 ist die Ordner Struktur 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 Oceanografic & meterolocical Laboratory erstellt. Hier befindet sich für jede Messboje ein Unterordner. In diesem Fall sind die Ordner der Bojen 1900200 und 1900201 zu sehen. Die Dateien meta, prof, Rtraj und tech sind eine Quelle für Metainformationen. (TODO Was steht in den metadaten?) 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.

```
./aoml/1900200/
- 1900200_meta.nc
- 1900200_prof.nc
- 1900200_Rtraj.nc
- 1900200_tech.nc
- profiles
    - D1900200_001.nc
    ...
    - D1900200_215.nc
    - D1900200_216.nc
./aoml/1900201/
...
```

*Listing 1: test*

# 3 Anforderungsanalyse

## 3.1 Systembeschreibung

Die zu entwickelnde Anwendung verfolgt einen datengetriebenen Ansatz. Die Daten müssen erhoben werden um diese dann über ein Webfrontend darstellbar zu machen. Die interessierten Benutzer können sich über die Web Präsenz die letzte Position der Bojen auf einer Karte ansehen. Durch einen Klick auf die Darstellung einer Messboje erhalten sie zudem Informationen und eine Darstellung der von dieser Messstation gemessenen Messwerte.

## 3.2 Ermittlung der Anforderungen

Ausgehend von einem ersten Prototypen und über User-Stories wurden einige Use-Cases entwickelt [4]. Hierbei wurden die zwei Klassen „Benutzende“ **(B)** und „Administrierende“ **(A)** mit den ihnen zugehörigen Anwendungsfällen herausgearbeitet.

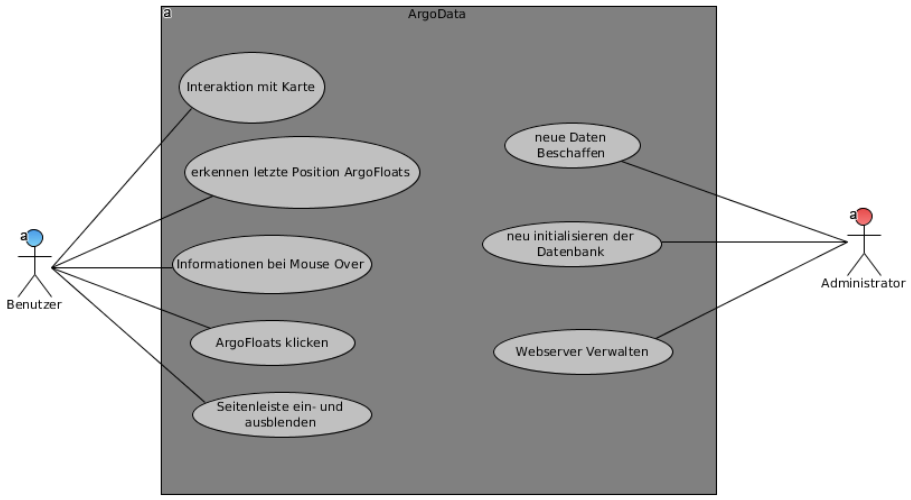



*Abbildung 3: Use case Diagramm der Anforderungen*

Anhand des so vorliegenden Modells wurde ein Abhängigkeitsgraph ausgearbeitet.[5]. Dieser beschreibt die Anforderungen und deren Abhängikeiten anhand eines gerichteten Graphen. Mit hilfe dieser Darstellung konnten die Anforderungen noch feiner ausgearbeitet und um Definitionslücken erweitert werden.

[4] Siehe Abbildung 3: Use case Diagramm der Anforderungen
[5] Siehe Abbildung 4: Anforderungen und ihre Abhängigkeiten als gerichteter Graph

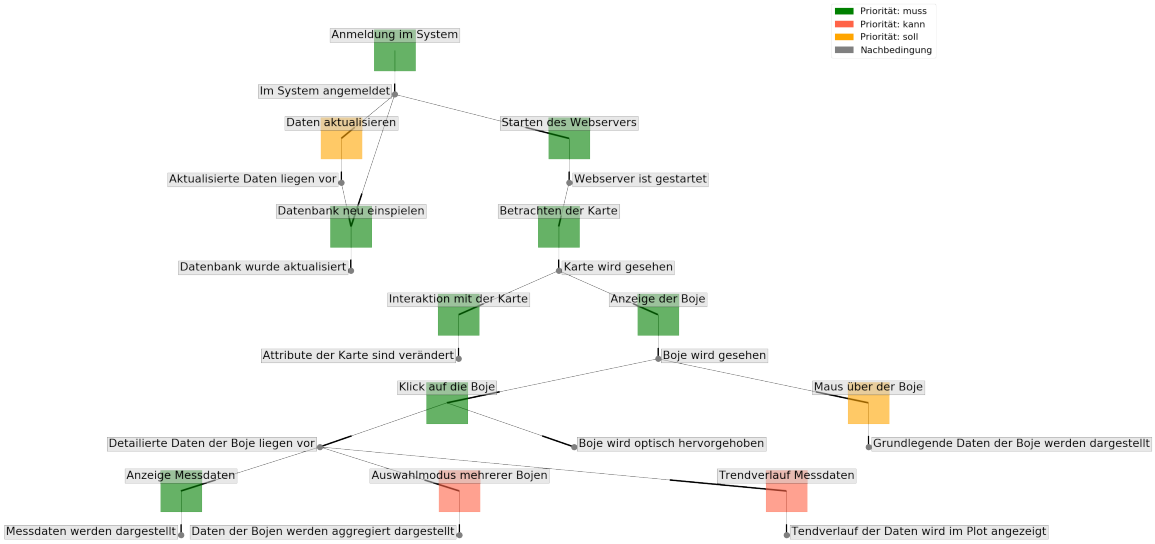



*Abbildung 4: Anforderungen und ihre Abhängigkeiten als gerichteter Graph*

## 3.3 Funktionale Anforderungen

### 3.3.1 Anwendende Perspektive

Die benutzenden erwarten die Darstellung einer Webapplikatin wenn diese die ihr zugehörige URL öffnen. Die Webseite soll über eine Kartenapplikation die Grenzen der Ozeane und der umliegenden Kontinente ersichtlich machen. Eine genauere Darstellung von Landmarkern wie Straßen, Städte oder Gebirge ist für den Aussagewert der Applikation nicht erforderlich. Die Kartendarstellung soll außerdem über Interaktionsmöglichkeiten, wie dem Einstellen der Zoomstufe, sowie des ausgewählten Kartenbereichs, verfügen.
Über den Kartendienst sind außerdem die letzte Position jeder Messboje aus dem Datensatz ersichtlich. Über Farbcodes wird der Status der Boje auf den ersten Blick erfahrbar gemacht. Um Streuung und Verdichtung an spezifischen Orten ersichtlich zu halten, werden alle Messstationen des angezeigten Kartenausschnittes dargestellt. Es erfolgt keine Zusammenfassung oder Ausblendung der Messstationen.
Wird die Maus über eine Boje geführt, so ist es sinnvoll hier bereits grundlegende Daten der Messstation darzustellen um eine Identifikation des jeweiligen Datensatzes zu ermöglichen. Ein Mausklick auf die schematische Darstellung der Boje soll spezifischere Daten der Boje anzeigen. Dazu soll in einem separierten Darstellungsfeld, der Verlauf der Messdaten, sowie einiger weiterer sinnvoller Werte der jeweiligen Messstation angezeigt werden. Es ist sinnvoll, wenn dieser Bereich je nach Wunsch des Benutzenden zu- und wieder abgestellt werden kann.

