



Westfälische Wilhelms-Universität Münster
Institut für Geoinformatik

Bachelorarbeit
im Fach Geoinformatik

Rich Data Interfaces for Copernicus Data

Themensteller: Prof. Dr. Albert Remke
Betreuer: Dr. Christian Knoth, Dipl.-Geoinf. Matthes Rieke
Ausgabetermin: tbd.
Abgabetermin: 22.08.2022

Vorgelegt von: Alexander Nicolas Pilz
Geboren: 06.12.1995
Telefonnummer: 0176 96982246
E-Mail-Adresse: apilz@uni-muenster.de
Matrikelnummer: 512 269
Studiengang: Bachelor Geoinformatik
Fachsemester: 6. Semester

Inhaltsverzeichnis

1	Einleitung	7
1.1	Motivation	7
1.2	Ziele	7
1.3	Aufbau	8
2	Grundlagen	9
2.1	Radarfernerkundung	9
2.2	Copernicus Programm	11
2.2.1	Ziele	11
2.2.2	Aufbau	11
2.2.3	Sentinel 1	12
2.2.4	Datenzugang	13
2.3	Überschwemmungsmonitoring	14
2.4	Application Programming Interfaces	16
2.4.1	Rich Data Interfaces	18
2.5	OGC und OGC Standards	18
2.6	OGC API - Processes - Part 1: Core	18
2.7	Evaluationskriterien	20
3	Implementierung	21
3.1	Softwarestack	21
3.2	Struktur	21
3.3	Ressourcen	22
3.4	Encodings	23
3.5	HTTP 1.1	23
3.6	Endpoints	24
3.6.1	API Landig Page Endpoint	24
3.6.2	API Definition Endpoint	25
3.6.3	Conformance Declaration Endpoint	25
3.6.4	Process List Endpoint	26
3.6.5	Process Description Endpoint	27
3.6.6	Process Execution Endpoint	28
3.6.7	Job List Endpoint	28
3.6.8	Job Endpoint	28
3.6.9	Job Results Endpoint	29
3.7	Domumentation	29
3.8	Prozesse	29
3.8.1	Echo	29

3.8.2	Überflutungsmonitoring	30
3.9	Zusätzliche Funktionalitäten	30
3.9.1	Coverage	30
3.9.2	Test Suit	30
4	Evaluation	31
5	Diskussion	32
6	Ausblick	33
7	Fazit	34
A	Pseudocode	37
A.1	Ablauf eines Requests an den API Landing Page Endpoint	37
A.2	Ablauf eines Requests an den API Definition Endpoint	38
A.3	Ablauf eines Requests an den Conformance Endpoint	38
A.4	Ablauf eines Requests an den Process List Endpoint	39
A.5	Ablauf eines Requests an den Process Description Endpoint	40
A.6	Ablauf eines Requests an den Process Execution Endpoint	40
A.7	Ablauf eines Requests an den Job List Endpoint	40
A.8	Ablauf eines Requests an den Job Endpoint	42
A.9	Ablauf eines Requests an den Job Results Endpoint	43
A.10	Ablauf eines Requests an den Coverage Endpoint	43
B	Quellcodeverzeichnis	43
B.1	Konfiguration von Werkzeug auf HTTP 1.1	43
B.2	Quellcode Landing Page Endpoint	43
B.3	Quellcode API Definition Endpoint	45
B.4	Quellcode Conformance Endpoint	46
B.5	Quellcode Process List Endpoint	47
B.6	Quellcode Process Description Endpoint	49
B.7	Quellcode Process Execution Endpoint	50
B.8	Quellcode Job List Endpoint	52
B.9	Quellcode Job Endpoint	53
B.10	Quellcode Job Results Endpoint	55
B.11	Quellcode Coverage Endpoint	56
B.12	Quellcode Echo Process	57
C	Schemata	57
C.1	landingPage.yaml	57

C.2	confClasses.yaml	57
C.3	processList.yaml	58
C.4	limit.yaml	58
C.5	link.yaml	58
C.6	processList.yaml	59
C.7	processSummary.yaml	59
C.8	jobControlOptions.yaml	60
C.9	transmissionMode.yaml	60
C.10	process.yaml	60
C.11	inputDescription.yaml	60
C.12	outputDescription.yaml	61
C.13	description.yaml	61
C.14	jobList.yaml	62
C.15	statusInfo.yaml	62
C.16	statusCode.yaml	63
C.17	exception.yaml	63
D	Ressourcen	63
D.1	landingPage.html	63
D.2	landingPage.json	65
D.3	confClasses.html	66
D.4	confClasses.json	67
D.5	processList.html	68
D.6	processDescription.html	69
D.7	FloodMonitoringProcessDescription.json	70
D.8	EchoProcessDescription.json	72
D.9	jobList.html	73
D.10	job.html	74
D.11	coverage.html	74

Abbildungsverzeichnis

1	Prinzip eines SAR Fernerkundungssystems [1]	10
2	Aufnahmeverfahren SAR Systemen [1]	11
3	Modell einer Webanwendung mit API	17
4	Struktur der prototypischen Implementierung [?]	22

Tabellenverzeichnis

1	Gängige Frequenz-Bänder in der Radarfernerkundung [1]	9
2	Eigenschaften der Aufnahmemodi der Sentinel-1 Mission [6]	13
3	Vorgesehene HTTP-Statuscodes [14]	24

Abkürzungsverzeichnis

API Application Programming Interface

OGC Open Geospatial Consortium

SAR Synthetic Aperture Radar

ESA European Space Agency

GMES Global Monitoring for Environmental Security

CAMS Copernicus Atmosphere Monitoring Service

CMEMS Copernicus Marine Environment Monitoring Service

CLMS Copernicus Land Monitoring Service

EMS Emergency Management Service

C3S Climate Change Service

SM Stripmap Mode

SLC Single Look Complex

GRD Ground Range Detected

OSW Ocean Swell Spectra

OWI Ocean Wind Field

RVL Radial Surface Velocity

NDSI Normalized Difference Sigma-Naught Index

DIAS Data and Information Access Services

JSON Java Script Object Notation

URL Uniform Resource Locator

XML Extensible Markup Language

WSGI Web Server Gateway Interface

REST Representational State Transfer

UUID Universally Unique Identifier

HATEOAS Hypermedia As The Engine Of Application State

1 Einleitung

1.1 Motivation

Wofür werden Geodaten über das Web gebraucht? Geodaten werden von einer breiten Nutzerschicht für die unterschiedlichsten Anwendungsfälle benötigt. Zu diesen Anwendungsfällen gehören unter anderem der Umwelt- und Katastrophenschutz. Dort werden flächendeckende, verlässliche und aktuelle Geodaten benötigt. Diese müssen schnell und einfach beschafft und verarbeitet werden können um zum Beispiel rechtzeitig Maßnahmen ergreifen oder mögliche Schäden quantifizieren. Flächendeckende Fernerkundungsdaten von diversen Plattformen werden vom europäischen Copernicus Programm erfasst und im Internet bereitgestellt und können über Webseiten und APIs beschafft werden. Manche Daten, wie die Radardaten der Sentinel-1 Mission, bedürfen jedoch einer komplexen Vorverarbeitung bevor aus ihnen verlässliche Aussagen über die in den abgebildeten Räume stattfindenden Phänomene gemacht werden können. Diese Vorverarbeitungen können nicht von allen Nutzern selbst durchgeführt werden. Zum einen verfügen nicht alle Nutzer über die nötigen Fachkenntnisse und zum anderen stehen nicht jedem Nutzer entsprechende technischen Infrastrukturen zur Verfügung. Um dieses Problem zu lösen und Nutzer mit einsatzbereiten Geodaten oder abgeleiteten Produkten zu versorgen können sogenannte Rich Data Interfaces zum Einsatz kommen. Diese würden Nutzer in die Lage versetzen vorverarbeitete oder abgeleitete Daten direkt über eine API beziehen zu können. Um die Interoperabilität sicherzustellen sollte diese nach den Maßgaben eines Standards implementiert werden. OGC API - Processes - Part 1: Core ist ein Standard ein solcher Standard.

1.2 Ziele

Ziel dieser Arbeit ist Beantwortung der Frage ob sich der OGC API - Processes - Part 1: Core Standards als Grundlage zur Implementierung von Rich Data Interfaces für die Daten des Copernicus Programmes eignet. Zur Klärung dieser Frage soll eine prototypische Web-Anwendung implementiert werden. Teil dieser Anwendung soll eine API sein welche die Vorgaben des OGC API - Processes - Part 1: Core Standards umsetzt. Um einen Bezug zu einem realen Anwendungsszenario zu schaffen sollen Daten welche für das Überschwemmungsmonitoring eignen als Ressourcen über die API abfragbar sein. Die zur Verfügung gestellten Ressourcen sollen aus Daten der Sentinel-1 Satellitenmission abgeleitet werden. Ein Test Suit welcher die Konformität der API zum zugrundeliegenden Standard teilweise überprüft ist soll ebenfalls Teil der Implementierung sein. Schlussendlich soll eine Evaluation der prototypischen Implementierung erfolgen. Dazu sollen die Nutzbarkeitsheuristiken nach Jakob verwendet werden welche die Nutzerfreundlichkeit und Anwendbarkeit in den Fokus ihrer Betrachtung rücken.

1.3 Aufbau

Im ersten Teil dieser Arbeit sollen die nötigen fachlichen Grundlagen kurz erläutert werden. Im Implementierungsteil wird die prototypische Implementierung des Rich Data Interface für Copernicus-Daten vorgestellt. Anschließend erfolgt die Evaluation. Der Diskussionsteil soll Raum für die Erörterung der Forschungsfrage bieten. Im Ausblick soll kurz umrissen werden wie eine hier prototypisch umgesetzte Anwendung weiterentwickelt werden könnte. Als Abschluss der Arbeit soll ein kurzes Fazit dienen welches die gewonnenen Erkenntnisse zusammenfasst.

2 Grundlagen

2.1 Radarfernerkundung

Bei der Radarfernerkundung werden vom Radarsystem in regelmäßigen Abständen elektromagnetische Signale ausgesandt. Nach dem Senden eines Signals (Chirp) folgt ein Zeitfenster, indem die Plattform auf Echos des ausgesandten Signals wartet. Trifft das ausgesandte Signal auf eine Oberfläche, zum Beispiel die Erdoberfläche, wird ein Bruchteil in Richtung Empfänger reflektiert und als Echo vom Fernerkundungssystem empfangen [1].

Die Radarfernerkundung gehört zu den aktiven Fernerkundungsmethoden da hier im Gegensatz zur optischen Fernerkundung nicht nur von Oberflächen reflektierte Strahlung von anderen Strahlungsquellen wie der Sonne aufgenommen wird, sondern das Fernerkundungssystem selbst als Strahlungsquelle dient. Messungen können daher tageszeitunabhängig erfolgen. Bildgebende Radarsysteme werden auf mobilen Plattformen montiert und blicken seitlich auf die zu beobachtende Oberfläche [1] (Abbildung 1).

Die Eigenschaften des reflektierten Signals hängen sowohl von Parametern des Aufnahmesystems als von Parametern der reflektierenden Oberfläche ab. So werden in der Radarfernerkundung verschiedenen Frequenzbänder verwendet, welche sich in Frequenz und Wellenlänge unterscheiden. Da sich die Wechselwirkungen zwischen Signalen unterschiedlicher Frequenzbänder und den reflektierenden Oberflächen unterscheidet können so unterschiedliche Aspekte der beobachteten Oberflächen hervorgehoben werden. Dabei kommen in der Regel Wellenlängen von 0.75m bis 120m zum Einsatz (siehe Tabelle 1). Mit einer größeren Wellenlänge kann ein Medium auch tiefer durchdrungen werden. Außerdem werden Wolken, Dunst und Rauch durchdrungen was den zusätzlich Vorteil bietet wetterunabhängig Messungen durchführen zu können [2].

Tabelle 1: Gängige Frequenz-Bänder in der Radarfernerkundung [1]

Frequenzband	Ka	Ku	X	C	S	L	P
Frequenz (GHz)	40-25	17.6-12	12-7.5	7.5-3.75	3.75-2	2-1	0.5-0.25
Wellenlänge (cm)	0.75–1.2	1.7–2.5	2.5–4	4–8	8–15	15–30	60–120

Die Rauigkeit ist eine Eigenschaft der reflektierenden Oberfläche und hat großen Einfluss auf das reflektierte Signal. Ist diese im Verhältnis zur verwandten Wellenlänge gering so kommt es zu spiegelnden Reflektionen und nur ein geringer Anteil des kehrt zum Empfänger zurück. Zusätzlich ist die Polarisation der ausgesandten und empfangenen Signale bei der Messung ausschlaggebend. Sie können horizontal oder vertikal polarisiert sein. Dies führt zu vier möglichen Polarisationsmodi für das Senden und das Empfangen nämlich HH, VV, HV und VH. Auch die Polarisation sorgt für eine unterschiedliche Wiedergabe von beobachteten Objekten und kann somit verwendet werden, um bestimmte Aspekte hervorzuheben [2]. Die Auflösung entlang des Azimut unterscheidet sich von der Auflösung in Blickrichtung. Die Auflösung in Azimutrich-

tung wird von der Antennenlänge bestimmt da diese festlegt wie lange die Reflektionen eines Objektes empfangen werden. Die Antennenlänge kann bauartbedingt nicht beliebig gesteigert werden. Die Auflösung in Blickrichtung hängt von der Bandbreite ab welche sich aus der Sendefrequenz und der Signaldauer zusammensetzt. Bei Radarsystemen mit einer synthetischen Apertur wird durch die Bewegung des Sensors in Azimutrichtung die wirksame Antennenlänge rechnerisch verlängert indem die reflektierten Signale eines beobachteten Objektes von verschiedenen Standpunkten und unterschiedlichen Zeitpunkten miteinander korreliert werden. So können hohe Azimutaufösungen erzielt werden. Solche Systeme eignen sich auch für den Einsatz auf Satelliten [2].

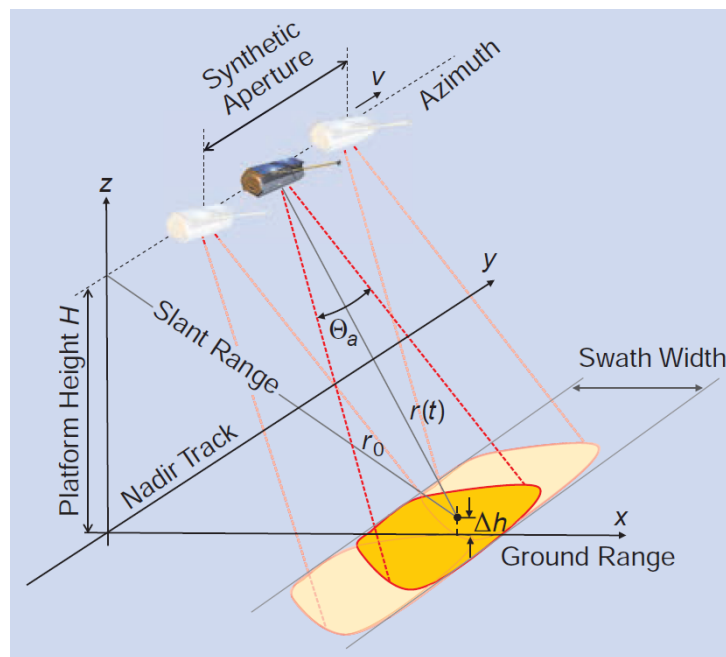


Abbildung 1: Prinzip eines SAR Fernerkundungssystems [1]

Solche Systeme können in unterschiedlichen Aufnahmeverfahren arbeiten. Das einfachste dieser Verfahren ist das Stripmap Verfahren bei dem nur ein Aufnahmestreifen kontinuierlich aufgenommen wird. Breitere Aufnahmestreifen können mit dem ScanSAR Verfahren erzielt werden. Dabei werden unter verschiedenen Depressionswinkeln, in Blickrichtung und zeitversetzt mehrere Subaufnahmestreifen erzeugt. Im Vergleich zum Stripmap Verfahren ist Auflösung jedoch geringer. Wird eine höhere Auflösung benötigt kann das Spotlight Verfahren zum Einsatz kommen, bei dem eine fixe Region über einen längeren Zeitraum hinweg beobachtet wird. Dies führt zu einer sehr langen wirksamen Antenne. Angepasste Verfahren oder Mischformen können je Beobachtungsszenario zum Einsatz kommen (siehe Abbildung 2) [1].