### 3.3.2 Administrative Perspektive

Um die Inhalte zu erstellen, werden auf Administrativer Sicht keine Journalistischen Tätigkeiten erwartet. Durch den Datengetriebenen Ansatz der Applikation müssen für das Erstellen der Inhalte keine Webmasken wie Texteingabefelder zur Verfügung gestellt werden. Vielmehr ist es hier sinnvoll, die hier benötigten Werkzeuge als Skripte zur Verfügung zu stellen, um die Administrativen Aufgaben über die Kommandozeile ausführen zu können.
Administrierende erwarten vom System dass sie neue Daten in die Datenbank überführen können. Hierbei ist es notwendig, dass die Datensätze des Argo Programms ausgelesen und in ein für die Datenbank aufbereitetes Format überführt werden.
Es wäre sinnvoll, die Aggregation der POIs ebenfalls über einen Programmablauf zu modellieren. In diesem Schritt müssten die Daten von der Webpräsenz des gooc heruntergeladen und an einem Ort entpackt werden. Zusätzlich müsste darauf geachtet werden, die Daten wieder zu entfernen, nachdem diese in die Datenbank des Systems überführt worden ist. Der Prozess der Beschaffung und Aktualisierung der Daten kann bis hin zur Vollautomatisierung abgebildet werden.

## 3.4 Nicht-Funktionale Anforderungen

### 3.4.1 Anwendende Perspektive

Für die Benutzenden ist die Usability der wichtigste Aspekt. Die Benutzungsmuster müssen klar ersichtlich und intuitiv erfahrbar sein. Die Darstellung der Applikation soll einfach und schlicht gehalten werden und dem gewohnten Erscheinen modernen Web Applikationen entsprechen.
Es soll darauf Wert gelegt werden, die Daten möglichst schnell nach der Interaktion zu Anzeige zu bringen, da das Warten auf Webapplikationen mit Frust verbunden ist, und eine hohe Absprungrate zur Folge hätte.

### 3.4.2 Administrative Perspektive

Für die Aggregation der Daten ist die Sicherheit ein zentraler Aspekt. Dies umfasst sowie Aspekte von Authentifizierung und Autorisierung der Rolle der Datenaggregation als auch eine Sicherheit um die Integrität der Heruntergeladenen Daten.

## 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 vorzuneh-

men. Hierbei soll darauf geachtet werden, nur diejenigen Daten zu verwenden, welche für die Erbringung des Dienstes notwendig sind.
Aus diesem Grund findet sich im Folgenden eine Auswahl aus dem Datenkatalog [6] von Argo mit der jeweiligen Begründung:

**Vereinfachung der Messdaten** 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 werden. Diese Sprachen gelten in der Entwicklung von Webseiten als Standard und werden von den gängigen Browsern unterstützt.
Die weiteren teile der Applikation soll mit einer Sprache entwickelt werden, die alle benötigten Teilbereiche umsetzen kann.
Das **Öffnen der netCDF** Dateien und die numerische Berechnung kann mit C, Java/Scala, R und Python erfolgen. Insbesondere die beiden letzten sind in der Datenverarbeitung und Numerik als etablierte Werkzeuge zu sehen.
Die Darstellung der Seite soll durch ein Webframework 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 Webframeworks 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 Auswahl von Programmbibliotheken, sowie der numerischen Berechnung besitzt die Sprache aber Vorteile und wird deswegen hier für die Implementierung genutzt.

[6] Siehe Quelle [CKT+15] S. 19 ff.

| Datenfeld | Beschreibung | Wird verwendet weil |
|---|---|---|
| PLATFORM_NUMBER | Eindeutige Identifikationsnummer einer einer Messstation | Wird benötigt um Messprofile eindeutig der Plattform zuordnen zu können. |
| CYCLE_NUMBER | Fortlaufende und innerhalb einer Messboje eindeutige Identifikationsnummer eines Messprofils | Dieser Wert wird benötigt, um Messungen eindeutig zuordnen zu können. |
| JULD | In Julian Date codierter Zeitpunkt der Übertragung (Begin oder ende?) einer Messung | Der Zeitpunkt wird benötigt um die Messungen in einen Zeitlichen Kontext zu stellen. |
| N_PARAM | Anzahl der Messsensoren. | Durch diesen Wert lässt sich die Anzahl der Sensoren ableiten. |
| LATITUDE & LONGITUDE | Die Positionsdaten einer Messung zum Zeitpunkt der Übetragung der Werte. | Dies ist ein essentieller Parameter um die Messungen in einen lokalen Zusammenhang zu stellen. |
| PRES | Messvektor des Wasserdrucks | Durch diesen Messwert lässt sich die Dichte der überliegenden Wassersäule herleiten. |
| TEMP | Messvektor der Wassertemperaturen | Die Temperatur des umliegenden Wassers ist einer der zentralen Messwerte. Durch diesen Messwert und seinen Trend lässt sich der Globale Klimawandel anschaulich darstellen. |
| PSAL | Messvektoren des Salzgehaltes | Ein weiterer Parameter, um die Auswirkungen des Klimawandels zu erfahren. Durch das Schmelzen der großen Eissschelfs an den Polen des Planeten ist eine Verringerung des Salzgehaltes der meere zu erwarten. |

*Tabelle 1: Auswahl der Daten aus dem ArgoProgramm*

### 3.6.2 Server

Für die Laufzeitumgebung der Applikation wird ein GNU/Linux eingesetzt werden. Die Software wird so entwickelt das sie unter den gängigen Distributionen lauffähig sein wird. Hier wird sich aber für Debian stable als Betriebssystem udn Laufzeitumgebung entschieden.