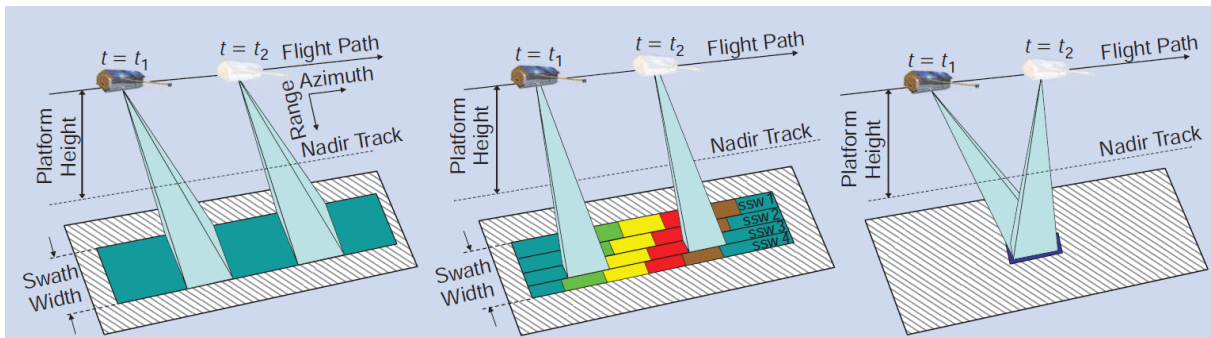


Abbildung 2: Aufnahmeverfahren SAR Systemen [1]

Im Gegensatz zu optischen Aufnahmeverfahren liefern die Rohdaten einer Befliegung mit Radarsensoren noch keine Bilddaten. Um Bilder zu erzeugen, bedarf es zunächst einer komplexen Verarbeitung der aus Amplitude und Phase bestehenden reflektierten Signale. Dabei werden die Daten entlang des Azimuts und der Blickrichtung gefiltert. In der Regel repräsentieren die Pixelwerte eines aus Radardaten abgeleiteten Bildes die Reflektivität des korrespondierenden Bodenelements. Mittels Geocodierung kann das so entstandene Bild verortet werden. Zusätzlich können diverse, ebenfalls rechen- und zeitintensive, Kalibrierungen vorgenommen werden. Dazu gehören Verfahren welche Rauscheffekte minimieren, die geometrischen Eigenschaften verbessern oder die Interpretation der Bilder erleichtern [1].

2.2 Copernicus Programm

2.2.1 Ziele

Das Copernicus-Programm ging aus dem Global Monitoring for Environmental Security Programm (GMES) Programm hervor welches 1998 mit dem Ziel initiiert wurde um Europa zu ermöglichen eine führende Rolle bei der Lösung von weltweiten Problemen im Kontext Umwelt und Klima zu verschaffen. Teil dieser Bestrebungen ist der Aufbau eines leistungsfähigen Programms zur Erdbeobachtung. 2012 wurde das GMES-Programm zum Copernicus-Programm umbenannt [3]. Erklärte Ziele des Copernicus-Programmes ist das Überwachen der Erde um den Schutz der Umwelt sowie Bemühungen von Katastrophen- und Zivilschutzbehörden zu unterstützen. Gleichzeitig soll die Wirtschaft im Bereich Raumfahrt und der damit verbundenen Dienstleistungen unterstützt und Chancen für neue Unternehmungen geschaffen werden [7].

2.2.2 Aufbau

Das Copernicus-Programm besteht aus Weltraum, In-Situ- und Service-Komponente. Zur Weltraum-Komponente gehören die verschiedenen Satellitenmissionen sowie Bodenstationen welche für den Betrieb sowie die Steuerung und Kalibrierung der Satelliten sowie der Verarbeitung und Validierung der Daten verantwortlich sind [7].

Sentinel-1 Satelliten sind mit bildgebenden Radarsystemen ausgerüstet und beobachten wetter-

und tageszeitunabhängig Land-, Wasser- und Eismassen, um unter anderem das Krisenmanagement zu unterstützen. Satelliten der Sentinel-2 Mission führen hochauflösende, multispektrale Kameras mit und liefern weltweit optische Fernerkundungsdaten.

Altimetrische und radiometrische Daten von Land- und Wasserflächen werden von der Sentinel-3 Satellitenmission gesammelt während spektrometrische Daten zur Überwachung der Luftqualität von Sentinel-4 und 5 Satelliten erfasst werden. Ozeanografische Daten sollen von den Sentinel-6 Satelliten geliefert werden [4].

Die In-Situ-Komponente sammelt Daten von See-, luft- und landbasierten Sensoren sowie geografische und geodätische Referenzdaten. Die harmonisierten Daten werden verwendet, um die Daten der Weltraum-Komponente zu verifizieren oder zu korrigieren. Gleichzeitig können räumliche oder thematische Lücken in der Datenabdeckung gefüllt werden [7] [5].

Zur Service-Komponente gehören unterschiedliche Dienste, welche jeweils auf Themengebiet abgestimmt sind und Daten in hoher Qualität bereitstellen. Der Copernicus Atmosphere Monitoring Service (CAMS) soll Informationen zur Luftqualität und der chemischen Zusammensetzung der Atmosphäre liefern. Daten bezüglich des Zustands und der Dynamik der Meere und deren Ökosysteme lassen sich über den Copernicus Marine Environment Monitoring Service (CMEMS) beziehen. Informationen zur Flächennutzung und Bodenbedeckung werden vom Copernicus Land Monitoring Service (CLMS) bereitgestellt. Um eine nachhaltige Klimapolitik planen und umsetzen zu können stellt der Copernicus Climate Change Service (C3S) aktuelle sowie historische Klimadaten bereit. Um den Zivilschutzbehörden schnelle Reaktionen auf Umweltkatastrophen zu ermöglichen, stellt der Emergency Management Service (EMS) entsprechende Fernerkundungsdaten bereit. Ähnliche Daten können von europäischen Zoll- und Grenzschutzbehörden über den Copernicus Security Service bezogen werden [7] [5].

2.2.3 Sentinel 1

Die Sentinel-1 Satellitenmission liefert wetter- und tageszeitunabhängige Radardaten der Erdoberfläche. Die Mission besteht aus zwei Satelliten, Sentinel-1 A und B, sowie einer Bodenkompente welche für Steuerung, Kalibrierung und Datenverarbeitung verantwortlich ist. Die Satelliten tragen als Hauptinstrument ein bildgebendes Radar mit synthetischer Apertur welches im C-Frequenzband arbeitet. Es stehen zwei Polarisationsmodi, Single (HH, VV) oder Dual (HH+HV, VV+VH), zur Verfügung [8]. Die Erfassung von Daten kann in vier Aufnahmemodi erfolgen welche sich in Auflösung, Streifenbreite und Anwendungsszenario unterscheiden (siehe Tabelle 2). Der Standardmodus ist der Stripmap Modus (SM) bei dem Aufnahmestreifen mit einer kontinuierlichen Folge von Signalen abgetastet wird [8]. Die Aufnahmemodi Interferometric Wide Swath Mode (IW) und Extra-Wide Swath Mode (EW) arbeiten im TOPSAR Verfahren mit drei beziehungsweise fünf Sub-Aufnahmestreifen um ein größeres Gebiet aber in geringerer Auflösung aufnehmen zu können. TOPSAR ist eine Abwandlung des ScanSAR Verfahrens bei dem die Antenne zusätzlich in Azimut-Richtung vor und zurück bewegt wird,

um die radiometrische Qualität der resultierenden Bilder zu verbessern. Wenn der Wave Modus (WV) zu Einsatz kommt werden kleine, Vignetten genannte, Szenen im Stripmap Verfahren aufgenommen. Sie werden in regelmäßigen Abständen und wechselnden Depressionswinkeln aufgenommen (siehe Abbildung ??) [1] [8].

Tabelle 2: Eigenschaften der Aufnahmemodi der Sentinel-1 Mission [6]

Modus	IW	WV	SM	EW
Polarisation	Dual	Single	Dual	Dual
Azimutauflösung (m)	20	5	5	40
Rage-Auflösung (m)	5	5	5	20
Streifenbreite (km)	250	20x20	80	410

Beide Satelliten befinden sich auf einem polnahen, sonnensynchronen Orbit. Ein Zyklus dauert 12 Tage, in denen die Erde 175 umrundet wird. Da es sich um ein Satellitenpaar handelt welches als Tandem die Erde umrundet wird ein Punkte alle sechs Tage von einem der Satelliten überflogen. Das System kann eine zuverlässige globale und systematische Abdeckung liefern. Dabei können im IW Modus alle relevanten Land-, Wasser- und Eismassen alle zwölf Tage vollständig von einem Satelliten erfasst werden. In Krisensituationen können nach Bedarf innerhalb von zweieinhalb und fünf Tagen Daten erfasst werden [6].

Nach dem Erfassen der Daten und Übersenden an eine Bodenstation werden diverse Vorverarbeitungsschritte vorgenommen in die sowohl interne also auch externe Parameter einfließen. Daraus ergeben sich diverse Produkte welche sich durch Aufnahmemodus (IW, SM, EW und WV), Produkt-Typ sowie durch ihre Auflösung (Full-, High-, und Medium-Resolution) unterscheiden. Single Look Complex (SLC) Produkte sind im wesentlichen kalibrierte Rohdaten in denen Amplitude und Phase nicht zur Reflektivität kombiniert wurden und die geometrische Auflösung sich in Azimut- und Blickrichtung unterscheidet. Ground Range Detected (GRD) Produkte bilden hingegen die Reflektivität ab und haben eine annähernd quadratische geometrische Auflösung. Die Reflektivität wird in der logarithmischen Maßeinheit Dezibel (dB) angegeben. Die Korrektur der Schrägdistanz in Blickrichtung erfolgt durch Projektion auf einen Ellipsoiden. [8]. Aus den Level-1 Produkten, SLC und GRD, können die Level-2 Produkte OSW, OWI und RVL abgeleitet werden.

2.2.4 Datenzugang

Die Daten des Copernicus-Programmes sollen einer möglichst breiten Nutzergruppe möglichst einfach zugänglich gemacht werden. Sie sollen frei zugänglich und kostenlos angeboten werden [7]. Daten der Sentinel-1, 2, 3 und 5 können über das von der ESA betriebene Copernicus

Open Access Hub bezogen werden. Datensätze können sowohl auf der Webseite als auch mithilfe einer API gesucht und heruntergeladen werden. Der Zugang zu Daten der Sentinel-3, 6 und 4 sowie weiterer Satelliten können über das dem Copernicus Open Access Hub ähnlichen EUMETCast bezogen werden. In Ergänzung zu diesen Quellen werden Daten von fünf privaten, in Kooperation mit dem Copernicus-Programm stehenden Unternehmen in unterschiedlichen Formen bereitgestellt. Diese als Data and Information Access Services (DIAS) bezeichneten Zugänge stellen unverarbeitete und abgeleitete Daten sowie Werkzeuge zur Analyse zur Verfügung [13]. Da die DIAS kommerziell betrieben werden müssen einige Dienste und Werkzeuge bezahlt werden während Nutzer sich lediglich am Copernicus Open Access Hub oder EUMETCast registrieren müssen. Zu erwähnen ist das die DIAS Zugriff auf die gesamten Daten gestatten. Aus dem Copernicus Open Access Hub lassen sich nur Teile der Daten synchron beziehen. In der Regel müssen Daten welche älter als einen Monat sind aus dem Archiv wiederhergestellt werden. Dieser Vorgang kann einige Zeit in Anspruch nehmen.

2.3 Überschwemmungsmonitoring

Um Wasserflächen und damit auch überflutete Areale auf Radarbildern zu erkennen können die Reflektionseigenschaften von Wasserflächen genutzt werden. Das Wasser eine sehr niedrige Rauigkeit besitzt kommt beim Aufprall eines Radarsignals zu einer spiegelnden Reflektion und nur ein sehr geringer Teil des Signals wird zum Empfänger zurückgeworfen. In den resultierenden Bildern äußert sich dieser Umstand in niedrigen Reflektivitätswerten. Um die Areale mit niedrigen Reflektivitätswerten zu detektieren können Verfahren genutzt werden, welche aus den Histogrammen der Bilder einen Schwellwert ermitteln. Um die Ergebnisse einer solchen Schwellwertbestimmung zu verbessern, sollten die Radardaten, zum Beispiel Sentinel-1 IW GRD, zusätzlich Kalibriert werden. So können die genaue Kenntnis über die tatsächliche Flugbahn des Satelliten dazu beitragen die geografische Genauigkeit zu verbessern. Diese kann zusätzlich durch Verfahren wie die Differentialentzerrung gesteigert werden die die durch das Relief entstandenen Lagefehler ausgleicht [2]. Die radiometrische Genauigkeit kann gesteigert werden indem zum Beispiel thermisches Rauschen aus den Daten entfernt wird und die Reflektivitätswerte zum sogenannten σ_0 -Wert umgerechnet werden. Dieser repräsentiert den Querschnitt der Reflektivität für eine normierte Fläche am Boden [11]. Dieses Maß erlaubt zudem das Vergleichen unterschiedlicher Radaraufnahmen. Auch sollte ein Speckle-Filter zum Einsatz kommen um. Dieser reduziert körnige Bildstrukturen welche auf homogenen Flächen in Radarbildern auftreten und die rechnerische Bildauswertung erschweren können. [2] [10]. Auf Basis des Schwellwertes kann eine Binärisierung des Bilder durchgeführt werden. Die entstehenden Werte würden überflutete beziehungsweise trocken liegende Areale repräsentieren [10]. Eines dieses Schwellwertverfahren wurde von Nobuyuki Otsu entwickelt und ist nach ihm benannt. Bei diesem Verfahren werden alle Werte eines Histogramms durchlaufen. Jeder dieser Werte teilt das Histogramm in zwei Gruppen und bildet so einen Schwellwert. Jener Wert welcher

die gewichtete Varianz zwischen der Klassen maximiert wird als optimaler Grenzwert angesehen [9]. Gegeben sei ein Bild C mit N Pixeln in L Grauwertstufen. Die Anzahl der Pixel einer Grauwertstufe i sein dann gegeben durch n_i und es gilt:

$$N = \sum_{i=1}^L n_i \quad (1)$$

Ein betrachteter Grenzwert t teilt das Bild in die Gruppen C_0 und C_1 wobei C_0 alle Pixel der Graustufen 1 bis t und C_1 alle Pixel der Graustufen $t + 1$ bis L enthält. Die Gewichte und Mittelwerte für die Gruppen C_0 und C_1 sind nun gegeben durch:

$$w_0 = w(t) = \sum_{i=1}^t p_i \text{ und } w_1 = \sum_{i=t+1}^L p_i \quad (2)$$

mit p_i :

$$p_i = \frac{n_i}{N} \quad (3)$$

sowie μ_0 , μ_1 und μ_T :

$$\mu_0 = \sum_{i=1}^t i p_i / w_0 \text{ und } \mu_1 = \sum_{i=t+1}^L i p_i / w_1 \text{ und } \mu_T = \sum_{i=1}^L i p_i \quad (4)$$

Die Klassenvarianzen σ_0^2 und σ_1^2 sind gegeben durch:

$$\sigma_0^2 = \sum_{i=1}^t (i - \mu_0)^2 p_i / w_0 \text{ und } \sigma_1^2 = \sum_{i=t+1}^L (i - \mu_1)^2 p_i / w_1 \quad (5)$$

Gesucht wird nun jeder Grenzwert t welcher die Inter-Klassenvarianz K maximiert:

$$K = \frac{\sigma_t^2}{\sigma_W^2} \quad (6)$$

mit σ_W^2 und σ_T^2 :

$$\sigma_W^2 = w_0 \sigma_0^2 + w_1 \sigma_1^2 \text{ und } \sigma_T^2 = \sum_{i=1}^L (i - \mu_T)^2 p_i \quad (7)$$

Die Binärisierung kann direkt auf Basis der Radaraufnahme der Überflutung, oder auf abgeleiteten Daten erfolgen. So können zum Beispiel das Radaraufnahme der Überflutung σ_0^f mit einer überflutungsfreien Referenzaufnahme σ_0^r kombiniert zum Normalized Difference Sigma-Naught Index (NDSI) [12]. Dabei werden die Reflektivitätswerte von zwei unterschiedlichen Zeitpunkten zum NDSI verrechnet welcher als Maß für die Stärke der Veränderung interpretiert werden kann.

$$NDSI = \frac{\sigma_0^f - \sigma_0^r}{\sigma_0^f + \sigma_0^r} \quad (8)$$

Dieses Maß bewegt sich zwischen -1 und 1 wobei Werte um 0 für identische Reflektionswerte an beiden Zeitpunkten und daher für geringe Veränderung stehen. Aufgrund der Reflektions-eigenschaften von Wasserflächen deuten Werte nahe -1 auf überflutete Areale hin [12]. Die der vielen und teilweise zeitintensiven Prozessierungsschritte können, je nach Größe des zu untersuchenden Areals, viel Zeit und Rechenleistung in Anspruch nehmen.

2.4 Application Programming Interfaces

Schnittstellen sind gemeinsame Grenzen zwischen funktionalen Einheiten über die mittels vorgegebener Kommunikationswege Informationen ausgetauscht werden können [17]. Ein Application Programming Interface erlaubt also den Austausch von Informationen zwischen zwei unterschiedlichen Programmen. Genauer ausgedrückt erlaubt eine API einem Programm Funktionen und Ressourcen eines anderen Programms zu nutzen. Die Nutzung von APIs erlaubt die Modularisierung von Anwendungen in voneinander unabhängige, und untereinander austauschbare Module [17].