### 3.6.3 Webframework

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 entwicklet, die News-Seite des *Lawrence Journal-World* umzusetzen und zu verwalten. Django ist ein etabliertes und häufig verwendetes Webframework. 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 darstellbar zu machen. Das Farmework folgt dem 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 Microfarmework zu sehen. Es wurde insbesondere in Hinblick auf Geschwindigkeit entwickelt. Das Framework erlaubt 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 verfogen beide den Ansatz eines Microframeworks. 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 als das geeignetste Werkzeug für diese Aufgabe.

### 3.6.4 Datenbank

Das 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. Als Object-Realational-Mapper steht für Python SQLAlchemy zur verfügung.

# 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.

$$\text{Boje} \rightarrow (\text{misst}_{\text{zu Zeitpunkt}}) \rightarrow \text{Messprofile} \rightarrow (\text{enhält}) \rightarrow \text{Aufzeichnungen} \quad (1)$$

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 5 ist eine mögliche Modellierung des Prozesses zu sehen.

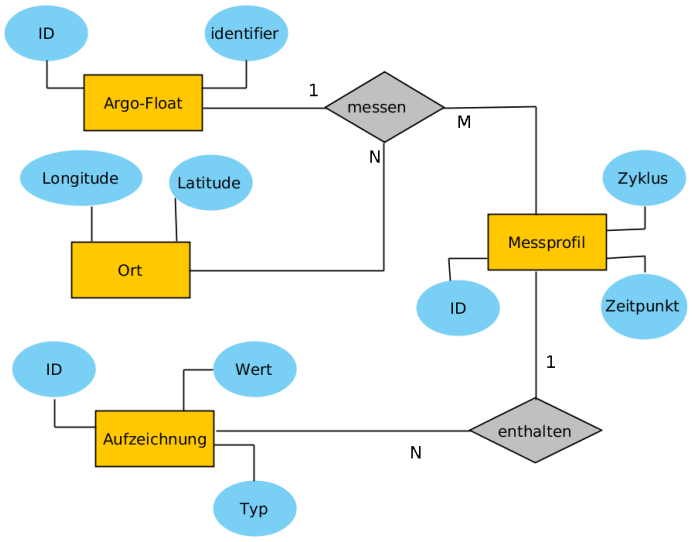



*Abbildung 5: Beschreibung der Entitäten der Datenaggregation*

## 4.2 Architektur

Die Applikation besteht aus zwei Grundkomponenten. Ein Teil ist für die Beschaffung und Aufbereitung der Daten zuständig, während der zweite Teil 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

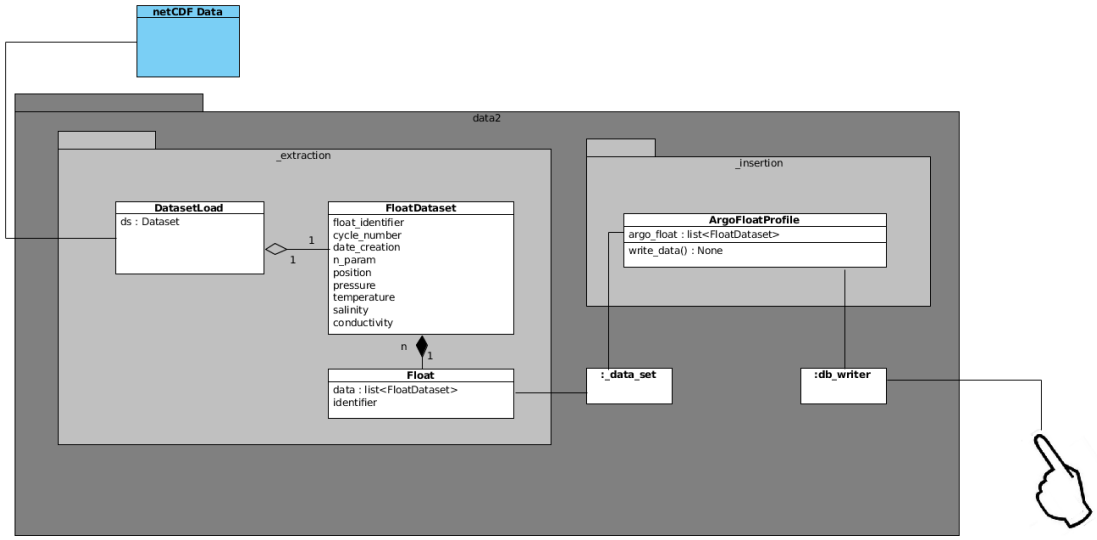



*Abbildung 6: Entwurf der Architektur der Aggregation der Daten*

logisches Format überführt werden. Des weiteren müssen diese Daten in die datenbank der Applikation überführt werden.
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 werden objektorientierte Strukturen für den ORM verwendet. Diese Strukturen teilt sich das Modul mit der Webapplikation. Die Behandlung der Daten sowie eine Steuerung des Prozesses sind als weitere Schnittstellen zu definieren.
Die Aggregation der Daten wird 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. Hier sollten Schnittstellen geschaffen werden, 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. Durch diese wird HTMl Und Javascript ausgeliefert, welches im Webbrowser der Benutzenden angezeigt werden kann.
Die zweite Teilkomponente ist dafür zuständig, die für die Anzeige benötigten Daten bereitzustellen. Die hier bereitgestellte API ist Zustandslos und dient als Schnittstelle zur Auswahl und Anzeige von Datensätzen aus dem Backend.

### 4.2.3 Ausarbeitung der Webrouten

```
1 GET     /
2
3 GET     /argo_float/[identifier]
4 GET     /argo_float/[identifier]/[data]
5 GET     /last_seen
6 GET     /last_seen/[force_reload]
```

Die darstellende Schicht verfügt nur über eine einzige Webseite. Über das Root-Element der Webapplikation wird die Karte mit den Messstationen ausgeliefert. Interaktionen und Veränderte Daten werden nach dem *singlepage-Prinzip* über diese Seite verfügbar gemacht.
Die API der Applikation soll zwei Funktionen erfüllen können. Für die Positionsbestimmung auf der Karte wird eine Geojson benötigt. Unter der Route `/last_seen` finden sich die letzten Positionen der Mesststationen zusammen mit den Daten die beim Mouseover angezeigt werden sollen. Das hier zurückgegebene Geojson wird über einen Cache vorgehalten. Um diesen Cache neu zu generieren ist es möglich der Route die Variable

`[force_reload]` zu übergeben. Um versehendliche oder böswillige Verwendungen dieser Route zu verhindern, handelt es sich bei der Variable um ein Geheimnis. Nur wenn dieses richtig angegeben wird, werden die Daten neu erstellt.
Über die Route `argo_float/[identifier]` wird ein Datensatz der über `identifier` ausgewählten Messstation zurückgegeben. Um nur einen spezifischen Messdatensatz zu erhalten, kann dieser über die Variable `data` ausgewählt werden.