Bei diesen kann es sich zum Beispiel um Web-Anwendungen handeln welche ihre Informationen über standardisierte Protokolle, zum Beispiel HTTP, über das Internet bereitstellen. Mit Requests können diese Informationen über die API angefragt werden. Die API beantwortet einen Request mit einem Response welcher die im Request spezifizierten Informationen enthält.

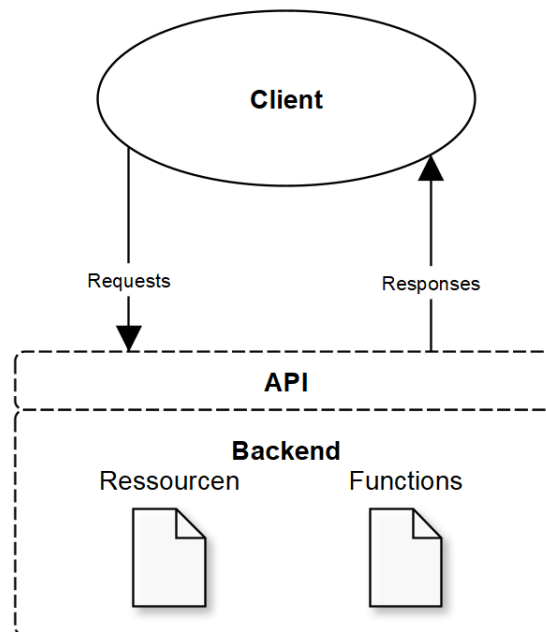


Abbildung 3: Modell einer Webanwendung mit API

Moderne Web-Anwendungen werden häufig nach dem REST Architekturstil entwickelt. Dieses beschreibt eine Reihe von Einschränkungen und Vereinbarungen für verteilte Hypermedia-Systeme wie das Internet [21]. Durch die Umsetzung der REST-Prinzipien soll das HTTP und auf diesem Basierende Infrastrukturen optimal genutzt sowie Webanwendungen skalierbar, ausfallsicher, transparent und zuverlässig wie möglich werden [21] [18]. Ein zentraler Begriff ist der der Ressourcen. Eine Ressource ist eine abstrakte Datenstruktur welche in verschiedenen konkreten Repräsentationen vorliegen kann. Typische Formate für Repräsentationen sind XML, JSON oder HTML. Ressourcen haben eine eindeutige URI welche häufig die Gestalt eines URL annimmt. Die Interaktion mit Ressourcen erfolgen mit den HTTP-Methoden Get, Post, Put und Delete [21]. Als RESTful bezeichnete Systeme sollen zudem stateless operieren. Dies bedeutet das alle für Beantwortung eines Requests nötigen Informationen im Request selber enthalten sind. Zusätzlich hängt der Erfolg oder Misserfolg eines Requests nicht von dem Ergebnis eines Anderen ab [18,21]. Allerdings sollen Responses zwischengespeichert werden können um häufig auftretenden Requests schneller beantworten zu können [18]. Die REST-Prinzipien können um die des HATEOAS-Architekturstil erweitert werden. Nach diesem sollte eine API selbstbeschreibend sein. Sämtliche Interaktionsmöglichkeiten mit Ressourcen sollten mit den Repräsentationen verknüpft sein. Zudem sollte jede Ressource Verknüpfungen zu anderen Ressourcen enthalten. Ein Client benötigt also nur Zugang zu einer Ressource um alle anderen Ressourcen zu erreichen. Dies Bedeutet auch das der Client nicht a-priori Kenntniss über die Details der API haben muss um mit ihr zu interagieren [21].

2.4.1 Rich Data Interfaces

Der Begriff Rich Data Interface kann unter zwei Gesichtspunkten betrachtet werden. Einerseits kann darunter eine API verstanden werden welche reich an Interaktionsmöglichkeiten ist, dem Nutzer also umfangreiche Funktionen zur Interaktion mit den durch die von der API angebotenen Ressourcen ermöglicht. Andererseits könnten die Ressourcen selber reich an Informationen sein. Reichhaltige Ressourcen können zum Beispiel vorprozessierte Daten sein welche von Nutzern direkt für weitere Analysen verwendet werden können ohne das diese selber die möglicherweise Aufwendige Vorprozessierung durchführen müssen.

Unter reichhaltigen Ressourcen können jedoch auch aus Rohdaten abgeleitete Produkte verstanden werden.

2.5 OGC und OGC Standards

Das Open Geospatial Consortium (OGC) widmet sich der Aufgabe die Entwicklung von internationalen Standards und unterstützender Dienste welche die Interoperabilität im Bereich der Geoinformatik verbessern voranzutreiben. Das OGC soll dabei offene Systeme und Techniken verbreiten welche es erlauben Dienste und Prozesse mit Raumbezug in Kreisen der Informatik verbreiten und die Nutzung von interoperabler und kommerzieller Software fördern [16]. Dabei wird versucht möglichst viel Akteure aus Wissenschaft, Wirtschaft und Verwaltung zu beteiligen um Standards zu schaffen welche auf möglichst breitem Konsens basieren. Zudem werden Mechanismen zur Zertifizierung von standardkonformen Software-Lösungen angeboten [16].

Das OGC formuliert Standards für unterschiedliche Themenbereiche. In Standards aus dem Bereich Data Models und Encodings werden Daten- und Datenaustauschformate definiert. Standards welche Webdienste und Schnittstellen zum Austausch von Geodaten beschreiben werden dem Bereich Services und APIs zugeordnet. Der OGC API - Processes - Part 1: Core Standard gehört in diesen Bereich. Die Bereiche Discovery und Containers enthalten Standards für die Speicherung von Geodaten sowie die Auffindbarkeit und Durchsuchbarkeit dieser. Ein weiterer Bereich widmet sich Standards zum Thema Sensornetzwerke.

2.6 OGC API - Processes - Part 1: Core

Der OGC API - Processes - Part 1: Core Standard soll das Bereitstellen von aufwendigen Prozessierungsaufgaben und ausführbaren Prozessen welche über eine Web API von anderen Programmen aufgerufen und gestartet werden können unterstützen [14]. Der Standard ist dabei von Konzepten des OGC Web Processing Service 2.0 Interface Standards beeinflusst und bedient sich des REST Architekturstils sowie der Java Script Object Notation (JSON).

Der Aufbau des OGC API - Processes - Part 1: Core Standards orientiert sich am OGC Spezifikationsmodell. Dieses beschreibt die modularen Komponenten eines Standards und wie diese

miteinander in Verbindung stehen [15]. Das Spezifikationsmodell definiert einen Standard als Teillösung eines Entwicklungsproblems. Diese Teillösung limitiert die Anzahl an möglichen Implementierungen. Ziel ist die Harmonisierung der Implementierungen und so die Interoperabilität zu steigern.

Der OGC API - Processes - Part 1: Core Standard formuliert Requirements, Recommendations und fasst diese zu Requirements-Classes zusammen welche wiederum ein Standardisierungsziel beschreiben. Requirements beschreiben Eigenschaften oder Vorgehensweisen die die Implementierung umsetzen muss um standardkonform zu sein. Recommendations sind hingegen nicht verpflichtend beschreiben aber aus Sicht der Autoren empfehlenswerte Eigenschaften oder Vorgehensweisen [15].

Jedes Requirement kann mit einem ebenfalls im Standard definierten Conformance-Test-Case überprüft werden. Diese Tests können zu Conformance-Test-Modules zusammengefasst welche alle Test zum prüfen einer Requirements-Class umfassen. Die Gesamtheit dieser Conformance-Test-Modules wird auch als Conformance-Test-Class bezeichnet. Alle vom Standard definierten Conformance-Test-Classes werden im Conformance-Suit zusammengefasst [15]. Erfüllt eine Implementierung alle im Conformance-Suit definierten Tests kann sie mit einem Certificate of Conformance für die implementierten Requirements-Classes versehen werden. Requirements, Recommendation und Conformance-Suit bilden gemeinsam eine Spezifikation welche nach der Anerkennung durch ein legitimes Gremium wie das OGC als Standard angesehen werden.

Der OGC API - Processes - Part 1: Core Standard definiert sieben Requirements-Classes welche Endpoints sowie deren Funktionen, Responses und mögliche Fehlersituationen umfassen. Die Requirements-Class Core beschreibt dabei die Kernfunktionalitäten welche von standardkonformen Implementierungen umgesetzt werden.

Da dem Nutzer mit diesen Kernfunktionalitäten Ressourcen zugänglich gemacht werden sollen werden in den Requirements-Classes JSON und HTML Repräsentationen dieser Ressourcen in JSON und HTML definiert [14].

Die Requirements-Class Core macht keine expliziten Vorgaben für die Beschreibung einer standardkonformen API. Solche Vorgaben finden sich in der Requirements-Class OpenAPI Specification 3.0. Die Requirements-Class OpenAPI Specification 3.0 definiert wie implementierte APIs mithilfe der OpenAPI 3.0 Spezifikation beschrieben und dokumentiert werden sollen [14]. Da der Standard primär, aber nicht ausschließlich, zum Breitstellen von Diensten aus dem Bereich der Geoinformatik genutzt werden soll wird in der Requirements-Class OGC Process Description die Nutzung des OGC Processes Description Formats zum Beschreiben von angebotenen Prozessen empfohlen [14].

Zusätzliche Funktionen werden in den Requirements-Classes Job-List, Callback und Dismiss beschreiben. Sie beschreiben zusätzliche Ressourcen und Interaktionsmöglichkeiten mit den auszuführenden Instanzen eines Prozesses welche als Jobs bezeichnet werden [14].

2.7 Evaluationskriterien

Die Evaluation einer Implementierung einer API kann unter verschiedenen Gesichtspunkten erfolgen. Zum einen können technische Aspekte Effizienz, Skalierbarkeit, Stabilität und Wartbarkeit untersucht werden. Zu dieser technischen Bewertung einer API kann auch das Überprüfen der Standardkonformität gezählt werden. Diese kann durch einen ebenfalls zu implementierenden Unit-Test teilweise überprüft werden da manche Testfälle des Test-Suits des Standards automatisch überprüft werden können [14]. Der Fokus der in dieser Arbeit durchgeführte Evaluation soll jedoch die Benutzbarkeit und Benutzerfreundlichkeit der Implementierung sein. Dafür können die von Jakob Nielsen 1993 aufgestellten Heuristiken verwendet werden. Dabei handelt es sich um zehn Heuristiken unter denen eine Schnittstelle betrachtet werden kann [19]. Die Heuristiken decken unter anderem die Themenfelder Verständlichkeit, Fehlerbehandlung, Dokumentation und Konsistenz ab [19].

3 Implementierung

3.1 Softwarestack

Die prototypische Entwicklung eines leichtgewichtigen Rich Data Interface für Copernicus-Daten erfolgte im Rahmen dieser Arbeit mit der Programmiersprache Python. Diese ist nicht nur aufgrund ihrer Einfachheit vorteilhaft sondern erlaubt auch den Zugriff auf eine große Zahl von Packages für die unterschiedlichsten Anwendungsfälle. Weite Teile des Rich Data Interface wurden mithilfe des Flask-Frameworks umgesetzt. Dieses erlaubt das schnelle Entwickeln einer leichtgewichtigen API. Neben Python Standard-Bibliotheken kommen nur wenige zusätzliche Packages zum Einsatz. Um zu Anfragen passende Sentinel-1 Datensätze aus dem Copernicus-Access-Hub zu finden kommt das Package `sentinelsat` zum Einsatz. Das Herunterladen der Sentinel-1 Datensätze erfolgt ebenfalls mit diesem Package. Die Prozessierungen der Sentinel-1 Daten werden mit dem Package `snappy` durchgeführt. Dieses ist ein Wrapper für die SNAP Software welche im Rahmen des Copernicus-Programmes kostenlos zur Verfügung gestellt wird. Mit `snappy` lassen sich die Funktionen der Sentinel-1 Toolbox in Python verwenden. Statistische Auswertungen der Radarbilder werden mit dem Package `skimage` durchgeführt. Da die Radarbilder jedoch auch zugeschnitten und mathematisch zu kombinieren sind werden auch die Packages `gdal`, `ogr` und `osr` verwendet.

3.2 Struktur

Die Anwendung ist in vier Python-Skripte aufgeteilt. Im `api.py` Skript ist die API der Anwendung definiert. Die geordnete Abarbeitung der angelegten Jobs werden vom Skript `processing.py` nach dem Modell der Queue gesteuert. Dies bedeutet das zeitgleich nur ein Job ausgeführt wird. Nach dessen Beendigung wird der älteste angelegte Job gewählt und bearbeitet. Die eigentlichen Prozesse sowie Hilfsfunktionen befinden sich im `utils.py` Skript. Dazu zählen auch die eigentlichen Prozesse sowie Parser für ihre Eingabedaten. Das `test.py` Skript kann dazu genutzt werden um die Stabilität und Standardkonformität der API zu testen und enthält des Test-Suit.

Diese Skripte verwalten Dateien in einem einfachen Verzeichnissystem. Templates für statische Ressourcen befinden sich im Verzeichnis `templates/`. HTML-Dateien befinden sich im Verzeichnis `templates/html/` und JSON-Dateien im Verzeichnis `templates/json/`. Das Unterverzeichnis `templates/json/processes/` enthält die Beschreibungen der von der Anwendung angebotenen Prozesse. Die Anwendung erlaubt das persistente hinterlegen von Sentinel-1 Datensätzen um zeitaufwendiges Herunterladen zu vermeiden. Diese Datensätze können im Verzeichnis `data/` abgelegt werden. Jeder Sentinel-1 Datensatz enthält eine `.kml`-Datei welche Metadaten zum Datensatz enthält. Diese werden im Unterverzeichnis `data/coverage/` abgelegt. Jeder angelegte Job, also jede auszuführende Instanz eines Prozesses erhält ein einzigartiges Verzeichnis innerhalb des Verzeichnisses `jobs/`. In diesem Verzeichnis befinden sich eine Status- und ei-

ne Job-Datei sowie ein weitere für die Bearbeitung des Jobs benötigte Dateien. Neben diesen Dateien enthält jedes Job-Verzeichnis ein Unterverzeichnis *results/* in dem die Ergebnisse des jeweiligen Jobs abgelegt werden.

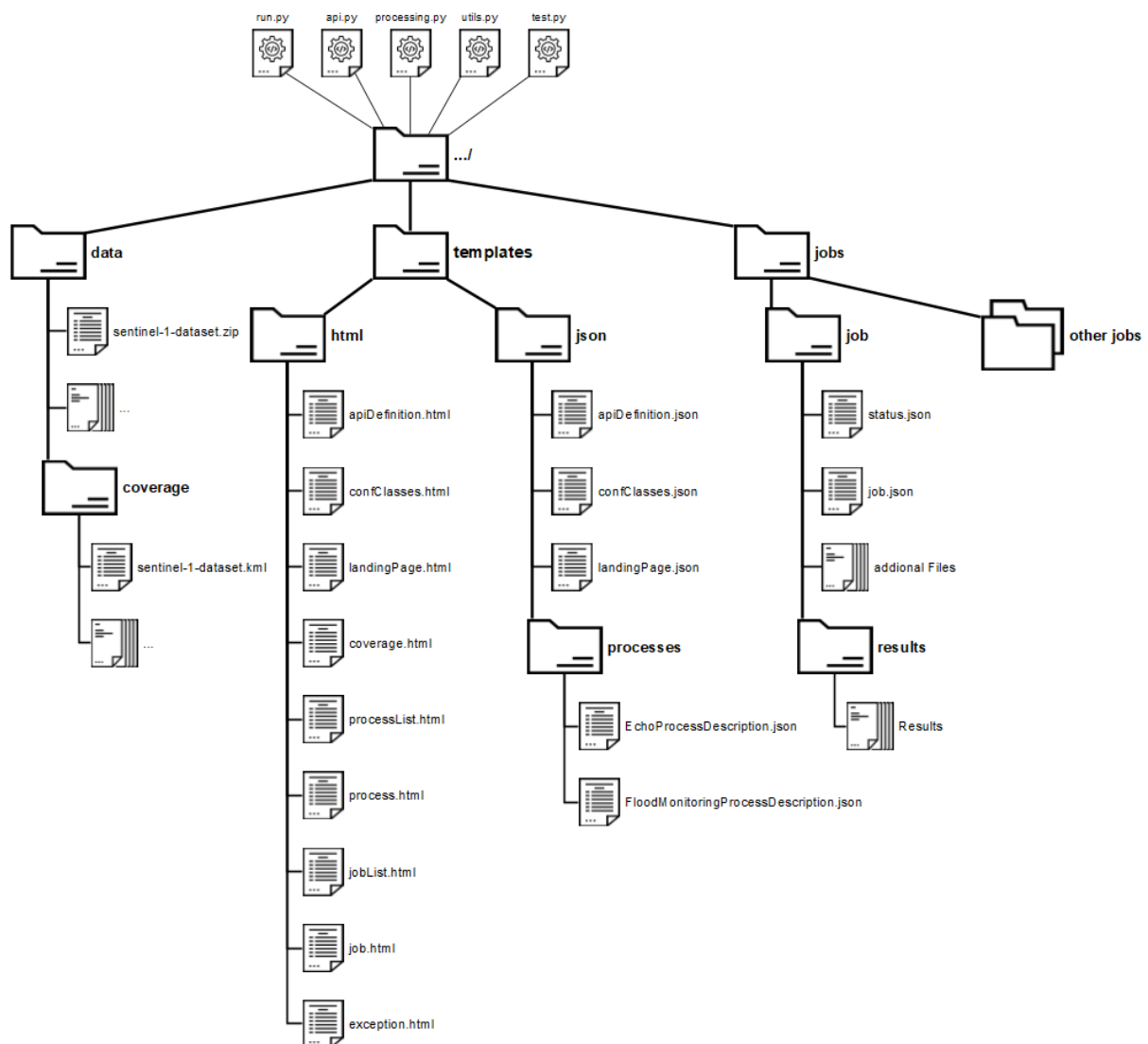


Abbildung 4: Struktur der prototypischen Implementierung [?]