Um die beiden Teilbereiche logisch zu trennen und modular vorhalten zu können, werden diese über blueprints abgebildet.

### 4.2.4 Aussehen der Webapplikation

Das Aussehen der Applikation soll dem Muster von gängigen Webapplikationen entsprechen. In Abbildung 7 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 ind ie Seite eingefügt. Dieses dient zur Darstellung der Messwerte des jeweiligen ArgoFloats.

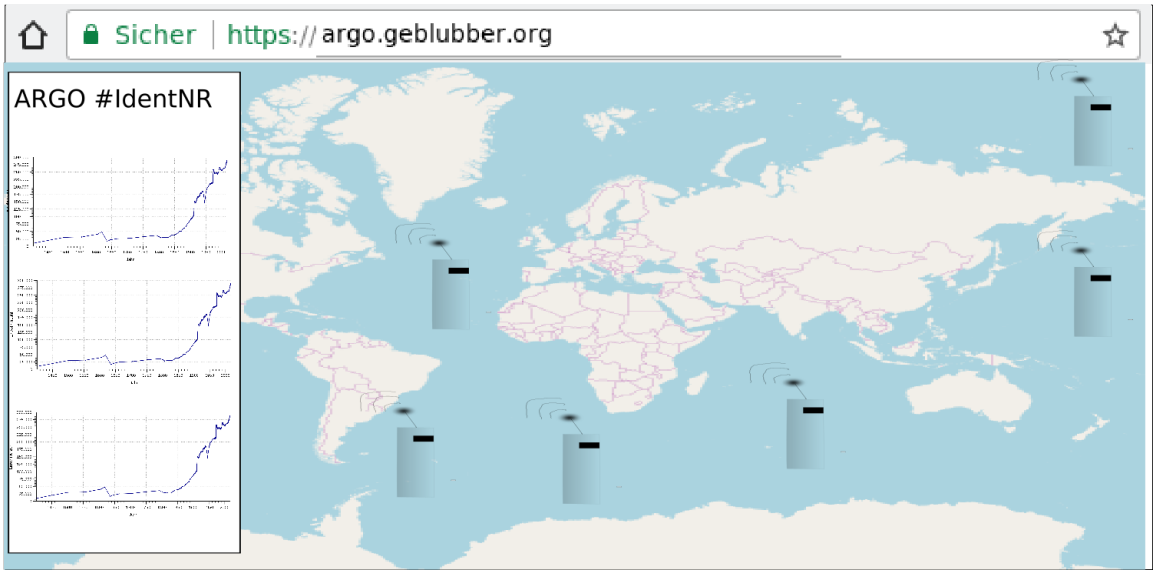



*Abbildung 7: Grafischer Grobentwurf der Webapplikation*

# 5 Grundlagen

## 5.1 Object Ralational Mapper

## 5.2 Verwendete Software

### 5.2.1 Network Common Data Format

Das Network Common Data Format (NetCDF) dient zum Austausch von wissenschaftlichen Daten. Es ist eine Weiterentwicklung des von der NASA entwickelten Common Data Format (CDF). Das Format zeichnet sich dadurch aus, dass es selbstbeschreibend ist. Dadurch wird die Dokumentation mit den Daten mitgeführt. Dies soll die Portabilität des Datensatzes verbessern. [7]
Neben einigen anderen Sprachen, wurde auch eine Bibliothek für Python entwickelt [8] 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 Berechnung des Datensatzerstellungsdatum 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
   ↪ REFERENCE_DATE_TIME
10 #     units: days since 1950-01-01 00:00:00 UTC
11 #     conventions: Relative julian days with decimal part (as parts of
   ↪ day)
12 #     _FillValue: 999999.0
13 # unlimited dimensions:
14 # current shape = (1,)
15 # filling off
16
17 julian_date = dataset.variables['JULD'][::][0]
18 dataset.close()
19
20 print(julian_date)
21 # 20062.5483218
22
23 juld_zero = datetime.datetime.strptime( '1950-01-01 00:00:00 UTC',
24                                         '%Y-%m-%d %H:%M:%S UTC')
25 date_creation = juld_zero + datetime.timedelta(days=int(julian_date))
26 print(date_creation)
27 # datetime.datetime(2004, 12, 5, 0, 0)
```

*Listing 2: Extraktion und Berechnung des Erstellungsdatums eines NetCDF-Datensatzes*

[7] vgl. [AGS+]
[8] Siehe [net]

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 Instanzierung.
Über das Attribut `dataset.variables` werden die Datensätze als `OrderedDict` gehalten. Bei der Extraktion eines Parameters erhält man zunächst die Dokumentation des jeweiligenm Datensatzes. In diesem Fall handelt es sich um den Zeitpunkt der Satteliten-übertragung 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. Beim Datumsformat handelt es sich um *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 extrahiert werden.
Zur Überführung in das Format des durch die ISO 8601 bei uns normierten 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 die Entferndung der Dezimalstellen vereinfacht werden. Die Casting-Operation `int()` rundet in jedem Fall ab.

### 5.2.2 numpy

Numpy ist eine Pythonbibliothek zur diskreten Verarbeitung von ein- oder mehrdimensionalen Arrays.
Nach der Extraktion der CDF-Daten, liegen diese als maskierte Arrays vor. Diese erlauben auch das fehlen einzelner Datensätze.

### 5.2.3 Julian Date

### 5.2.4 Flask

Flask wurde im Jahre (YXXX) von Armin Ronacher gegründet.
Die Idee startete Ursprünglich als Aprilscherz. [9] Der Designansatz wurde aber von vielen Menschen als positiv aufgefasst. Dieser wurde daraufhin unter dem Namen Flask released.