Insgesamt handelt es sich bei der prototypischen Implementierung eines Rich Data Interfaces for Copernicus-Daten also um eine Anwendung welche Überschwemmungsmasken und Daten zur Hochwasseranalyse als Ressourcen bereitstellt. Zugriff auf diese Ressourcen erhalten Nutzer über eine OGC API - Processes - Part 1: Core standardkonforme API.

3.3 Ressourcen

Ressourcen sind die über die Endpoints der API bereitgestellten Informationen und Dateien. Sie können in unterschiedlichen Repräsentationen vorliegen. Die Ressourcen werden vom OGC API - Processes - Part 1: Core Standard mittels Schemata im .yaml-Format beschrieben. Diese

definieren neben den Bezeichnungen für Elemente auch ihre Datentypen und ob sie optional sind oder nicht. Die meisten angebotenen Ressourcen können um Media-Type *text/html*, also als HTML-Dateien oder im Media-Type *application/json* also als JSON-Dateien angefragt werden. Um aus diesen Dateien einen Response zu generieren werden .html-Dateien zuvor mit der Methode *render_template()*, welcher auch dynamische Inhalte übergeben werden können gerendert während .json-Dateien zunächst geladen und anschließend mit der Methode *jsonify()* bearbeitet werden. Beide genannten Methoden geben ein Response-Objekt zurück welches zusammen mit Headern und einem HTTP-Statuscode versandt werden kann.

Jede Ressource enthält zudem eine Verknüpfungen zu sich selbst mit der Relation *self* und gegebenenfalls eine Verknüpfung zur Ressource im jeweils anderen Media-Type mit der Relation *alternate*.

3.4 Encodings

In der Requirements Class JSON wird definiert welche Ressourcen im Media-Type *application/json* angefragt werden können. Dazu gehören alle Responses der Endpunkte API Landing Page, API Definition, Conformance Deklaration, Prozess Liste, Prozess Beschreibung, Prozess Ausführung und Job Status welche mit dem HTTP-Statuscode 200 versandt werden. Da die prototypische Implementierung auch die Endpunkte Job Liste und Coverage bereitstellt können die korrespondierenden Ressourcen auch im Media-Type *application/json* angefragt werden.

In der Requirements-Class HTML werden analog zur Requirements Class JSON jene Ressourcen definiert welche im Media-Type *text/html* angefragt werden können. Jedoch entfällt in dieser Requirements-Class die Einschränkung auf bestimmte Endpunkte und alle Responses welche mit dem HTTP-Statuscode 200 versandt werden müssen den Media-Type *text/html* unterstützen.

Stellen Endpoints ihre Ressourcen sowohl den Media-Type *application/json* als auch *text/html* zur Verfügung so können Nutzer diesen über den optional Parameter *f* oder *content_type* spezifizieren. Wird kein Media-Type über diese Parameter spezifiziert so wird standardmäßig der Media-Type *text/html* verwendet.

3.5 HTTP 1.1

Die Umsetzung des Requirements HTTP 1.1 (RFC 2616) aus der Requirements-Class Core verlangt das die API exklusiv das HTTP 1.1 unterstützt. Das Flask-Framework nutzt standardmäßig das HTTP 1.0. Teil des Flask-Frameworks ist die WSGI Bibliothek Werkzeug welche das Implementieren von Webanwendungen erlaubt. Um die verwendete HTTP-Version von 1.0 auf 1.1 umzustellen müssen Variablen in Werkzeug angepasst werden. Nach dem Import der

Module WSGIRequestHandler und BaseWSGIServer kann in beiden die Version des HTTP Protokolls angepasst werden (siehe Anhang ??).

In diesem Requirement werden zudem alle HTTP-Statuscodes gelistet die Nutzer von einer standardkonformen Implementierung mindestens erwarten können.

Tabelle 3: Vorgesehene HTTP-Statuscodes [14]

HTTP-Statuscode	Bedeutung
200	OK
201	Created
204	No Content
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
410	Gone
429	Too Many Requests
500	Internal Server Error
501	Not Implemented

Alle erfolgreichen Anfragen welche eine Resource liefern mit dem HTTP-Statuscode 200 beantwortet. Die Verwendung nicht zulässiger HTTP-Methoden resultieren in Antworten mit dem Status-Code 405 während Anfragen für nicht unterstützte Media-Types mit dem Status-Code 406 beantwortet werden. Kommt es zu Fehlern bei der Ausführung des Programmcodes antwortet die Anwendung mit dem HTTP-Statuscode 500. Werden durch eine Anfrage Ressourcen neu erzeugt oder nicht gefunden antwortet die Anwendung mit den HTTP-Statuscodes 201 beziehungsweise 404. Der Standard erlaubt die Nutzung weiterer HTTP-Statuscodes [14].

3.6 Endpoints

3.6.1 API Landig Page Endpoint

Der API Landing Page Endpoints ist Teil der Requirements-Class Core. Der Endpoint kann über den URL *http://HOST:PORT/?f=<MEDIA-TYPE>* angefragt werden und liefert als Resource die API Landing Page (siehe Anhang C.1). Die einzig zulässige HTTP-Methode für diesen Endpoint ist die HTTP-Get Methode.

Die API Landing Page kann als Eintrittspunkt zu allen anderen Funktionalitäten der Anwendung bezeichnet werden. Sie enthält Verknüpfungen zu den Endpoints, API Landing Page, API Definition, Conformance, Process List, Process Description, Job List und Coverage.

Die API Landing Page kann in den Media-Types *text/html* (siehe Anhang D.1) und *application/json* (siehe Anhang D.2) abgerufen werden.

Ein an den API Landing Page Endpoint gerichteter Request wird zunächst auf die verwendete HTTP-Methode hin überprüft. Handelt es sich nicht um die HTTP-Get Methode wird der HTTP-Statuscode 405 als Response versandt. Im nächsten Schritt erfolgt die Prüfung des angefragten Media-Types. Enthält der Parameter *f* oder der Parameter *content_type* einen nicht unterstützten Media-Type wird als Response der HTTP-Statuscode 406 versandt. Wird der Media-Type nicht spezifiziert oder *text/html* angefragt wird die landingPage.html an die Funktion an den Renderer übergeben um ein Response-Objekt zu erzeugen. Spezifiziert der Request den Media-Type *application/json* wird das Response-Objekt aus der landingPage.json erzeugt. Dieses wird mit dem HTTP-Statuscode 200 sowie den Headern *resource* und *link* versandt. Ersterer enthält den Wert *landingPage* während letzterer den URL zur API Landing Page im angefragten Media-Type enthält. Kommt es während der Ausführung zu einem Fehler im Programmcode wird mit dem HTTP-Statuscode 500 geantwortet (siehe Anhang 1 und B.2) [14].

3.6.2 API Definition Endpoint

Der API Definition Endpoint gehört ebenfalls zu Requirements-Class Core. Der API Definition Endpoint kann über den URL *http://HOST:PORT/api?f=<MEDIA-TYPE>* angefragt werden und liefert als Ressource die API Definition. Auch für diesen Endpoint ist die einzig zulässige HTTP-Methode Get. Die API Definition enthält detaillierte Informationen zur API. In ihr sind alle verfügbaren Endpoints mit ihren Parametern und Responses aufgeführt. Auch die API Definition kann in den Media-Types *text/html* und *application/json* abgerufen werden.

Requests an den API Definition Endpoint werden ebenfalls zunächst auf die verwendete HTTP-Methode hin überprüft. Wird nicht HTTP-Get verwendet wird als Response der HTTP-Statuscode 405 versandt. Wird im Request mittels des Parameters *f* oder *content_type* ein nicht unterstützter Media-Type angefragt wird ein Response mit dem HTTP-Statuscode 406 versandt. Requests welche keinen oder einen unterstützten Media-Type spezifizieren lösen die Generierung eines Response-Objektes aus entweder der apiDefinition.html oder apiDefinition.json aus. Der *resource*-Header erhält den Wert *apiDefinition*. Dieser wird als Teil des Responses mit dem HTTP-Statuscode 200 versandt. Ebenfalls Teil dieses Responses ist ein *link*-Header welche einen zum request passenden URL enthält. Auch diese Endpoint fängt Fehler bei der Ausführung mit dem versenden des HTTP-Statuscodes 500 ab (siehe Anhang 2 und B.3) [14].

3.6.3 Conformance Declaration Endpoint

Ein weiterer zur Requirements-Class Core gehörender Endpoint ist der Conformance Declaration Endpoint. Der Conformance Declaration Endpoint kann über den URL *http://HOST:PORT/conformance?f=*

TYPE> angefragt werden und liefert als Ressource die Conformance Declaration (siehe Anhang C.2). Get ist ebenfalls die einzige zulässige HTTP-Methode für Requests an diesen Endpoint. Die Conformance Declaration enthält Verknüpfungen zu allen von der Anwendung implementierten Requirements Classes und steht in den Media-Types *text/html* (siehe Anhang D.3) und *application/json* (siehe Anhang D.4) zu Verfügung.

Wie bereits beschrieben wird ein ankommender Request zunächst auf die von ihm verwendete HTTP-Methode überprüft und bei nicht Zulässigkeit dieser mit dem HTTP-Statuscode 405 geantwortet. Ebenso erfolgt eine Überprüfung des mit dem Parameter *f* oder *content_type* spezifizierten Media-Types wobei die Anfrage eines nicht unterstützten Media-Types in einem Response mit dem HTTP-Statuscode 406 resultiert. Je nach im Request spezifizierten Media-Type wird das Response-Objekt entweder aus der *confClasses.html* oder der *confClasses.json* generiert. Der *resource*-Header enthält den Wert *conformance* während der *link*-Header eine zum Request identische URL enthält. Response-Objekt und Header werden mit dem HTTP-Statuscode 200 versandt. Auftretende Fehler werden mit einem Response mit dem HTTP-Statuscode 500 abgefangen (siehe Anhang 3 und B.4) [14].

3.6.4 Process List Endpoint

Der ebenfalls zur Requirements-Class Core gehörende Process List Endpoint kann über den URL *http://HOST:PORT/processes?f=<MEDIA-TYPE>&limit=<INTEGRER>* angefragt werden. Request sind nur mit der HTTP-Methode Get zulässig. Als Ressource erhalten Nutzer eine detaillierte Liste der durch die Anwendung angebotenen Prozesse (siehe Anhang C.6). In dieser Liste finden sich die Bezeichnungen, Steueroptionen sowie die Ein- und Ausgaben der Prozesse. Die Process Liste kann in den Media-Types *text/html* (siehe Anhang D.5) und *application/json* angefragt werden.

Wie bei der Beantwortung von Requests an die bereits vorgestellten Endpoints werden Requests mit unzulässigen HTTP-Methoden oder nicht unterstützten Media-Types, welche ebenfalls mit den Parametern *f* oder *content_type* spezifiziert werden, mit Responses welche die HTTP-Statuscodes 405 beziehungsweise 406 enthalten beantwortet. Zusätzlich erfolgt allerdings eine Prüfung des *limit*-Parameters. Dieser kann dazu verwendet werden die Liste auf eine spezifizierte Anzahl von Objekten einzuschränken. Wird ein Limit kleiner 0 oder größer 10000 spezifiziert wird der Wert auf den Standardwert 10 gesetzt. Unabhängig vom gewählten Media-Type werden zunächst alle Prozess Beschreibungen aus dem Verzeichnis */templates/json/processes/* geladen und in einem Array gespeichert. Soll der Response im Media-Type *text/html* versandt werden wird dieses mit dem *limit*-Parameter eingeschränkt und dem Renderer zusammen mit der *processList.html* übergeben. Das entstehende Response-Objekt wird zusammen mit *resource*- und *link*-Header sowie dem HTTP-Statuscode 200 versandt. Für Generierung eines Responses im Media-Type *application/json* wird das auf das spezifizierte Limit

eingeschränkte Array zum einem JSON-Objekt hinzugefügt welches noch um die Verknüpfungen *self* und *alternate* ergänzt wird. Aus diesem Objekt wird ein Response-Objekt generiert welches zusammen mit den beiden Headern *link* und *resource* und dem HTTP-Statuscode 200 versandt wird. Der *resource*-Header enthält dabei den Wert *processes*. Treten während der beschriebenen Schritte Fehler auf wird ein Response mit dem HTTP-Statuscode 500 versandt (siehe Anhang 4 und B.5) [14].

3.6.5 Process Description Endpoint

Unter dem URL *http://HOST:PORT/processes/<processID>?f=<MEDIA-TYPE>* kann der Process Description Endpoint angefragt werden. Dieser ist Teil der Requirements-Class Core. Welche Prozessdetails zurückgegeben werden hängt vom *processID*-Parameter ab. Alle Prozesse werden über eine eindeutige Bezeichnung, die *processID* gekennzeichnet. Sie kann der Prozess Liste entnommen werden. Um eine Prozess Beschreibung anzufragen darf nur die HTTP-Get Methode verwendet werden. Als Ressource liefert der Endpoint eine detaillierte Beschreibung des im Request spezifizierten Prozesses und enthält Informationen zu den Steueroptionen sowie den Ein- und Ausgaben des Prozesses. Die Beschreibungen sind dabei so strukturiert wie durch die Requirements-Class OGC Process Description vorgegeben (siehe Anhang C.10). Wie die Prozess Liste kann die auch die Process Beschreibung im Media-Type *text/html* (siehe Anhang D.6) oder *application/json* angefragt werden. Jeder Prozess der von der Anwendung angeboten wird muss eine eigene Prozess-Beschreibung haben (siehe Anhang D.8 und D.7).

Wird von diesem Endpoint ein Response generiert werden zunächst die verwendete HTTP-Methode sowie der angefragte Media-Type, welcher über die Parameter *f* oder *content_type* spezifiziert wird, überprüft. Ungültige beziehungsweise nicht unterstützte HTTP-Methoden und Media-Types werden durch Responses mit den HTTP-Statuscodes 405 beziehungsweise 406 beantwortet. Im nächsten Schritt wird geprüft der über den Parameter *processID* spezifizierte Prozess überhaupt in der Anwendung registriert ist. Ist dies nicht der Fall wird als Response eine Exception (siehe Anhang C.17) mit dem HTTP-Statuscode 200 und einem *resource*-Header mit dem Wert *no-such-process* versandt. Existiert die spezifizierte *processID* wird die korrespondierende Prozess Beschreibung geladen. Um ein Response-Objekt zu generieren wird je nach spezifiziertem Media-Type werden entweder dem Renderer die *process.html* zusammen mit der geladenen Prozess Beschreibung übergeben oder direkt aus dieser ein solches generiert. Das Response-Objekt wird zusammen mit dem HTTP-Statuscode 200 sowie den *resource* und *link* Headern versandt. Der *resource*-Header enthält den Wert *process - <processID>*. Fehler werden durch Responses mit dem HTTP-Statuscode 500 abgefangen siehe Anhang 5 und B.6 [14].

3.6.6 Process Execution Endpoint

Die Requirements-Class Core definiert auch die Funktionalität zum Ausführen der angebotenen Prozesse. Um Prozesse auszuführen können Requests mit der HTTP-Methode Post an den URL *http://HOST:PORT/processes/<processID>/execution* gestellt werden. Dabei müssen die in der Prozess Beschreibung aufgeführten Inputs sowie die gewünschten Outputs im Request enthalten sein. Diese werden zunächst verifiziert. Ein erfolgreicher Request resultiert in der Erstellung eines Jobs.

?Wie läuft die generierung des Response ab und was enthält dieser?

3.6.7 Job List Endpoint

Ein Aufruf des URL *http://HOST:PORT/jobs?f=<MEDIA-TYPE>* liefert als Ressource eine Liste aller angelegten Jobs (siehe Anhang C.14). Die Job Liste kann nur mit der HTTP-Get Methode abgefragt werden. Diese kann mit zusätzlichen Parametern eingeschränkt werden. Analog zur Prozess Liste kann die Job Liste im Media-Type *text/html* (siehe Anhang D.9) und *application/json* angefragt werden.

?Wie läuft die generierung des Response ab und was enthält dieser?

3.6.8 Job Endpoint

Die Requirements-Class Core definiert auch den Job Endpoint. Requests an den Job Endpoint können an den URL *http://HOST:PORT/jobs/<jobID>?f=<MEDIA-TYPE>* gestellt werden. Welcher Job Status zurückgegeben werden soll wird über den *jobID*-Parameter spezifiziert. Die im Response enthaltenen Ressource ist der Job Status (siehe Anhang C.15). Der Job Status kann nur mit der HTTP-Get Methode abgerufen werden. Dieser kann unter anderem entnommen werden ob ein und wann ein Job gestartet wurde, in welches Bearbeitungsschritt er sich befindet oder ob der Job bereits erfolgreich abgeschlossen oder gescheitert ist. Die Job Status Ressource steht als *text/html* (siehe Anhang D.10) und *application/json* zur Verfügung.

Wird im Request eine vorhandene *jobID* angefragt wird die *status.json* aus dem *jobs/<jobID>/* Verzeichnis geladen und je nach gewünschtem Media-Type ein Response generiert. Wurde die Ressource im Media-Type *text/html* angefragt wird ein HTML-Template zusammen mit der geladenen *status.json* dem Renderer übergeben. Ein Response im Media-Type *application/json* wird um die Verknüpfungen ergänzt und versandt.

Soll ein Job abgebrochen werden kann der Job Endpoint mit der HTTP-Delete Methode aufgerufen werden. In diesem Fall wird der Job-Status des Jobs auf *dismisses* gesetzt. Dies sorgt für den Abbruch der Bearbeitung.

3.6.9 Job Results Endpoint

Die Ergebnisse eines erfolgreichen Jobs können unter dem URL *http://HOST:PORT/jobs/<jobID>//results?f=TYPE>* abgerufen werden. Die bereitgestellten Ressourcen variieren je nach Prozess, gewähltem Output sowie dessen Transmission-Mode, *value* oder *reference* und Response-Type *document* oder *raw*. Sie können mit der HTTP-Get Methode abgerufen werden.

Wird ein Request akzeptiert wird zunächst geprüft ob die Job-ID überhaupt existiert und der Job bereits den Job-Status *successful* aufweist. Es folgt die Generierung eines Result-Dokuments oder eines Download-Links. !Pseudode!

3.7 Domumentation

Für die Dokumentation der API soll der OpenAPI 3.0 Standard genutzt werden [14]. Dieser definiert eine sprachenunabhängige Schnittstellenbeschreibung für Web APIs [20]. Diese Schnittstellenbeschreibung ist dabei so formuliert das sie sowohl menschen- als auch maschinenlesbar ist. Ziel dieser Dokumentationsweise ist es die API so leicht verständlich wie möglich zu machen und so die Benutzerfreundlichkeit zu steigern [20].

Die Dokumentation soll sämtliche Endpoints mit den möglichen Responses und zu erwartenden HTTP-Statuscodes beschreiben. Die Dokumentation der API der prototypischen Implementierung auf der Web-Seit SwaggerHub gepflegt. Die Dokumentation konnte von dort als HTML- und JSON-Datei bezogen werden. Diese dienen als Repräsentationen der Ressource API Definition welche über den API Definition Endpoint angefragt werden kann.

3.8 Prozesse

3.8.1 Echo

Da eine standardkonforme API mindestens einen testbaren Prozess anbieten muss ist der Echo-Prozess ebenfalls Teil der prototypischen Implementierung [14]. Dieser nimmt einen beliebigen Wert entgegen. Nach einer kurzen, simulierten Bearbeitungszeit kann dieser wieder als Resultat abgefragt werden.

Nach dem Start eines Echo Prozesses wird zunächst überprüft ob der Job nicht den Status *dismissed* aufweist. Wäre dies der Fall wird die Ausführung abgebrochen. Andernfalls wird der *started*-Eintrag in der status.json mit dem aktuellen Zeitstempel versehen und der zurückzugebende Wert aus den Eingaben des Jobs gelesen. Schlägt dies fehl wird der *status*-Eintrag in der status.json auf *failed* gesetzt und der Ausführung abgebrochen. Anschließend wartet das Programm fünf Sekunden. Nach einer erneuten Prüfung des Job-Status wird das Ergebnis als .json in das *results/-*Verzeichnis des Jobs geschrieben. Diese results.json enthält den Eingabewert und die Nachricht das es sich um ein Echo handelt. Als letzter Schritt wird der Job-Status, der Fortschritt, der Infotext sowie der Beendigungszeitpunkt in der Status-Datei des Jobs aktualisiert B.12.

3.8.2 Überflutungsmonitoring

3.9 Zusätzliche Funktionalitäten

3.9.1 Coverage

Der nicht im Standard definierte Coverage Endpoint kann über den URL *http://HOST:PORT/coverage?f=<META-INFO>* mit der HTTP-Get Methode erreicht werden. Als Ressource liefert er eine Liste aller Sentinel-1 Datensätze welche persistent gespeichert sind. Jobs welche auf diese Datensätze zugreifen können schneller abgearbeitet werden da ein zeitaufwendiges Herunterladen der Datensätze entfällt. Die Coverage Ressource kann nur mit der HTTP-Get Methode angefragt werden. Diese werden mit ihrem Produktnamen und dem Datum der Aufnahme und ihrer Bounding-BBox aufgeführt. Die Bounding-BBox gibt Aufschluss über die räumliche Ausdehnung des Datensatzes. Die Ressource kann in den Media-Types *text/html* (siehe Anhang D.11) oder *application/json* angefragt werden.

3.9.2 Test Suit

Um die Funktionen der API und seine Standardkonformität testen zu können wurde ein Unit-Test oder test Suit aus dem im Standard beschriebenen Abstract Test Suit abgeleitet und implementiert. Zu bemerken ist dabei das der prototypische implementierte Test-Suit lediglich einige maschinen-testbare Testfälle abdeckt. Werden nun Änderungen am Programm vorgenommen kann mit der Ausführung dieses Test-Suits einfach die Funktionalität und Stabilität der API überprüft werden.

4 Evaluation

5 Diskussion

6 Ausblick

Docker Reproducibility Andere Datenquellen Andere Schwellwertverfahren Evalation der technischen Eigenschaften

7 Fazit

Literatur

- [1] A. Moreira, M. Younis, P. Prats-Iraola, G. Krieger, I. Hajnsek und K. P. Papathanassiou (2013, April 17). A Tutorial on Synthetic Aperture Radar [Online]. Verfügbar unter: https://www.researchgate.net/publication/257008464_A_Tutorial_on_Synthetic_Aperture_Radar (Zugriff am: 6. Juni 2022).
- [2] J. Albertz, Einführung in die Fernerkundung, 4. Auflage Darmstadt: Wissenschaftliche Buchgesellschaft, 2009
- [3] Europäische Kommission (2018, Oktober 06). Copernicus: 20 years of History [Online]. Verfügbar unter: <https://www.copernicus.eu/en/documentation/information-material/signature-esafrance-collaborative-ground-segment> (Zugriff am: 13. Juni 2022).
- [4] European Space Agency (2018). Sentinels - Space for Copernicus [Online]. Verfügbar unter: <https://www.d-copernicus.de/daten/satelliten/daten-sentinels/> (Zugriff am: 13. Juni 2022).
- [5] Europäische Kommission (2019). What is Copernicus [Online]. Verfügbar unter: <https://www.copernicus.eu/en/documentation/information-material/brochuresbrochures> (Zugriff am: 13. Juni 2022).
- [6] ESA Communications (2012, März). Sentinel-1 ESA's Radar Observatory Mission for GMES Operational Services [Online]. Verfügbar unter: <https://sentinel.esa.int/web/sentinel/missions/sentinel-1/overview> (Zugriff am: 13. Juni 2022).
- [7] Europäisches Parlament und Rat der Europäischen Union (2014, April 24). Regulation (EU) No 377/2014 Establishing the Copernicus Programme and repealing Regulation (EU) No 911/2010 [Online]. Verfügbar unter: <https://www.kowi.de/Portaldata/2/Resources/horizon2020/coop/Copernicus-regulation.pdf> (Zugriff am: 13. Juni 2022).
- [8] M. Bourbigot, H. Johnson, R. Piantanida (2016, März 03). Sentinel-1 Product Definition [Online]. Verfügbar unter: https://sentinel.esa.int/web/sentinel/user-guides/sentinel-1-sar/document-library/-/asset_publisher/1dO7RF5fJMbd/content/sentinel-1-product-definition (Zugriff am: 13. Juni 2022).
- [9] N. Otsu (1976, Januar). A Threshold Selection Method from Gray-Level Histograms [Online]. Verfügbar unter: <https://ieeexplore.ieee.org/document/4310076/citations#citations> (Zugriff am: 14. Juni 2022).
- [10] A. McVittie (2019, Februar). Sentinel-1 Flood mapping tutorial [Online]. Verfügbar unter: <https://step.esa.int/main/doc/tutorials/> (Zugriff am: 15. Juni 2022).

- [11] N. Miranda und P.J. Meadows (2015, Mai 21). Radiometric Calibration of S-1 Level-1 Products Generated by the S-1 IPF [Online]. Verfügbar unter: https://sentinel.esa.int/web/sentinel/user-guides/document-library/-/asset_publisher/xlslt4309D5h/content/sentinel-1-radiometric-calibration-of-products-generated-by-the-s1-ipf (Zugriff am: 15. Juni 2022).
- [12] N. I. Ulloa, S.-H. Chiang und S.-H. Yun (2020, April 27). Flood Proxy Mapping with Normalized Difference Sigma-Naught Index and Shannon's Entropy [Online]. Verfügbar unter: <https://doi.org/10.3390/rs12091384> (Zugriff am: 21. Juni 2022).
- [13] Europäische Kommission (2018, Juni). The DIAS: User-friendly Access to Copernicus Data and Information [Online]. Verfügbar unter: <https://www.copernicus.eu/en/access-data/dias> (Zugriff am: 24. Juni 2022)
- [14] B. Pross und P. A. Vretanos. (2021, Dezember 20). OGC API – Processes – Part 1: Core [Online]. Verfügbar unter: <https://docs.opengeospatial.org/is/18-062r2/18-062r2.html> (Zugriff am: 24. Juni 2022).
- [15] S. Cox, D. Danko, J. Greenwood, J.R. Herring, A. Matheus, R. Pearsall, C. Portele, B. Reff, P. Scarponcini, A. Whiteside (2009, Oktober 19). The Specification Model - A Standard for Modular specifications [Online]. Verfügbar unter: <https://www.ogc.org/standards/modularspec> (Zugriff am: 27. Juni 2022).
- [16] Open Geospatial Consortium (2021, Dezember 16). Bylaws of Open Geospatial Consortium [Online]. Verfügbar unter: <https://www.ogc.org/ogc/policies> (Zugriff am: 27. Juni 2022).
- [17] C. Holmes, D. WWesloh, C. Heazel, G. Gale, A. Christl, J. Lieberman, C. Reed, J. Herring, M. Desruisseaux, D. Blodgett, S. Simmons, B. de Lathower und G. Percivall (2017, Februar 23). OGC Open Geospatial APIs - White Paper [Online]. Verfügbar unter: <https://docs.ogc.org/wp/16-019r4/16-019r4.html> (Zugriff am: 03. Juli 2022).
- [18] F. Houbie, S. Sankaran, J. Lieberman, P. Vretanos, J. Masó (2016, Januar 16). OGC Testbed 11 REST Interface Engineering Report [Online]. Verfügbar unter: <https://www.ogc.org/docs/er> (Zugriff am: 05. Juli 2022).
- [19] J. Nielsen, Usability Engineering, Mountain View: Academic Press Inc., 1993
- [20] D. Miller, J. Whitlock, M. Gardiner, M. Ralphson, R. Ratovsky, U. Sarid (2021, February 15). OpenAPI Specification v3.1.0 [Online]. Verfügbar unter: <https://spec.openapis.org/oas/v3.0.1> (Zugriff am: 07. Juli 2022).
- [21] M. Biehl, API Design, 1. Auflage Suurstoffi: API-University Press, 2016

A Pseudocode

A.1 Ablauf eines Requests an den API Landing Page Endpoint

Programmablauf 1 Ablauf eines Requests an den API Landing Page Endpoint

```
HTTP-Methode ← Request.HTTP-Method
Encoding ← Request.f
if HTTP-Methode != GET then
    return HTTP-Statuscode 405
else
    try
        if Encoding == text/html or Encoding == None then
            Response ← render_template(templates/html/landingPage.html)
            Link-Header ← http://HOST:PORT/?f=text/html
            Resource-Header ← landingPage
            return HTTP-Statuscode 200 and Response with Link- and Resource-Header
        else if Encoding == application/json then
            JSON ← open(templates/json/landingPage.json)
            Response ← jsonify(Datei)
            Link-Header ← http://HOST:PORT/?f=application/json
            Resource-Header ← landingPage
            return HTTP-Statuscode 200 and Response with Link- and Resource-Header
        else
            return HTTP-Statuscode 406
        end if
    except
        return HTTP-Statuscode 500
end if
```

A.2 Ablauf eines Requests an den API Definition Endpoint

Programmablauf 2 Ablauf eines Requests an den API Definition Endpoint

```
HTTP-Methode ← Request.HTTP-Method
Encoding ← Request.f
if HTTP-Methode != GET then
    return HTTP-Statuscode 405
else
    try
        if Encoding == text/html or Encoding == None then
            Response ← render_template(templates/html/apiDefinition.html)
            Link-Header ← http://HOST:PORT/api?f=text/html
            Resource-Header ← apiDefinition
            return HTTP-Statuscode 200 and Response with Link- and Resource-Header
        else if Encoding == application/json then
            JSON ← open(templates/json/apiDefinition.json)
            Response ← jsonify(Datei)
            Link-Header ← http://HOST:PORT/api?f=application/json
            Resource-Header ← apiDefinition
            return HTTP-Statuscode 200 and Response with Link- and Resource-Header
        else
            return HTTP-Statuscode 406
        end if
    except
        return HTTP-Statuscode 500
end if
```

A.3 Ablauf eines Requests an den Conformance Endpoint

Programmablauf 3 Ablauf eines Requests an den Conformance Endpoint

```
HTTP-Methode ← Request.HTTP-Method
Encoding ← Request.f
if HTTP-Methode != GET then
    return HTTP-Statuscode 405
else
    try
        if Encoding == text/html or Encoding == None then
            Response ← render_template(templates/html/confClasses.html)
            Link-Header ← http://HOST:PORT/conformance?f=text/html
            Resource-Header ← conformance
            return HTTP-Statuscode 200 and Response with Link- and Resource-Header
        else if Encoding == application/json then
            JSON ← open(templates/json/confClasses.json)
            Response ← jsonify(Datei)
            Link-Header ← http://HOST:PORT/conformance?f=application/json
            Resource-Header ← conformance
            return HTTP-Statuscode 200 and Response with Link- and Resource-Header
        else
            return HTTP-Statuscode 406
        end if
    except
        return HTTP-Statuscode 500
end if
```

A.4 Ablauf eines Requests an den Process List Endpoint

Programmablauf 4 Ablauf eines Requests an den Process List Endpoint

```
HTTP-Methode ← Request.HTTP-Method
Encoding ← Request.f
Limit ← Request.limit
if HTTP-Methode != GET then
    return HTTP-Statuscode 405
else
    try
        if Limit == None or Limit <= 0 or Limit > 100000 then
            Limit ← 10
        else
            Limit ← Anfrage.limit
        end if
        if Encoding == text/html or Encoding == None then
            Process List ← []
            Processes ← Process Descriptions in templates/json/processes
            for Process in Processes do
                JSON ← open(jobs/Job-ID/status.json)
                Process List append JSON
            end for
            Response ← render_template(templates/html/processList.html, Process List[0:Limit])
            Link-Header ← http://HOST:PORT/processList?f=application/json
            Resource-Header ← processList
            return HTTP-Statuscode 200 and Response with Link- and Resource-Header
        else if Encoding == application/json then
            Process List ← []
            Processes ← List of Process descriptions in templates/json/processes
            for Process in Processes do
                JSON ← open(jobs/Job-ID/status.json)
                Process Liste append JSON
            end for
            add Links to self and alternate to Process Liste
            Response ← jsonify(Process Liste[0:Limit])
            Link-Header ← http://HOST:PORT/processList?f=application/json
            Resource-Header ← processList
            return HTTP-Statuscode 200 and Response with Link- and Resource-Header
        else
            return HTTP-Statuscode 406
        end if
    except
        return HTTP-Statuscode 500
end if
```