```
1 from flask import Flask
2 app = Flask(__name__)
3
4 @app.route('/')
5 def hello_world():
6     return 'Hello, ArgoData!'
```

*Listing 3: Minimalbeispiel für die Verwendung von Flask*

[9] [Ron11]

Das Framework verfolgt den Ansatz „*Why things work the way they work*“. Funktionalitäten werden also nicht versteckt, sondern sind dem Entwickler einfach zugänglich. Durch diese Designentscheidung ist es möglich ganze Webapplikationen mit nur wenigen Zeilen Code und in einer einzigen Sourcecode-Datei zu realisieren. Diese Transparenz wird durch ein global verfügbares „magisches“ !sic Objekt ermöglicht. Es lassen sich auch weitere Instanzen innerhalb einer Applikation erstellen. Teile der Applikation können über Blueprints modularisiert werden.
Durch die Templateengine **Jinja** erlaubt Flask das generalisieren von HTML-Dokumenten. Durch diese ist es möglich erweiterbare Grundtemplates anzulegen, Variablen abzufragen, zu überprüfen und über Kollektionen zu iterieren.
Durch sogenannte **Blueprints** ist es möglich, eine logische Trennung zwischen Teilbereichen der Applikation vorzunehmen. Diese Modularisierung erlaubt die Definierung von spezifischen Datenordnern sowie für HTML-Templates als auch spezifischer subrouten der Module innerhalb der Applikation.
(...)
In der Grundausstatung der Funktionsumfang von Flask sehr beschränkt. Der KISS-Ansatz bedingt, dass benötigte Funktionalitäten nachgerüstet werden müssen. Im Folgenden wird auf einige in dieser Applikation sinnvolle Erweiterungen eingegangen:

**flask_sqlalchemy** erlaubt die transparente Anbindung des ORM in die Applikation. In der Grundausstatung besitzt Flask keine Funktionalität um Datenbanken aus der Webapplikation anzusprechen. Durch die Einbindung dieser Erweiterung erhält man ein Framework, das dem Entwurfmuster MVC entspricht.

Entspürechend den in Flask gängigen Entwurfsmustern entspricht, wird die Anbdindung an den ORM durch ein globales Objekt modelliert. Dieses ist eine Instanz der Klasse `flask_sqlalchemy.SQLAlchemy` und stellt die Operationen von SQLAlchemy zur verfügung.

**flask_script** rüstet den Funktionsumfang der Webapplikation um eine zentrale Steurungseinheit auf. Diese Erweiterung besitzt bereits vorgefertigte Steuerungsmodule. Darunter finden sich scripte zum starten der Webapplikation oder zum starten eines Python-REPL mit den Umgebungsvariablen der Webapplikation. Unter Verwendung des Decorator Entwurfsmuster lässt sich das definierte Steuerungsmodul um beliebige Funktionalitäten erweitern.

### 5.2.5 SQLAlchemy

Ein **Object Ralational Mapper** oder kurz ORM ist die Schnittstelle zwischen in objektorientiert modellierten Programmteilen und realtionalen Datenbanken.

Zwischen diesen beiden Paradigmen existieren strukturelle Unterschiede. So besitzen Objekte im Unterschied zu Relationen eine Identität und unterstützen Vererbung. In der relationalen Algebra ist eine solche Komplexität nicht vorgesehen. Relationen bestehen aus Tupeln, identifizierbar durch den Primärschlüssel mit Beziehungen durch Fremdschlüssel. Durch ein Objekt-relationales Mapping wird versucht, diesen **impedance Mismatch** möglichst transparent zu überwinden.

Ein bei der Entwicklung von Programmen mit Python weit verbreiteter ORM ist **SQLAlchemy**. So wird dieser zum Beispiel im Backend von reddit oder bei Mozilla eingesetzt. SQLAlchemy wird für die Verwendung in Flask empfohlen.[10]

Im folgenden wird die Verwendung von SQLAlchemy exemplarisch an folgender Beziehung aufgezeigt

$$\text{Messprofil} \rightarrow (\text{enthält}) \rightarrow \text{Aufzeichnungen}$$

In Listing 4 ist zu sehen, welche Methoden benötigt werden um SQLAlchemy zu verwenden.
(...) Aufzählung????
Danach wird ein Basisobjekt erstellt. Dieses wird verwendet um Datenfelder im Kontext des ORM registrieren zu können.

```
1 from sqlalchemy import *
2 from sqlalchemy.ext.declarative import declarative_base
3 from sqlalchemy.orm import relation, sessionmaker
4 from datetime import datetime, timedelta
5
6 Base = declarative_base()
```

*Listing 4: SQLAlchemy 1*

Die Registrierung von Datenklassen um ORM erfolgt durch Vererbung. Wie in Listing 5 zu sehen ist, erbt die Klasse 'Profile' von der Basisklasse 'Base'.
Als Attribute besitzt die Klasse id, also den eindeutigen Primärschlüssel für die spätere Zuordnung des Datensatzes. Sowie die Variable timestamp, welche die Zuordnung einer Messung zu einem spezifischen Zeitpunkt erlaubt. Letzterer wird über den initializer der Klasse vorgegeben, muss also bei der Instanzierung des Objektes angegeben werden.

```
1
2 class Profile(Base):
3     __tablename__ = "profiles"
4
5     id = Column(Integer, primary_key=True)
```

[10] Siehe [Ron11] S. 33

```
6       timestamp = Column(Date)
7
8       def __init__(self, timestamp):
9           self.timestamp = timestamp
10
11      def __repr__(self):
12          return f'<Profile {self.id!r}>'
```

*Listing 5: SQLAlchemy 1a*

In Listing 6 ist der Aufbau der Klasse 'Record' zu sehen. Dieser ist etwas komplexer und besitzt neben den reinen Datenfeldern noch die Attribute 'profile_id' und 'profile'. Diese erlauben es, die Beziehung der beiden Datenfelder zu beschreiben. Die für die Datenfelder verwendeten Typen werden von SQLAlchemy zur Verfügung gestellt.
Über das erste Attribut wird der Fremdschlüssel des Messprofils angegeben, das die spezifische Aufzeichnung enthält. Die Auswahl erfolgt über eine Objekt der Klasse 'RoreignKey()'. Die Art der beziehung wird in der darauf folgenden Zeile definiert. Über die Methode 'relation()' wird im Klassenattribut 'profile' eine beziehung definiert. Bei der Übergabe einer Instanz der Klasse 'Profile' wird dieses Objekt in diese Beziehung eingebunden. Wie diese Einbindung von statten gehen soll, wird über den Parameter 'lazy' definiert. Der Parameter 'backref' erlaubt es, die Beziehung bidirektional zu implementieren. Der hier vergebene Name kann bei der Erstellung eines Profiles verwendet werden um Records zuzuordnen.

```
1
2   class Record(Base):
3       __tablename__ = 'records'
4
5       id = Column(Integer, primary_key=True)
6       data_type = Column(String(30))
7       value = Column(Float)
8
9       profile_id = Column(Integer, ForeignKey('profiles.id'))
10      profile = relation('Profile', backref='records', lazy=False)
11
12      def __init__(self, data_type, value, profile=None):
13          self.value = float(value)
14          self.data_type = data_type
15
16          self.profile = profile
17
18      def __repr__(self):
19          return f'<Record {self.id!r}>'
```

*Listing 6: SQLAlchemy 1b*

Durch Attribute werden die jeweiligen Werte und die IDs der Einträge definiert. Diese werden innerhalb der Klasse, wenn nötig, durch das Schlüsselwort `self` verfügbar gemacht. Zwischen den Datenfeldern besteht eine $1 \times N$ Beziehung. Diese wird innerhalb der Klasse

Record in den Zeilen 16 und 17 definiert. In der Variable profile_id wird der Fremdschlüssel gespeichert. Der Fremdschlüssel ist notwendig, um die Referenzierung zwischen den verschiedenen Tupeln in den Tabellen vornehmen zu können. Über die Methode relation() wird die Art der Beziehung zwischen den Datenfeldern definiert. Da die Beziehung in diesem Fall bidirektional sein soll, wird hier über den Parameter 'backref' ein Name definiert, über den die Einträge auf der Seite der Profile angesprochen werden können.
SQLAlchemy verwaltet die Verbindung zum DBMS über ein sogenanntes engine-Objekt. Dieses wird in der ersten Zeile in Listing 7 erstellt. In diesem Fall, wird eine SQLite-Datenbank verwendet.
Die Datenklassen aus Listing 4 werden über deren Basisklasse in dieser engine registriert. Dies ermöglicht das Mapping zwischen den Objekten und der relationalen Datenbank.
Die Operationen zur Persestierung, sind in der Klasse Session zusammengefasst. Diese wird erzeugt und instantiiert.
In den Zeilen 7 - 13 in Listing 7 werden nun spezifische Datenfelder Definiert. Es ist zu sehen, dass die Beziehung zwischen den Datenfeldern eine Implementierung in beide Richtungen zulässt.
Zuerst wird ein Record r1 mit einem anonymen Profile definiert. Diese Aufzeichnung trägt ein Datenfeld Temperature mit einem spezifischen Wert.
Anschließend wird in p1 eine Instanz von Profile gespeichert. Dieses trägt zwei Aufzeichnungen die zum selben Zeitpunkt erhoben worden sind.

Im nächsten Schritt werden über add und commit die Datenfelder zur session hinzugefügt und danach auf die Datenbank übertragen.
Wenn ein Fehler bei dieser Operation erfolgt, werden die Änderungen über die Methode rollback() wieder rückgängig gemacht.

```
1  engine = create_engine('sqlite:///argo.db')
2  Base.metadata.create_all(engine)
3
4  Session = sessionmaker(bind=engine)
5  session = Session()
6
7  r1 = Record(data_type="Temperature", value=15, profile=Profile(datetime.
       ↪ now()))
8
9  p1 = Profile(timestamp=datetime.now() - timedelta(days=3))
10 p1.records = [Record(data_type="Temperature", value=25),
11                Record(data_type="Pressure", value=55)]
12
13 try:
14     session.add(r1)
15     session.add(p1)
16     session.commit()
```

```
except Exception as err:
    print(err)
    session.rollback()
```

*Listing 7: SQLAlchemy 1*

SQLAlchemy stellt zudem Methoden für das Formulieren von Queries zur Verfügung. In Listing 8 werden exemplarisch einige der zuvor gespeicherten Werte aus der Datenbank gelesen und angezeigt.
Die Abfragen werden intern in SQL Queries übersetzt. Bei Bedarf ist es auch möglich, Abfragen über SQL Queries zu stellen.

```
for p in session.query(Profile).all():
    for r in p.records:
        print(f"Zu Zeitpunkt {p.timestamp} wurde {r.data_type} mit {r.
    ↪ value} gemessen.")

# Zu Zeitpunkt 2018-01-20 wurde Temperature mit 15.0 gemessen.
# Zu Zeitpunkt 2018-01-17 wurde Temperature mit 25.0 gemessen.
# Zu Zeitpunkt 2018-01-17 wurde Pressure mit 55.0 gemessen.
```

*Listing 8: SQLAlchemy 1*

Für die Verwendung in Flask gibt es eine Erweiterung für die Verwendung von SQLAlchemy. Durch diese werden einige der oben genannten Schritte vereinfacht. Zum beispiel werden `Session`, `engine` und `Base` über ein zentrales Objekt `db` abgebildet. Prinzipiell erfordert aber auch die Verwendung von flask-sqlalchemy die selben Schritte.

### 5.2.6 JSON?

### 5.2.7 openLayers

OpenLayers ist ein Framework zur Entwicklung webbasierter Geoapplikationen. Dieses ist in JavaScript entwickelt und erlaubt durch eine layerstruktur den Aufbau komplexer Kartenapplikationen. Dabei lassen sich die Elemente der Karten durch folgende Objekte strukturieren:

**`ol.Map`**

**`ol.View`**

**`ol.Overlay`**

**`ol.layer.Base`**

**`ol.interaction.Interaction`**

**`ol.control.Control`**

OpenLayers ist eine Programmbibliotek 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 verschiedensten Quellen zu Rendern. Dabei können Kachel- oder auch Vektorbasierte Materialien eingebunden werden.

**Kachel-Layer** liefern Bilddaten direkt aus. Dabei werden die Kartenmatherialien in Kacheln aufgeteilt.Dadurch das die Bilder in Teilbereiche untergliedert sind, müssen nur die Bildbereiche ausgeliefert werden, die für die Darstellung des Kartenausschnittes vonnöten sind.

**Vektor-Layer** sind Beschreibungen von Linien und Polygonen. Die Darstellung wird nicht anhand von Bildmaterial generiert, sondern anhand von geometrischen Beschreibungen gezeichnet.

Neben der reinen Darstellung erlaubt die Bibliothek auch die Definition von „`controls`“. Dabei lassen sich Zoom-, Rotations- und Mouse-Positions-Effekte umsetzen. Siehe Quelle [GSH15] S. 296
Zu zeichnende Daten lassen sich über GEOJson-Objekte modellieren. Die Bibliothek folgt einem objektorientierten Entwurfsmuster und erlaubt das transparente Hinzufügen und Ändern von Layern der Karte.

### 5.2.8 D3

## 5.3 ...

# 6 Implementierung

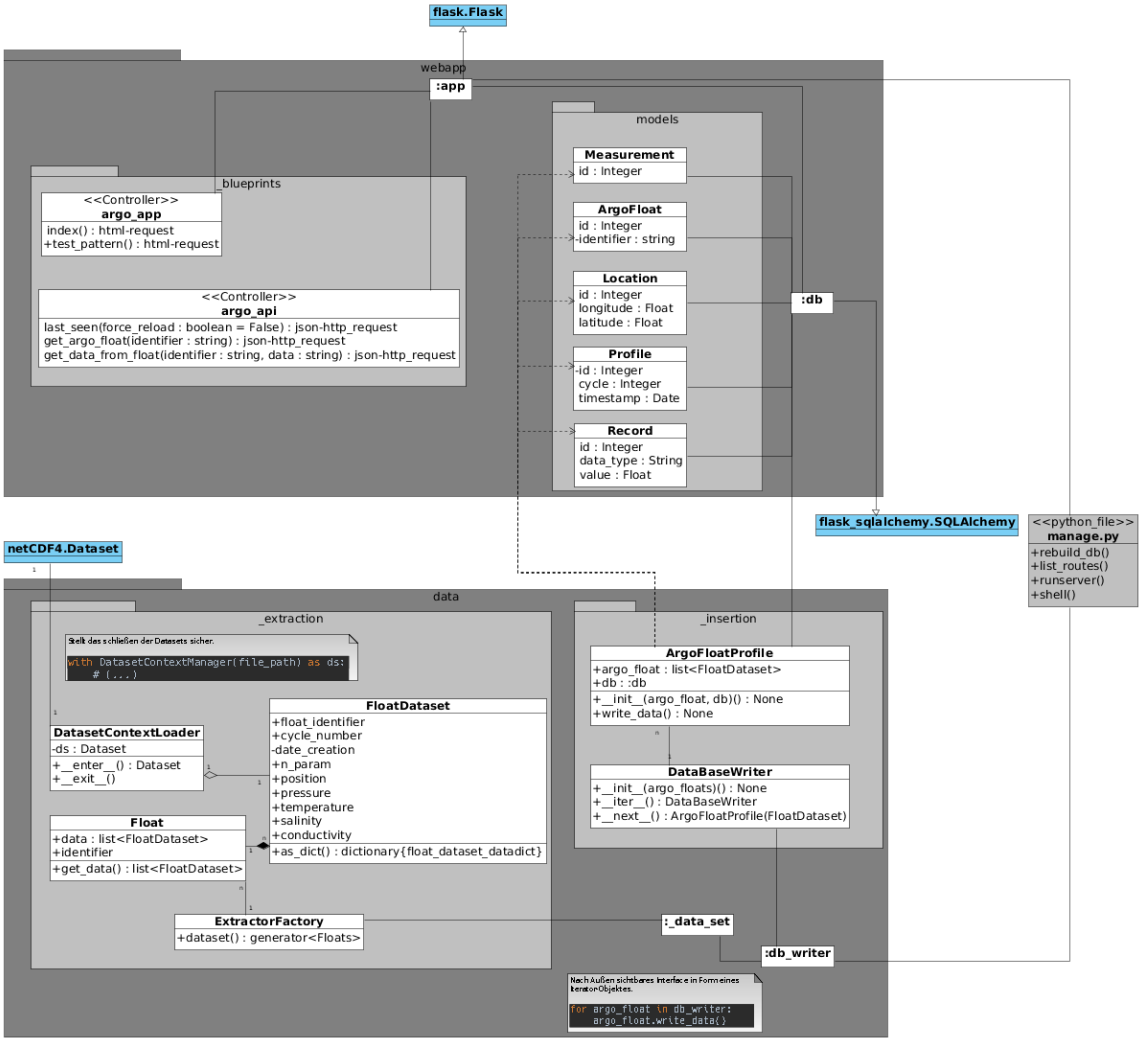



*Abbildung 8: Architekturbeschreibung*

## 6.1 Schnittstellen der Datenaggregation

Bei der Aggregation der Daten aus dem Argo Programm sind Schnittstellen auszuarbeiten. Diese Implementierung ist im folgenden beschrieben.

**Dateninput** Der Datensatz besteht vor der Verarbeitung aus zahlreichen 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.