A.5 Ablauf eines Requests an den Process Description Endpoint

Programmablauf 5 Ablauf eines Requests an den Process Description Endpoint

```
HTTP-Methode ← Request.HTTP-Method
Encoding ← Request.f
Process-ID ← Request.processID
if HTTP-Methode != GET then
    return HTTP-Statuscode 405
else
    try
        if Encoding == text/html or Encoding == None then
            if templates/json/processes/<Process-ID>ProcessDescription.json exists then
                JSON ← open(templates/json/processes/<Process-ID>ProcessDescription.json)
                Response ← render_template(templates/html/process.html, JSON)
                Link-Header ← http://HOST:PORT/processes/<processID>?f=text/html
                Resource-Header ← process - Process-ID
                return HTTP-Statuscode 200 and Response with Link- and Resource-Header
            else
                Exception ← No such process exception
                Resource-Header ← no-such-process
                return HTTP-Statuscode 404 and Exception with Resource-Header
            end if
        else if Encoding == application/json then
            if templates/json/processes/<Process-ID>ProcessDescription.json exists then
                JSON ← open(templates/json/processes/<Process-ID>ProcessDescription.json)
                Response ← jsonify(JSON)
                Link-Header ← http://HOST:PORT/processes/<Process-ID>?f=application/json
                Resource-Header ← process - Process-ID
                return HTTP-Statuscode 200 and Response with Link- and Resource-Header
            else
                Exception ← No such process exception
                Resource-Header ← no-such-process
                return HTTP-Statuscode 404 and Exception with Resource-Header
            end if
        else
            return HTTP-Statuscode 406
        end if
    except
        return HTTP-Statuscode 500
end if
```

A.6 Ablauf eines Requests an den Process Execution Endpoint

Programmablauf 6 Ablauf eines Requests an den Process Execution Endpoint

```
HTTP-Methode ← Anfrage.HTTP-Methode
```

A.7 Ablauf eines Requests an den Job List Endpoint

Programmablauf 7 Ablauf eines Requests an den Job List Endpoint

```
HTTP-Methode ← Anfrage.HTTP-Methode
```

A.8 Ablauf eines Requests an den Job Endpoint

Programmablauf 8 Ablauf eines Requests an den Job Status Endpoint

```
HTTP-Methode ← Anfrage.HTTP-Method
Encoding ← Request.f
Job-ID ← Request.jobID
if HTTP-Methode == GET then
  try
    if Encoding == text/html or Encoding == None then
      if jobs/<Job-ID>/status.json exists then
        JSON ← open(jobs/<Job-ID>/status.json)
        Response ← render_template(templates/html/job.html, JSON)
        Link-Header ← http://HOST:PORT/jobs/<Job-ID>?f=text/html
        Resource-Header ← job - <Job-ID>
        return HTTP-Statuscode 200 and Response with Link- and Resource-Header
      else
        Exception ← No such job exception
        Resource-Header ← no-such-job
        return HTTP-Statuscode 404 and Exception with Resource-Header
      end if
    else if Encoding == application/json then
      if jobs/<Job-ID>/status.json exists then
        Job ← open(jobs/<Job-ID>/status.json)
        add Links to self and alternate to Job
        Response ← jsonify(Job)
        Link-Header ← http://HOST:PORT/jobs/<Job-ID>?f=application/json
        Resource-Header ← job - <Job-ID>
        return HTTP-Statuscode 200 and Response with Link- and Resource-Header
      else
        Exception ← No such job exception
        Resource-Header ← no-such-job
        return HTTP-Statuscode 404 and Exception with Resource-Header
      end if
    else
      return HTTP-Statuscode 406
    end if
  except
    return HTTP-Statuscode 500
else if HTTP-Methode == DELETE then
  try
    if jobs/<Job-ID>/status.json exists then
      Job ← open(jobs/<Job-ID>/status.json)
      if Job.status != dismissed then
        Job ← open(jobs/<Job-ID>/status.json)
        if Encoding == text/html or Encoding == None then
          Response ← render_template(templates/html/job.html)
          Link-Header ← http://HOST:PORT/jobs/<Job-ID>?f=text/html
          Resource-Header ← job-dismissed
          return HTTP-Statuscode 200 and Response with Link- and Resource-Header
        else if Encoding == application/json then
          Response ← jsonify(Job)
          Link-Header ← http://HOST:PORT/jobs/<Job-ID>?f=application/json
          Resource-Header ← job-dismissed
          return HTTP-Statuscode 200 and Response with Link- and Resource-Header
        else
          return HTTP-Statuscode 406
        end if
      else
        return HTTP-Statuscode 406
      end if
    else
      if Encoding == text/html or Encoding == None then
        Response ← render_template(templates/html/job.html)
        Link-Header ← http://HOST:PORT/jobs/<Job-ID>?f=text/html
        Resource-Header ← job-dismissed
        return HTTP-Statuscode 410 and Response with Link- and Resource-Header
      else if Encoding == application/json then
        Response ← jsonify(Job)
        Link-Header ← http://HOST:PORT/jobs/<Job-ID>?f=application/json
        Resource-Header ← job-dismissed
        return HTTP-Statuscode 410 and Response with Link- and Resource-Header
      else
        return HTTP-Statuscode 406
      end if
    end if
  except
    return HTTP-Statuscode 500
```

A.9 Ablauf eines Requests an den Job Results Endpoint

Programmablauf 9 Ablauf eines Requests an den Job Results Endpoint

```
HTTP-Methode ← Anfrage.HTTP-Methode
```

A.10 Ablauf eines Requests an den Coverage Endpoint

Programmablauf 10 Ablauf eines Requests an den Coverage Endpoint

```
HTTP-Methode ← Request.HTTP-Methode
Encoding ← Request.f
if HTTP-Methode != GET then
    return HTTP-Statuscode 405
else
    try
        KML ← KML-Files in data/coverage/
        Coverages ← []
        BBoxes ← []
        for File in KML do
            Data ← open(File)
            Dataset ← Data.Name, Data.BBox, Data.Date
            BBox ← Data.BBox
            Coverages append Dataset
            BBoxes append BBox as .geojson
        end for
        if Encoding == text/html or Encoding == None then
            x
        else if Encoding == application/json then
            y
        else
            return HTTP-Statuscode 406
        end if
    except
        return HTTP-Statuscode 500
end if
```

B Quellcodeverzeichnis

B.1 Konfiguration von Werkzeug auf HTTP 1.1

Quellcode B.1: Konfiguration von Werkzeug auf HTTP 1.1

```
1 from flask import Flask
2 from werkzeug.serving import WSGIRequestHandler
3 from werkzeug.serving import BaseWSGIServer
4 WSGIRequestHandler.protocol_version = "HTTP/1.1"
5 BaseWSGIServer.protocol_version = "HTTP/1.1"
```

B.2 Quellcode Landing Page Endpoint

Quellcode B.2: Landing Page Endpoint

```
1 #landingpage endpoint
2 @app.route('/', methods = ['GET'])
3 def getLandingPage():
4     app.logger.info('/')
5     try:
6         if(request.content_type == "text/html" or
7            request.args.get('f')== "text/html" or
8            request.args.get('f') == None):
9             response = render_template('html/landingPage.html')
10             return response, 200, {
11                 "link": "localhost:5000/?f=text/html",
12                 "resource": "landingPage"
13             }
14         elif(request.content_type == "application/json" or
15              request.args.get('f')== "application/json"):
16             file = open('templates/json/landingPage.json',)
17             payload = json.load( file)
18             file.close()
19             response = jsonify(payload)
20             return response, 200, {
21                 "link": "localhost:5000/?f=application/json",
22                 "resource": "landingPage"}
23         else:
24             return "HTTP status code 406: not acceptable", 406
25     except:
26         return "HTTP status code 500: internal server error", 500
```

B.3 Quellcode API Definition Endpoint

Quellcode B.3: API Definition Endpoint

```
1 #api endpoint
2 @app.route('/api', methods = ['GET'])
3 def getAPIDefinition():
4     app.logger.info('/api')
5     try:
6         if(request.content_type == "text/html" or
7            request.args.get('f')== "text/html" or
8            request.args.get('f') == None):
9             response = render_template('html/apiDefinition.html')
10            return response, 200, {
11                "link": "localhost:5000/apiDefinition?f=text/html",
12                "resource": "apiDefinition"}
13        elif(request.content_type == "application/json" or
14             request.args.get('f')== "application/json"):
15            file = open('templates/json/apiDefinition.json',)
16            payload = json.load( file)
17            file.close() #close apiDefinition.json
18            response = jsonify(payload)
19            return response, 200, {
20                "link": "localhost:5000/api?f=application/json",
21                "resource": "apiDefinition"}
22        else:
23            return "HTTP status code 406: not acceptable", 406
24    except:
25        return "HTTP status code 500: internal server error", 500
```

B.4 Quellcode Conformance Endpoint

Quellcode B.4: Conformance Endpoint

```
1 #conformance endpoint
2 @app.route('/conformance', methods = ['GET'])
3 def getConformance():
4     app.logger.info('/conformance')
5     try:
6         if(request.content_type == "text/html" or
7            request.args.get('f')== "text/html" or
8            request.args.get('f') == None):
9             response = render_template('html/confClasses.html')
10            return response, 200, {
11                "link": "localhost:5000/conformance?f=text/html",
12                "resource": "conformance"}
13        elif(request.content_type == "application/json" or
14             request.args.get('f')== "application/json"):
15            file = open('templates/json/confClasses.json',)
16            payload = json.load( file)
17            file.close()
18            response = jsonify(payload)
19            return response, 200, {
20                "link": "localhost:5000/conformance?f=application/json",
21                "resource": "conformance"}
22        else:
23            return "HTTP status code 406: not acceptable", 406
24    except:
25        return "HTTP status code 500: internal server error", 500
```

B.5 Quellcode Process List Endpoint

Quellcode B.5: Process List Endpoint

```
1 #processes endpoint
2 @app.route('/processes', methods = ['GET'])
3 def getProcesses():
4     app.logger.info('/processes')
5     if(request.args.get('limit') == None or
6         int(request.args.get('limit')) <= 0 or
7         int(request.args.get('limit')) > 1000):
8         limit = 10 #set limit to default value
9     else:
10        limit = int(request.args.get('limit'))
11    try:
12        if(request.content_type == "text/html" or
13            request.args.get('f')=="text/html" or
14            request.args.get('f') == None):
15            processList = [] #initialize list of processes
16            processDescriptions = os.listdir("templates/json/processes")
17            counter = 0
18            for i in processDescriptions:
19                file = open('templates/json/processes/' + i,)
20                process = json.load( file)
21                file.close()
22                processList.append(process)
23                counter += 1
24                if(counter == limit):
25                    break
26            response = render_template('html/processes.html',
27                                     processes=processList)
28            return response, 200, {
29                "link": "localhost:5000/processes?f=text/html",
30                "resource": "processes"}
31        elif(request.content_type == "application/json" or
32             request.args.get('f')=="application/json"):
33            processList = [] #initialize list of processes
34            processDescriptions = os.listdir("templates/json/processes")
35            for i in processDescriptions:
36                file = open('templates/json/processes/' + i,)
37                process = json.load( file)
38                file.close()
39                processList.append(process)
40            processes = {"processes": processList[0:limit],
41                        "links": [ #add links to self and alternate
42                            {
43                                "href": "localhost:5000/processes?f=applicattion/json",
44                                "rel": "self",
45                                "type": "application/json"
46                            },
47                            {
48                                "href": "localhost:5000/processes?f=text/html",
49                                "rel": "alternate",
50                                "type": "text/html"
51                            }
52                        ]}
53            response = jsonify(processes)
54            return response, 200, {
55                "link": "localhost:5000/processes?f=application/json",
```

```
56         "resource": "processes"}
57     else:
58         return "HTTP status code 406: not acceptable", 406
59 except:
60     return "HTTP status code 500: internal server error", 500
```


B.6 Quellcode Process Description Endpoint

Quellcode B.6: Process Description Endpoint

```
1 #process endpoint
2 @app.route('/processes/<processID>', methods = ['GET'])
3 def getProcess(processID):
4     app.logger.info('/processes/' + processID)
5     try:
6         if(request.content_type == "text/html" or
7            request.args.get('f')=="text/html" or
8            request.args.get('f') == None):
9             if(os.path.exists('templates/json/processes/'
10 + str(processID) + 'ProcessDescription.json')):
11                 file = open('templates/json/processes/'
12 + str(processID)
13 + 'ProcessDescription.json',)
14                 process = json.load( file)
15                 file.close()
16                 response = render_template("html/Process.html", process=process)
17                 return response, 200, {"link": "localhost:5000/processes/"
18 + str(processID)
19 + "?f=text/html",
20 "resource": str(processID)}
21             else:
22                 exception = render_template('html/exception.html',
23                 title="No such process exception",
24                 description="Requested process could not be found",
25                 type="no-such-process")
26                 return exception, 404, {"resource": "no-such-process"}
27         elif(request.content_type == "application/json" or
28              request.args.get('f')=="application/json"):
29             if(os.path.exists('templates/json/processes/'
30 + str(processID)
31 + 'ProcessDescription.json')):
32                 file = open('templates/json/processes/'
33 + str(processID)
34 + 'ProcessDescription.json',)
35                 payload = json.load( file)
36                 file.close()
37                 response = jsonify(payload)
38                 return response, 200, {"link": "localhost:5000/processes/"
39 + str(processID)
40 + "?f=application/json",
41 "resource": str(processID)}
42             else:
43                 exception = {"title": "No such process exception",
44                 "description": "Requested process could not be found",
45                 "type": "no-such-process"}
46                 return exception, 404, {"resource": "no-such-process"}
47         else:
48             return "HTTP status code 406: not acceptable", 406
49     except:
50         return "HTTP status code 500: internal server error", 500
```

B.7 Quellcode Process Execution Endpoint

Quellcode B.7: Process Execution

```
1 @app.route('/processes/<processID>/execution', methods = ['POST'])
2 def executeProcess(processID):
3     app.logger.info('/processes/' + processID + '/execution')
4     try:
5         if(os.path.exists('templates/json/processes/'
6             + str(processID)
7             + 'ProcessDescription.json')):
8             data = json.loads(request.data.decode('utf8').replace("'", ''))
9             inputParameters = utils.parseInput(processID, data)
10            if(inputParameters == False):
11                return "HTTP status code 400: bad request", 400
12            jobID = str(uuid.uuid4())
13            created = str(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"))
14            os.mkdir("jobs/" + jobID)
15            os.mkdir("jobs/" + jobID + "/results/")
16
17            job_file = {"jobID": str(jobID),
18                "processID": str(processID),
19                "input": inputParameters[0],
20                "output": inputParameters[2],
21                "responseType": inputParameters[1],
22                "path": "jobs/" + jobID,
23                "results": "jobs/" + jobID + "/results/",
24                "downloads": "jobs/" + jobID + "/downloads/"}
25            json.dumps(job_file, indent=4)
26            with open("jobs/" + jobID + "/job.json", 'w') as f:
27                json.dump(job_file, f)
28            f.close()
29
30            status_file = {"jobID": str(jobID),
31                "processID": str(processID),
32                "status": "accepted",
33                "message": "Step 0/1",
34                "type": "process",
35                "progress": 0,
36                "created": created,
37                "started": "none",
38                "finished": "none",
39                "links": [
40                    {
41                        "href": "localhost:5000/jobs/"
42                        + jobID + "?f=application/json",
43                        "rel": "self",
44                        "type": "application/json",
45                        "title": "this document as JSON"},
46                    {
47                        "href": "localhost:5000/jobs/"
48                        + jobID + "?f=text/html",
49                        "rel": "alternate",
50                        "type": "text/html",
51                        "title": "this document as HTML"}
52                ]
53            }
54            json.dumps(status_file, indent=4) #dump content
55            with open("jobs/" + jobID + "/status.json", 'w') as f: #create file
```

```

56         json.dump(status_file, f) #write content
57     f.close() #close file
58
59     response = jsonify(status_file) #create response
60     return response, 201, {"location": "localhost:5000/jobs/"
61 + jobID + "?f=application/json",
62     "resource": "job"}
63     exception = {"title": "No such process exception",
64     "description": "Requested process could not be found",
65     "type": "no-such-process"}
66     return exception, 404
67 except:
68     return "HTTP status code 500: internal server error", 500

```

B.8 Quellcode Job List Endpoint

Quellcode B.8: Job List Endpoint

```
1 #job endpoint for status and dismiss
2 @app.route('/jobs/<jobID>', methods = ['GET', 'DELETE'])
3 def getJob(jobID):
```

B.9 Quellcode Job Endpoint

Quellcode B.9: Job Endpoint

```
1 #job endpoint for status and dismiss
2 @app.route('/jobs/<jobID>', methods = ['GET', 'DELETE'])
3 def getJob(jobID):
4     if(request.method == 'GET'):
5         app.logger.info('[GET] /jobs/' + jobID)
6         try:
7             if(request.content_type == "application/json" or
8                 request.args.get('f')=="application/json"):
9                 if(os.path.exists('/jobs/' + str(jobID) + '/status.json')):
10                     file = open('/jobs/' + str(jobID) + '/status.json')
11                     data = json.load( file)
12                     file.close()
13                     response = jsonify(data)
14                     return response, 200, {"link": "localhost:5000/jobs/"
15 + str(jobID)
16 + "?f=application/json",
17 "resource": "job - "
18 + str(jobID)}
19             else:
20                 exception = {"title": "No such job exception",
21 "description": "No job with the requested jobID could be found",
22 "type": "no-such-job"}
23                 return exception, 404, {"resource": "no-such-job"}
24             elif(request.content_type == "text/html" or
25                  request.args.get('f')=="text/html" or
26                  request.args.get('f') == None):
27                 if(os.path.exists('/jobs/' + str(jobID) + '/status.json')):
28                     file = open('/jobs/' + str(jobID) + '/status.json')
29                     job = json.load( file) #create response
30                     file.close() #close status.json
31                     response = render_template("html/Job.html", job=job)
32                     return response, 200, {"link": "localhost:5000/jobs/" + str(jobID)
33 + "?f=text/html",
34 "resource": "job - "
35 + str(jobID)}
36             else:
37                 exception = render_template('html/exception.html', title="No such job exception", d
38 type="no-such-job")
39                 return exception, 404, {"resource": "no-such-job"} #return not found if requested j
40             else:
41                 return "HTTP status code 406: not acceptable", 406 #return not acceptable if requested
42         except:
43             return "HTTP status code 500: internal server error", 500 #return internal server error if
44     if(request.method == 'DELETE'):
45         app.logger.info('[DELETE] /jobs/' + jobID) #add log entry when endpoint is called
46         try:
47             if(os.path.exists('/jobs/' + str(jobID) + '/status.json')): #check if jobID exists
48                 with open('/jobs/' + str(jobID) + '/status.json', "r") as f: #open status.json
49                     file = json.load(f) #load data from status.json
50                     if( file["status"] != "dismissed"): #if job is not dismissed
51                         file["status"] = "dismissed" #set status to dismissed
52                         file["message"] = "job dismissed" #set emssage to dismissed
53                     f.close() #close status.json
54                     with open('/jobs/' + str(jobID) + '/status.json', "w") as f: #write status.json
55                         json.dump( file, f) #dump content
```

```

56         f.close() #close status.json
57         file = open('jobs/' + str(jobID) + '/status.json') #open status.json
58         data = json.load( file) #load data from status.json
59         file.close() #close status.json
60
61         if(request.content_type == "text/html" or #check requested content-type from re
62             request.args.get('f')== "text/html" or
63             request.args.get('f') == None):
64             file = open('jobs/' + str(jobID) + '/status.json') #open status.json
65             job = json.load( file) #create response
66             file.close() #close status.json
67             response = render_template("html/Job.html", job=job) #render dynamic job
68             return response, 200, {"link": "localhost:5000/jobs/" + str(jobID) + "?f=t
69         else:
70             response = jsonify(data) #create response
71             return response, 200, {"link": "localhost:5000/jobs/" + str(jobID) + "?f=a
72         elif(request.content_type == "application/json" or #check requested content-type fr
73             request.args.get('f')== "application/json"): #check requested content-type f
74             data = json.load( file) #load data from status.json
75             file.close() #close status.json
76             response = jsonify(data) #create response
77             return response, 410, {"link": "localhost:5000/jobs/" + str(jobID) + "?f=appli
78     else:
79         exception = {"title": "No such job exception", "description": "No job with the requeste
80         return exception, 404 #return not found if requested job is not found
81 except:
82     return "HTTP status code 500: internal server error", 500 #return internal server error if

```

B.10 Quellcode Job Results Endpoint

Quellcode B.10: Job Results Endpoint

```
1 #job endpoint for status and dismiss
2 @app.route('/jobs/<jobID>', methods = ['GET', 'DELETE'])
3 def getJob(jobID):
```

B.11 Quellcode Coverage Endpoint

Quellcode B.11: Coverage Endpoint

```
1 #job endpoint for status and dismiss
2 @app.route('/jobs/<jobID>', methods = ['GET', 'DELETE'])
3 def getJob(jobID):
```


B.12 Quellcode Echo Process

Quellcode B.12: Echo Process

```
1 def echoProcess(job):
2     if(checkForDismissal(job.path + '/status.json') == True):
3         return
4
5     setStarted(job.path + '/status.json')
6
7     try:
8         input = job. input[0]
9         time.sleep(5)
10    except:
11        updateStatus(job.path + '/status.json', "failed", "The job has failed", "-")
12        return
13
14    if(checkForDismissal(job.path + '/status.json') == True):
15        return
16
17    result ={"result": input,
18            "message": "This is an echo"}
19    json.dumps(result, indent=4)
20    with open(job.results + "result.json", 'w') as f: #create file
21        json.dump(result, f) #write content
22        f.close() #close file
23    updateStatus(job.path + '/status.json', "successful", "Step 1 of 1 completed", "100")
24    setFinished(job.path + '/status.json')
```

C Schemata

C.1 landingPage.yaml

Schema C.1: landingPage.yaml

```
1 type: object
2 required:
3   - links
4 properties:
5   title:
6     type: string
7     example: Example processing server
8   description:
9     type: string
10    example: Example server
11  links:
12    type: array
13    items:
14      $ref: "link.yaml"
```

C.2 confClasses.yaml

Schema C.2: confClasses.yaml

```
1 type: object
2 required:
3   - conformsTo
4 properties:
5   conformsTo:
6     type: array
7     items:
8       type: string
9       example: "http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/core"
```

C.3 processList.yaml

Schema C.3: processList.yaml

```
1 type: object
2 required:
3   - processes
4   - links
5 properties:
6   processes:
7     type: array
8     items:
9       $ref: "processSummary.yaml"
10  links:
11    type: array
12    items:
13      $ref: "link.yaml"
```

C.4 limit.yaml

Schema C.4: limit.yaml

```
1 name: limit
2 in: query
3 required: false
4 schema:
5   type: integer
6   minimum: 1
7   maximum: 10000
8   default: 10
9 style: form
10 explode: false
```

C.5 link.yaml

Schema C.5: link.yaml

```
1 type: object
2 required:
3   - href
4 properties:
```

```

5  href:
6    type: string
7  rel:
8    type: string
9    example: service
10 type:
11   type: string
12   example: application/json
13 hreflang:
14   type: string
15   example: en
16 title:
17   type: string

```

C.6 processList.yaml

Schema C.6: processList.yaml

```

1  type: object
2  required:
3    - processes
4    - links
5  properties:
6    processes:
7      type: array
8      items:
9        $ref: "processSummary.yaml"
10   links:
11     type: array
12     items:
13       $ref: "link.yaml"

```

C.7 processSummary.yaml

Schema C.7: processSummary.yaml

```

1  allOf:
2    - $ref: "descriptionType.yaml"
3    - type: object
4      required:
5        - id
6        - version
7      properties:
8        id:
9          type: string
10       version:
11         type: string
12       jobControlOptions:
13         type: array
14         items:
15           $ref: "jobControlOptions.yaml"
16       outputTransmission:
17         type: array
18         items:

```

```

19         $ref: "transmissionMode.yaml"
20     links:
21         type: array
22         items:
23             $ref: "link.yaml"

```

C.8 jobControlOptions.yaml

Schema C.8: jobControlOptions.yaml

```

1 type: string
2 enum:
3     - sync-execute
4     - async-execute
5     - dismiss

```

C.9 transmissionMode.yaml

Schema C.9: transmissionMode.yaml

```

1 type: string
2 enum:
3     - value
4     - reference
5 default:
6     - value

```

C.10 process.yaml

Schema C.10: process.yaml

```

1 allOf:
2     - $ref: "processSummary.yaml"
3     - type: object
4       properties:
5         inputs:
6             additionalProperties:
7                 $ref: "inputDescription.yaml"
8         outputs:
9             additionalProperties:
10                 $ref: "outputDescription.yaml"

```

C.11 inputDescription.yaml

Schema C.11: inputDescription.yaml

```

1 allOf:
2     - $ref: "descriptionType.yaml"
3     - type: object
4       required:

```

```

5     - schema
6   properties:
7   minOccurs:
8     type: integer
9     default: 1
10  maxOccurs:
11    oneOf:
12    - type: integer
13      default: 1
14    - type: string
15      enum:
16        - "unbounded"
17  schema:
18    $ref: "schema.yaml"

```

C.12 outputDescription.yaml

Schema C.12: outputDescription.yaml

```

1 allOf:
2 - $ref: "descriptionType.yaml"
3 - type: object
4   required:
5   - schema
6   properties:
7   schema:
8     $ref: "schema.yaml"

```

C.13 description.yaml

Schema C.13: description.yaml

```

1 type: object
2 properties:
3   title:
4   type: string
5   description:
6   type: string
7   keywords:
8   type: array
9   items:
10    type: string
11  metadata:
12  type: array
13  items:
14    $ref: "metadata.yaml"
15  additionalParameters:
16  allOf:
17    - $ref: "metadata.yaml"
18    - type: object
19      properties:
20        parameters:
21        type: array
22        items:
23          $ref: "additionalParameter.yaml"

```

C.14 jobList.yaml

Schema C.14: jobList.yaml

```
1 type: object
2 required:
3   - jobs
4   - links
5 properties:
6   jobs:
7     type: array
8     items:
9       $ref: "statusInfo.yaml"
10  links:
11    type: array
12    items:
13      $ref: "link.yaml"
```

C.15 statusInfo.yaml

Schema C.15: statusInfo.yaml

```
1 type: object
2 required:
3   - jobID
4   - status
5   - type
6 properties:
7   processID:
8     type: string
9   type:
10    type: string
11    enum:
12      - process
13   jobID:
14     type: string
15   status:
16     $ref: "statusCode.yaml"
17   message:
18     type: string
19   created:
20     type: string
21     format: date-time
22   started:
23     type: string
24     format: date-time
25   finished:
26     type: string
27     format: date-time
28   updated:
29     type: string
30     format: date-time
31   progress:
32     type: integer
33     minimum: 0
34     maximum: 100
35   links:
```

```

36     type: array
37     items:
38       $ref: "link.yaml"

```

C.16 statusCode.yaml

Schema C.16: statusCode.yaml

```

1 type: string
2 nullable: false
3 enum:
4   - accepted
5   - running
6   - successful
7   - failed
8   - dismissed

```

C.17 exception.yaml

Schema C.17: exception.yaml

```

1 title: Exception Schema
2 description: JSON schema for exceptions based on RFC 7807
3 type: object
4 required:
5   - type
6 properties:
7   type:
8     type: string
9   title:
10    type: string
11   status:
12    type: integer
13   detail:
14    type: string
15   instance:
16    type: string
17 additionalProperties: true

```

D Ressourcen

D.1 landingPage.html

Ressource D.1: landingPage.html

```

1 <!DOCTYPE html>
2 <html>
3   <body>
4     <h1>links:</h1>
5     <p>

```

```

6      href:<a href="localhost:5000/?f=text/html">
7          localhost:5000/?f=text/html</a><br>
8      rel: self<br>
9      type: text/html<br>
10     title: This document as HTML
11 </p>
12 <p>
13     href:<a href="localhost:5000/?f=application/json">
14         localhost:5000/?f=application/json</a><br>
15     rel: alternate<br>
16     type: application/json<br>
17     title: This document as JSON
18 </p>
19 <p>
20     href:<a href="localhost:5000/api?f=application/json">
21         localhost:5000/apiDefinition?f=application/json</a><br>
22     rel: service-desc<br>
23     type: application/json<br>
24     title: API definition for this endpoint as JSON
25 </p>
26 <p>
27     href:<a href="localhost:5000/api?f=text/html">
28         localhost:5000/apiDefinition?f=text/html</a><br>
29     rel: service-desc<br>
30     type: text/html<br>
31     title: API definition for this endpoint as HTML
32 </p>
33 <p>
34     href:<a href="localhost:5000/conformance?f=application/json">
35         localhost:5000/conformance?f=application/json</a><br>
36     rel: conformance<br>
37     type: application/json<br>
38     title: OGC API - Processes conformance classes implemented by this server as JSON
39 </p>
40 <p>
41     href:<a href="localhost:5000/conformance?f=text/html">
42         localhost:5000/conformance?f=text/html</a><br>
43     rel: conformance<br>
44     type: text/html<br>
45     title: OGC API - Processes conformance classes implemented by this server as HTML
46 </p>
47 <p>
48     href:<a href="localhost:5000/processes?f=application/json">
49         localhost:5000/processes?f=application/json</a><br>
50     rel: processes<br>
51     type: application/json<br>
52     title: Metadata about the processes as JSON
53 </p>
54 <p>
55     href:<a href="localhost:5000/processes?f=text/html">
56         localhost:5000/processes?f=text/html</a><br>
57     rel: processes<br>
58     type: text/html,<br>
59     title: Metadata about the processes as HTML
60 </p>
61 <p>
62     href:<a href="localhost:5000/jobs?f=application/json">
63         localhost:5000/jobs?f=application/json</a><br>
64     rel: jobs<br>

```



```

65     type: application/json<br>
66     title: The endpoint for job monitoring as JSON
67 </p>
68 <p>
69     href:<a href="localhost:5000/jobs?f=text/html">
70         localhost:5000/jobs?f=text/html</a><br>
71     rel: jobs<br>
72     type: text/html<br>
73     title: The endpoint for job monitoring as HTML
74 </p>
75 <p>
76     href:<a href="localhost:5000/coverage?f=application/json">
77         localhost:5000/coverage?f=application/json</a><br>
78     rel: coverage<br>
79     type: application/json<br>
80     title: The endpoint for coverage as JSON
81 </p>
82 <p>
83     href:<a href="localhost:5000/coverage?f=text/html">
84         localhost:5000/coverage?f=text/html</a><br>
85     rel: coverage<br>
86     type: text/html<br>
87     title: The endpoint for coverage as HTML
88 </p>
89 </body>
90 </html>

```

D.2 landingPage.json

Ressource D.2: landingPage.json

```

1 {
2     "links": [
3         {
4             "href": "localhost:5000/?f=application/json",
5             "rel": "self",
6             "type": "application/json",
7             "title": "This document"
8         }, {
9             "href": "localhost:5000/?f=text/html",
10            "rel": "alternate",
11            "type": "text/html",
12            "title": "This document as HTML"
13        },
14        {
15            "href": "localhost:5000/api?f=application/json",
16            "rel": "service-desc",
17            "type": "application/json",
18            "title": "API definition for this endpoint as JSON"
19        },
20        {
21            "href": "localhost:5000/api?f=text/html",
22            "rel": "service-desc",
23            "type": "text/html",
24            "title": "API definition for this endpoint as HTML"
25        },
26        {

```

```

27     "href": "localhost:5000/conformance?f=application/json",
28     "rel": "conformance",
29     "type": "application/json",
30     "title": "OGC API - Processes conformance classes implemented by this server as JSON"
31 },
32 {
33     "href": "localhost:5000/conformance?f=text/html",
34     "rel": "conformance",
35     "type": "text/html",
36     "title": "OGC API - Processes conformance classes implemented by this server as HTML"
37 },
38 {
39     "href": "localhost:5000/processes?f=application/json",
40     "rel": "processes",
41     "type": "application/json",
42     "title": "Metadata about the processes as JSON"
43 },
44 {
45     "href": "localhost:5000/processes?f=text/html",
46     "rel": "processes",
47     "type": "text/html",
48     "title": "Metadata about the processes as HTML"
49 },
50 {
51     "href": "localhost:5000/jobs?f=application/json",
52     "rel": "jobs",
53     "type": "application/json",
54     "title": "The endpoint for job monitoring as JSON"
55 },
56 {
57     "href": "localhost:5000/jobs?f=text/html",
58     "rel": "jobs",
59     "type": "text/html",
60     "title": "The endpoint for job monitoring as HTML"
61 },
62 {
63     "href": "localhost:5000/coverage?f=application/json",
64     "rel": "coverage",
65     "type": "application/json",
66     "title": "The endpoint for coverage as JSON"
67 },
68 {
69     "href": "localhost:5000/coverage?f=text/html",
70     "rel": "coverage",
71     "type": "text/html",
72     "title": "The endpoint for coverage as HTML"
73 }
74 ]
75 }