```
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 9: Implementierung des Kontextmanagers zur Sicherstellung der richtigen Dateibehandlung.label*

**Steuerung** 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 ?? zu sehen. Das Interface wird über die Funktion `def __next__(self)` in Zeile 12 realisiert. Dieses delegiert die Iteration zum Objekteigenen Generator `self.argo_floats`. In dem Moment, in dem ein Objekt aus dem Generator verarbeitet wird, findet die Verarbeitung der dem zugrunde liegenden Datenstruktur statt.
In Zeile 14 wird der Datensatz über ein Objekt ausgeliefert, dass 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          self.argo_floats = argo_floats
7          self.db = db
8          self.app = app
9
10     def __iter__(self):
11         return self
12
13     def __next__(self):
14         self.argo_float = next(self.argo_floats)
15         return ArgoFloatProfile(self.argo_float, self.db, self.app)
```

*Listing 10: Implementierung des Iterators zur Steuerung der Aggregationssequenzlabel*

Die Verwendung der Schnittstelle ist in Listing ?? zu sehen. Dabei wird die Sequenz in Zeile 3 iteriert um den Datensatz daraufhin über die daraufhin folgende Zeile in die Datenbank zu überführen.

```
from data import db_writer

for argo_float in db_writer:
    argo_float.write_data()
```

*Listing 11: Verwendung der Schnittstelle zur Steuerung der Datenaggregationlabel*

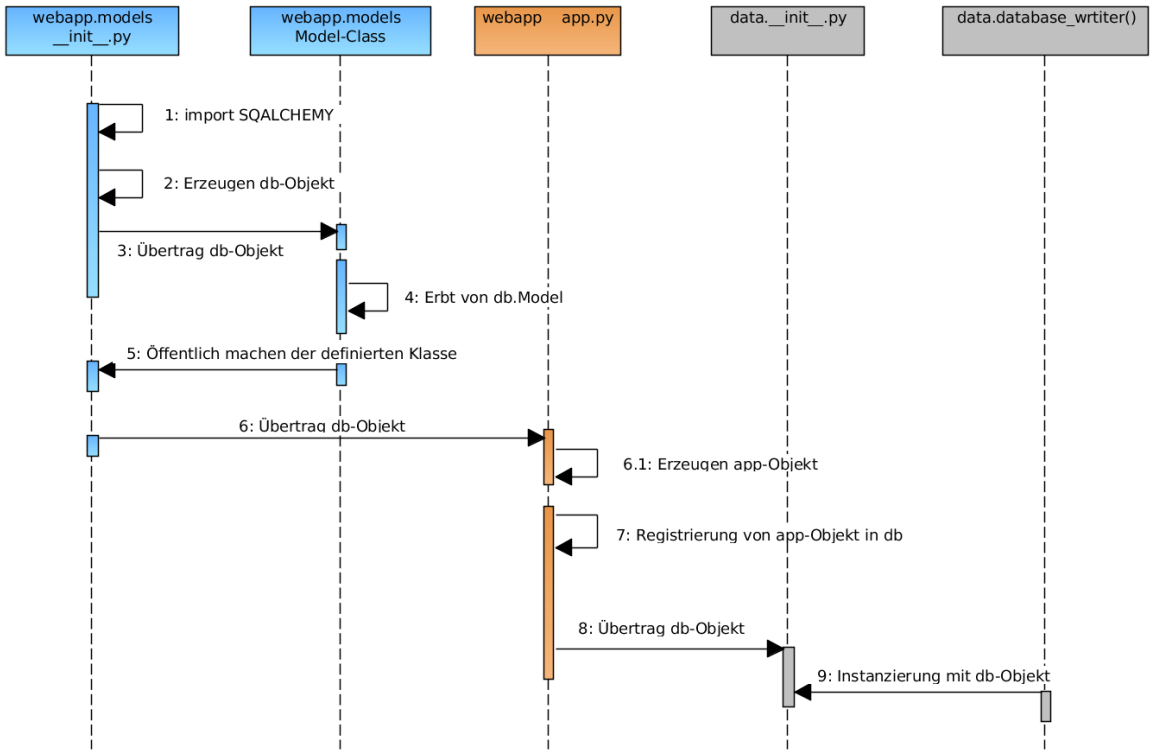
**Die Extraktion der Daten über Lazy Loading** ist eine zentrale Problemstellung des Programmablaufs an dieser Stelle. Aus diesem Grund wurde eine Factory Klasse implementiert um den Vorgang und die Datenstruktur in einem generator-Objekt zu realisieren. Der Vorteil eines Generators gegenüber einer Liste Oder Tupes besteht darin, dass Daten erst zu dem Zeitpunkt angefragt und geschaffen werden, wenn diese aus der Sequenz angefordert werden. In Listing 12 ist die Implementierung dieser Schnittstelle zu sehen. Diese erwartet als Parameter den Pfad zum Verzeichnis, der netCDF-Daten.
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 ArgoFloat Datenobjekte bereitzustellen, ohne diese bereits bei der Instantiierung kennen und abarbeiten zu müssen.