```

D.3 confClasses.html

Ressource D.3: confClasses.html

```

1 <!DOCTYPE html>
2 <html>
3   <body>

```

```

4      <h1>conforms to:</h1>
5      <p><a href="https://docs.ogc.org/is/18-062r2/18-062r2.html#toc21">
6      https://docs.ogc.org/is/18-062r2/18-062r2.html#toc21</a></p>
7      <p><a href="https://docs.ogc.org/is/18-062r2/18-062r2.html#toc40">
8      https://docs.ogc.org/is/18-062r2/18-062r2.html#toc40</a></p>
9      <p><a href="https://docs.ogc.org/is/18-062r2/18-062r2.html#toc41">
10     https://docs.ogc.org/is/18-062r2/18-062r2.html#toc41</a></p>
11     <p><a href="https://docs.ogc.org/is/18-062r2/18-062r2.html#toc42">
12     https://docs.ogc.org/is/18-062r2/18-062r2.html#toc42</a></p>
13     <p><a href="https://docs.ogc.org/is/18-062r2/18-062r2.html#toc47">
14     https://docs.ogc.org/is/18-062r2/18-062r2.html#toc47</a></p>
15     <p><a href="https://docs.ogc.org/is/18-062r2/18-062r2.html#toc53">
16     https://docs.ogc.org/is/18-062r2/18-062r2.html#toc53</a></p>
17     <p>
18         <b>links:</b><br><br>
19         <b>href:</b> <a href="localhost:5000/conformance?f=text/html">
20         localhost:5000/conformance?f=text/html</a><br>
21         <b>rel:</b> self<br>
22         <b>type:</b> text/html<br>
23         <b>title:</b> This Document as HTML<br>
24         <br>
25         <b>href:</b> <a href="localhost:5000/conformance?f=application/json">
26         localhost:5000/conformance?f=application/json</a><br>
27         <b>rel:</b> alternate<br>
28         <b>type:</b> application/json<br>
29         <b>title:</b> This document as JSON<br>
30     </p>
31 </body>
32 </html>

```

D.4 confClasses.json

Ressource D.4: confClasses.json

```

1 {
2   "conformsTo": [
3     "https://docs.ogc.org/is/18-062r2/18-062r2.html#toc21",
4     "https://docs.ogc.org/is/18-062r2/18-062r2.html#toc40",
5     "https://docs.ogc.org/is/18-062r2/18-062r2.html#toc41",
6     "https://docs.ogc.org/is/18-062r2/18-062r2.html#toc42",
7     "https://docs.ogc.org/is/18-062r2/18-062r2.html#toc47",
8     "https://docs.ogc.org/is/18-062r2/18-062r2.html#toc53"
9   ],
10  "links": [
11    {
12      "href": "localhost:5000/conformance?f=application/json",
13      "rel": "self",
14      "type": "application/json",
15      "title": "This Document as JSON"
16    },
17    {
18      "href": "localhost:5000/conformance?f=text/html",
19      "rel": "alternate",
20      "type": "text/html",
21      "title": "This Document as HTML"
22    }
23  ]

```

D.5 processList.html

Ressource D.5: processList.html

```

1 <!DOCTYPE html>
2 <html>
3   <body>
4     {% block body %}
5     <p>
6       {% for process in processes %}
7       <b>Title:</b> {{process.title}}<br>
8       <b>processID: </b>{{process.id}}<br>
9       Description: {{process.description}}<br>
10      Version: {{process.version}}<br>
11      Job control options: {{process.jobControlOptions}}<br>
12      Output transmission: {{process.outputTransmission}}<br>
13      <p><b>inputs:</b></p>
14      <p id='{{process.id}}inputs'><p>
15      <p><b>outputs:</b></p>
16      <p id='{{process.id}}outputs'><p>
17      <script>
18        var inputs = '{{process.inputs}}'
19        var inputsString = inputs.replaceAll('&#39;', '')
20        var inputsJSON = JSON.parse(inputsString)
21        var outputs = '{{process.outputs}}'
22        var outputsString = outputs.replaceAll('&#39;', '')
23        var outputsJSON = JSON.parse(outputsString)
24        console.log(outputsJSON)
25        for (var key in inputsJSON) {
26          var inputs = document.createElement("p")
27          inputs.innerHTML = "<p><b>" + key
28            + "</b><br>Description: "
29            + inputsJSON[key].description
30            + "<br><b>Schema:</b><br>Type: "
31            + inputsJSON[key].schema.type + "</p>"
32          document.getElementById('{{process.id}}inputs').appendChild(inputs);
33        }
34        for (var key in outputsJSON) {
35          var outputs = document.createElement("p")
36          outputs.innerHTML = "<p><b>" + key
37            + "</b><br><b>Description: </b>"
38            + outputsJSON[key].description
39            + "<br><b>Schema:</b><br>Type: "
40            + outputsJSON[key].schema.type + "</p>"
41          document.getElementById('{{process.id}}outputs').appendChild(outputs);
42        }
43      </script>
44      <p>
45        <b>links: </b><br>
46        href: <a href=localhost:5000/processes/{{process.id}}?f=application/json>
47        localhost:5000/processes/{{process["id"]}}?f=application/json</a><br>
48        rel: process<br>
49        type: application/json<br>
50        title: Process description<br><br>
51        href: <a href=localhost:5000/processes/{{process.id}}?f=text/html>

```

```

52         localhost:5000/processes/{{process["id"]}}?f=text/html</a><br>
53         rel: process<br>
54         type: text/html<br>
55         title: Process description<br>
56     </p>
57     <p>=====</p>
58     {% endfor %}
59 </p>
60 {% endblock %}
61 <p><b>links:</b><br>
62     href:<a href="localhost:5000/processes?f=text/html">
63     localhost:5000/processes?f=text/html</a><br>
64     rel: self<br>
65     type: text/html<br>
66     title: This document<br>
67     <br>
68     href:<a href="localhost:5000/processes?f=application/json">
69     localhost:5000/processes?f=application/json</a><br>
70     rel: alternate<br>
71     type: application/json<br>
72     title: This document as JSON<br>
73 </p>
74 </body>
75 </html>

```

D.6 processDescription.html

Ressource D.6: processDescription.html

```

1 <!DOCTYPE html>
2 <html>
3     <body>
4         <p>
5             <b>processID:</b> {{process.id}}<br>
6             <b>Title:</b> {{process.title}}<br>
7             <b>Description:</b> {{process.description}}<br>
8             <b>Version:</b> {{process.version}}<br>
9             <b>Job control options:</b> {{process.jobControlOptions}}<br>
10            <b>Output transmission mode:</b> {{process.outputTransmission}}
11        </p>
12        <p><b>inputs:</b></p>
13        <p id='{{process.id}}'><p>
14            <script>
15                var inputs = '{{process.inputs}}'
16                var inputsString = inputs.replaceAll('&#39;', '"')
17                var inputsJSON = JSON.parse(inputsString)
18                console.log(inputsJSON)
19                for (var key in inputsJSON) {
20                    var newElement = document.createElement("p")
21                    newElement.innerHTML = "<p><b>" + key
22                    + "</b><br>Description: "
23                    + inputsJSON[key].description
24                    + "<br><b>Schema:</b><br>Type: "
25                    + inputsJSON[key].schema.type + "</p>"
26                    document.getElementById('{{process.id}}').appendChild(newElement);
27                }
28            </script>

```

```

29     <p>
30     <b>links:</b><br>
31     href:<a href="localhost:5000/processes/{{process.id}}?f=text/html">
32     localhost:5000/processes/{{process.id}}?f=text/html</a><br>
33     rel: self<br>
34     type: text/html<br>
35     title: This document as HTML<br>
36     <br>
37     href:<a href="localhost:5000/processes/{{process.id}}?f=application/json">
38     localhost:5000/processes/{{process.id}}?f=application/json</a><br>
39     rel: alternate<br>
40     type: application/json<br>
41     title: This document as JSON
42 </p>
43 </body>
44 <html>

```

D.7 FloodMonitoringProcessDescription.json

Ressource D.7: FloodMonitoringProcessDescription.json

```

1 {
2   "id": "FloodMonitoring",
3   "title": "Flood Monitoring",
4   "description": "This process accepts a Test input and returns an echo",
5   "version": "1.0.0",
6   "jobControlOptions": [
7     "async-execute", "dismiss"
8   ],
9   "outputTransmission": [
10    "value"
11  ],
12  "inputs": {
13    "preDate": {
14      "title": "preDate",
15      "description": "The input value",
16      "schema": {
17        "type": "string"
18      }
19    },
20    "postDate": {
21      "title": "postDate",
22      "description": "The input value",
23      "schema": {
24        "type": "string"
25      }
26    },
27    "username": {
28      "title": "username",
29      "description": "The input value",
30      "schema": {
31        "type": "string"
32      }
33    },
34    "password": {
35      "title": "password",
36      "description": "The input value",

```

```

37     "schema": {
38         "type": "string"
39     }
40 },
41 "bbox": {
42     "title": "Bounding Box Input Example",
43     "description": "This is an example of a BBOX literal input.",
44     "schema": {
45         "type": "object",
46         "required": [
47             "bbox"
48         ],
49         "properties": {
50             "bbox": {
51                 "type": "array",
52                 "oneOf": [
53                     {
54                         "minItems": 4,
55                         "maxItems": 4
56                     },
57                     {
58                         "minItems": 6,
59                         "maxItems": 6
60                     }
61                 ],
62                 "items": {
63                     "type": "number"
64                 }
65             },
66             "crs": {
67                 "type": "string",
68                 "format": "uri",
69                 "default": "http://www.opengis.net/def/crs/OGC/1.3/CRS84",
70                 "enum": [
71                     "http://www.opengis.net/def/crs/OGC/1.3/CRS84",
72                     "http://www.opengis.net/def/crs/OGC/0/CRS84h"
73                 ]
74             }
75         }
76     }
77 },
78 },
79 "outputs": {
80     "bin": {
81         "title": "outputDocument",
82         "description": "The output document",
83         "schema": {
84             "type": "string",
85             "contentEncoding": "binary",
86             "contentMediaType": "application/tiff"
87         }
88     },
89     "ndsi": {
90         "title": "outputDocument",
91         "description": "The output document",
92         "schema": {
93             "type": "string",
94             "contentEncoding": "binary",
95             "contentMediaType": "application/tiff"

```

```

96     }
97   }
98 },
99 "links": [
100 {
101   "href": "localhost:5000/processes/Echo?f=application/json",
102   "rel": "self",
103   "type": "application/json",
104   "title": "This document"
105 },
106 {
107   "href": "localhost:5000/processes/Echo?f=text/html",
108   "rel": "alternate",
109   "type": "text/html",
110   "title": "This document as HTML"
111 }
112 ]
113 }

```

D.8 EchoProcessDescription.json

Ressource D.8: EchoProcessDescription.json

```

1 {
2   "id": "Echo",
3   "title": "Echo",
4   "description": "This process accepts a Test input and returns an echo",
5   "version": "1.0.0",
6   "jobControlOptions": [
7     "async-execute", "dismiss"
8   ],
9   "outputTransmission": [
10     "value"
11   ],
12   "inputs": {
13     "echo": {
14       "title": "echo",
15       "description": "Value to be echoed",
16       "schema": {
17         "type": "string"
18       }
19     }
20   },
21   "outputs": {
22     "outgoingEcho": {
23       "title": "outgoingEcho",
24       "description": "The output document containing the echoed value",
25       "schema": {
26         "type": "object",
27         "contentType": "application/json",
28         "required": [
29           "result",
30           "message"
31         ],
32         "properties": {
33           "result": {
34             "type": "string"

```



```

35     },
36     "message": {
37         "type": "string"
38     }
39 }
40 }
41 }
42 },
43 "links": [
44     {
45         "href": "localhost:5000/processes/Echo?f=application/json",
46         "rel": "self",
47         "type": "application/json",
48         "title": "This document"
49     },
50     {
51         "href": "localhost:5000/processes/Echo?f=text/html",
52         "rel": "alternate",
53         "type": "text/html",
54         "title": "This document as HTML"
55     }
56 ]
57 }

```

D.9 jobList.html

Ressource D.9: jobList.html

```

1  <!DOCTYPE html>
2  <html>
3      <body>
4          {% block body %}
5              <p>
6                  {% for status in status %}
7                      <b>jobID:</b> {{status.jobID}}<br>
8                      processID: {{status.processID}}<br>
9                      Type: {{status.type}}<br>
10                     Status: {{status.status}}<br>
11                     Message: {{status.message}}<br>
12                     Progress: {{status.progress}}<br>
13                     Created: {{status.created}}<br>
14                     Started: {{status.started}}
15                 <p>
16                     <b>links: </b><br>
17                     href:<a href="localhost:5000/jobs/{{status.jobID}}?f=text/html">
18                     localhost:5000/jobs/{{status.jobID}}?f=text/html</a><br>
19                     rel: status<br>
20                     type: text/html,<br>
21                     title: status as HTML<br>
22                     <br>
23                     href:<a href="localhost:5000/jobs/{{status.jobID}}?f=application/json">
24                     localhost:5000/jobs/{{status.jobID}}?f=application/json</a><br>
25                     rel: status<br>
26                     type: application/json,<br>
27                     title: status as JSON<br>
28                 </p>
29                 <p>=====</p>

```

```

30         {% endfor %}
31     </p>
32     {% endblock %}
33     <p><b>links:</b><br>
34         href:<a href="localhost:5000/jobs?f=text/html">
35         localhost:5000/jobs?f=text/html</a><br>
36         rel: self<br>
37         type: text/html<br>
38         title: This document<br>
39         <br>
40         href:<a href="localhost:5000/jobs?f=application/json">
41         localhost:5000/jobs?f=application/json</a><br>
42         rel: alternate<br>
43         type: application/json<br>
44         title: This document as JSON<br>
45     </p>
46 </body>
47 </html>

```

D.10 job.html

Ressource D.10: job.html

```

1  <!DOCTYPE html>
2  <html>
3      <body>
4          <p>
5              <b>jobID: {{job.jobID}}</b><br>
6              <b>processID: {{job.processID}}</b><br>
7              Status: {{job.status}}<br>
8              Message: {{job.message}}<br>
9              Progress: {{job.progress}}<br>
10             Created: {{job.created}}<br>
11             Cstarted: {{job.started}}<br><br>
12             <b>links:</b>
13             <br>href: <a href="localhost:5000/jobs/{{job.jobID}}?f=text/html">
14             localhost:5000/jobs/{{job.jobID}}?f=text/html</a>
15             <br>rel: self
16             <br>title: Job Status
17             <br> type: text/html<br>
18             <br>href: <a href="localhost:5000/jobs/{{job.jobID}}?f=application/json">
19             localhost:5000/jobs/{{job.jobID}}?f=application/json</a>
20             <br>rel: alternate
21             <br>title: Job Status
22             <br>type: application/json
23         </p>
24     </body>
25 </html>

```

D.11 coverage.html

Ressource D.11: coverage.html

```

1  <!DOCTYPE html>

```

```

2 <html>
3 <body>
4 <link rel="stylesheet" href="https://unpkg.com/leaflet@1.8.0/dist/leaflet.css"
5 integrity="sha512-hoalWLoI8r4UszCkZ5kL8vay0GVae1oxXe/2A4A06J9+580uKHD03JdHb7NzwzK5xr/Fs0W40kiNHxM9
6 crossorigin="" />
7 <script src="https://unpkg.com/leaflet@1.8.0/dist/leaflet.js"
8 integrity="sha512-BB3hKbKW0c9Ez/TAwyWxNXeoV9c1v6FieYiBieIWkpLjauysF18Nzgr1MBNBXf8/KABdlkX68nAhlwCDF
9 crossorigin=""></script>
10 <h1>coverage:</h1>
11 {% block body %}
12 <p>
13     {% for coverage in coverages %}
14     <b>Product:</b> {{coverage.name}}<br>
15     BBox: {{coverage.bbox}}<br>
16     Date: {{coverage.date}}<br>
17     <br>
18     {% endfor %}
19 </p>
20 {% endblock %}
21 <div id="map" style="width: 600px; height: 400px"></div>
22 <script>
23     var map = L.map('map').setView([51.505, -0.09], 13);
24     L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
25         maxZoom: 19,
26         attribution: 'Copyright OpenStreetMap'
27     }).addTo(map);
28
29     var bboxes = '{{bboxes}}';
30     var bboxesString = bboxes.replaceAll('&#39;', '"');
31     var bboxesJSON = JSON.parse(bboxesString)
32
33     var coverage = L.featureGroup().addTo(map)
34     for (var i=0; i < bboxesJSON.length; i++) {
35         name = bboxesJSON[i].features[0].properties.name
36         date = bboxesJSON[i].features[0].properties.date
37         console.log(name)
38         layer = L.geoJSON(bboxesJSON[i]).bindPopup("Name: " + name + "<br>Date: " + date, {maxWidth: "a
39         layer.addTo(coverage)
40     }
41     map.fitBounds(coverage.getBounds())
42 </script>
43 <p><b>links:</b><br>
44     href:<a href="localhost:5000/coverage?f=text/html">localhost:5000/coverage?f=text/html</a><br>
45     rel: self<br>
46     type: text/html<br>
47     title: This document<br>
48     <br>
49     href:<a href="localhost:5000/coverage?f=application/json">localhost:5000/coverage?f=application/j
50     rel: alternate<br>
51     type: application/json<br>
52     title: This document as JSON<br>
53 </p>
54 </body>
55 </html>

```

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die Bachelorarbeit zum Thema Rich Data Interfaces for Copernicus Data selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.

Münster, den 13. Juli 2022