```
import os

from ._float import Float


class ExtractorFactory:
    def __init__(self, data_directory):
        self.extractor_generator = (
            Float(os.path.join(data_directory,
                               profile_directory))
            for profile_directory in os.listdir(data_directory))

        self.__sum_floats = sum([os.path.isdir(os.path.join(
    ↪ data_directory, d)) for d in os.listdir(data_directory)])

    def get_data_sets(self):
        return self.extractor_generator

    def float_count(self):
        return self.__sum_floats
```

*Listing 12: Factory zur Extrahierung der Datensätze.*

## 6.2 SQLAlchemy

Die Einbindung von SQLAlchemy in den Kontext der Webapplikation muss Festgelegten Muster folgen. In Abbildung 6.2 kann dieser Ablauf über ein Sequenzdiagramm nachvollzogen werden.




Die Besonderheit in diesem Prozess besteht darin, das zirkuläre Imports vermieden werden müssen. Daher wird das Objekt für die Verwaltung der Datenbank Schritt für Schritt implementiert.
Das Objekt von SQLAlchemy wird bei der Erstellung der Models mit erzeugt. In Listing 13 ist dargestellt, auf welche Art die Initialisierung des Moduls *Models* erfolgen kann.

```
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 13: Initialisierung der Mpodulstruktur mitsamt SQLAlchemy*

Die Schnittstelle zur Datenbank wird in Zeile 3 über eine Instanz der Klasse SQLAlchemy initialisiert. Diese trägt zu diesem Zeitpunkt noch keinerlei Informationen zu Datenbank

oder verwendeter Models. Erst nachdem dieses Objekt erzeugt wurde, dürfen die Klassen für die Models in den sichtbaren Bereich des Modules gebracht werden.
In Listing 14 ist exemplarisch die Implementierung des Models für ein ArgoFloat aufgezeigt.

```
1  from . import db
2
3
4  class ArgoFloat(db.Model):
5      __tablename__ = 'argo_floats'
6
7      measurements = db.relationship('Measurement', backref='argo_floats',
       ↪  lazy='dynamic')
8
9      id = db.Column(db.Integer, primary_key=True)
10     identifier = db.Column(db.String(10))
11
12     def __init__(self, identifier):
13         self.identifier = identifier
14
15     def __repr__(self):
16         return f'<Argo Float {self.id!r}>'
17
18
19 class ArgoFloatTmpTable(ArgoFloat):
20     __bind_key__ = 'data_input'
```

*Listing 14: Implementierung des Modules für eine Entität eines ArgoFloats*

In Zeile 1 ist zu sehen, dass die Instanz der Datenbankschnittstelle aus dem Modulkontext geladen wird. Dies wäre nicht möglich, wäre das Objekt noch nicht initialisiert, bevor das Model über die `__init__.py` aufgerufen worden wäre.
In Zeile 4 ist zu sehen, das das Model von der Oberklasse `db.Model` erbt. Dadurch wird das in den folgenden Zeilen definierte Model im Kontext der Datenbankschnittstelle registriert und kann nun angesprochen werden. Auf diese Weise werden alle Models an der Schnittstelle registriert.

```
1  from flask import Flask
2  from flask_twisted import Twisted
3
4  from .models import db
5
6  # http://flask.pocoo.org/docs/0.12/patterns/appfactories/
7
8  app = Flask(__name__, static_url_path='/static')
9  app.config.from_pyfile('./argo.cfg')
10
11 db.init_app(app)
12
13 from .blueprints import argo_api, argo_app
14
15 app.register_blueprint(argo_api)
16 app.register_blueprint(argo_app)
```

```
17
18 # Announce all routes of the Application
19
20 from . import models
21
22 twisted = Twisted(app)
```

*Listing 15: Initialisierung des Modules der webapplikation*

## 6.3 Flask Webapp

## 6.4 Kartendarstellung über OpenLayers

# 7 Testen

# 8 Demonstration und Auswertung

# 9 Anhang

# Literatur

[AGS+] Rew AuthorsRuss, 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.

[Arg] Argo float data and metadata from global data assembly centre (argo gdac). http://www.seanoe.org/data/00311/42182/. (Accessed on 10/09/2017).

[Arg17a] Documentation - Argo Data Management. http://www.argodatamgt.org/Documentation, Dec 2017. [Online; accessed 12. Dec. 2017].

[Arg17b] Argo data selection - Argo Data Management. http://www.argodatamgt.org/Access-to-data/Argo-data-selection, Dec 2017. [Online; accessed 12. Dec. 2017].

[CKT+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.

[GSH15] Thomas Gratier, Paul Spencer, and Erik Hazzard. *OpenLayers 3 Beginner s Guide*. Packt Publishing - ebooks Account, 2015.

[net] Python cdf4. http://unidata.github.io/netcdf4-python/.

[Ron11] Armin Ronacher. Opening the flask. http://mitsuhiko.pocoo.org/flask-pycon-2011.pdf, 2011.

[SG] Megan Scanderbeg and John Gould. A beginners' guide to accessing argo data. http://www.argo.ucsd.edu/Argo_data_guide.pdf.