

Landwirtschaftliche Datendrehscheibe für effiziente Ressourcenschonende Prozesse



Deliverable D5.2

Prozesse und Dokumentation für Betrieb und Erweiterung der Systeme

Version	1.0
Datum	11.09.2020
Verantwortlicher Partner	Hochschule Osnabrück
Art des Deliverables	Dokument
Verbreitung	<Öffentlich>
Projektkoordinator	Krone

Gefördert durch:



aufgrund eines Beschlusses
des Deutschen Bundestages



Autoren

Dieses Dokument wurde erstellt von Hochschule Osnabrück

Beiträge wurden verfasst von

Julian Klose – Hochschule Osnabrück

Noah Große Starmann – Hochschule Osnabrück

© Copyright 2020 SDSD
Koordinator: Maschinenfabrik Bernard KRONE GmbH & Co. KG

Versionen

Version	Datum	Beschreibung
0.0	11.09.2020	Erster Entwurf durch Hochschule Osnabrück
1.0	29.09.2020	Finalisierung durch Hochschule Osnabrück

Zusammenfassung

Im ersten Teil wird die Inbetriebnahme der SDSD-Plattform beschrieben.

Dazu wird zunächst im ersten Schritt die Installation der Datenbankmanagementsysteme *MongoDB*, *Cassandra* und *Stardog* beschrieben. Es sind für den erfolgreichen Betrieb der SDSD-Plattform in Kombination mit den genannten Datenbankmanagementsystemen nur Konfigurationsschritte durch den Einrichtenden für *Cassandra* und *Stardog* auszuführen. Die restliche initiale Konfiguration erfolgt beim ersten Aufruf des SDSD-Servers. Der zweite Schritt dreht sich um den eigentlichen SDSD-Server. Es wird ein Service angelegt, der die komfortable Steuerung der Plattform über *systemctl* vornimmt. Im letzten Schritt wird die SSL Verschlüsselung für den SDSD-Server angelegt. Dazu werden die Werkzeuge *HAProxy*, *Let's Encrypt* und *Certbot* eingesetzt.

Außerdem wird im Rahmen der Dokumentation über den Betrieb der Systeme erläutert, wie Parser und Dienste zu entwickeln und zu betreiben sind.

Inhaltsverzeichnis

1	Inbetriebnahme der SDSD-Plattform.....	6
1.1	Installation der Datenbankmanagementsysteme	6
1.2	Erstellung einer eigenen SDSD Konfigurationsdatei.....	9
1.3	Start und Bereitstellung des Servers	14
1.4	Inbetriebnahme der SSL Verschlüsselung	15
2	Parser- / Dienstentwicklung	20
2.1	Parserentwicklung.....	20
2.1.1	Anlegen von Wikinormia Formaten und Klassen	20
2.1.2	Erstellung eines Parsers mit Wikinormia Typen	22
2.1.3	Kreation eines Parsers mit einem Apache Jena RDF Modell	31
2.1.4	Geodaten in einem Parser verarbeiten.....	34
2.1.5	TimeLogs im Parser verarbeiten	35
2.2	Entwicklung eines Dienstes.....	37
3	Resultate	39

1 Inbetriebnahme der SDSD-Plattform

Das Ziel dieses Dokuments ist es, den Leser in die Lage zu versetzen, die SDSD Plattform in Betrieb zu nehmen, um sie nachfolgend weitergehend verwenden zu können. Es werden die dafür verwendeten Prozesse erläutert und eine Dokumentation erstellt. Um die Plattform in Betrieb nehmen zu können, müssen Datenbankmanagementsysteme installiert werden, die SSL-Verschlüsselung konfiguriert werden, sowie der eigentliche SDSD Server gestartet werden. Außerdem muss eine eigene SDSD-Konfigurationsdatei erstellt werden. Diese Schritte werden im Folgenden erläutert. Die Unterkapitel sind dabei so aufgebaut, dass sie aufeinanderfolgend den Prozess zur Installation der Plattform dokumentieren. Der Installationsprozess wird für einen Server mit einem Linux-Betriebssystem auf Basis von Ubuntu erläutert. Daher nutzen die gezeigten Befehle die Paketverwaltung *apt*. Eine weitere Grundannahme ist hier, dass „app.sdsd-projekt.de“ als Domain genutzt wird. Diese ist durch die eigene URL zu ersetzen.

1.1 Installation der Datenbankmanagementsysteme

Die SDSD-Plattform verwendet zur Datenspeicherung die Datenbankmanagementsysteme MongoDB, Redis, Cassandra und Stardog. Die Installation dieser Systeme wird in diesem Abschnitt erläutert. Die Konfiguration der Datenbankmanagementsysteme erfolgt in großen Teilen beim ersten Start des SDSD-Servers; lediglich bei Cassandra und Stardog sind noch Schritte für die Konfiguration auszuführen. Für MongoDB müssen Benutzer und Datenbanken generiert werden.

Im ersten Schritt werden die Datenbankmanagementsysteme MongoDB und Redis installiert. Diese Installation erfolgt durch das folgende Kommando:

```
sudo apt install mongodb redis
```

Redis benötigt keine weitere Konfiguration. Es müssen dort keine Benutzer angelegt werden. Bei MongoDB hingegen sollten Benutzer für die weitere Verwendung generiert werden. Es wird empfohlen, einen Nutzer *sdsd*, sowie eine Datenbank *sdsd* anzulegen (s. <https://docs.mongodb.com/manual/tutorial/create-users/>, <https://www.mongodb.com/basics/create-database>). Bei MongoDB ist ebenfalls zu beachten, dass ein externer Zugriff nur dann erfolgen kann, wenn MongoDB entsprechend konfiguriert ist. Sollten hier keine Kenntnisse vorhanden sein, so ist hier empfohlen die MongoDB Dokumentation einzusehen

(<https://docs.mongodb.com/manual/core/security-mongodb-configuration/>).

Nachfolgend wird das Cassandra Datenbankmanagementsystem heruntergeladen und installiert. Cassandra ist nicht in den Standard-Paketquellen der *apt*-Paketverwaltung inkludiert. Daher muss eine neue Paketquelle hinzugefügt werden. Dies geschieht mittels der folgenden Kommandos:

```
curl https://downloads.apache.org/cassandra/KEYS | sudo
apt-key add -
echo "deb https://downloads.apache.org/cassandra/debian
311x main" | sudo tee -a
/etc/apt/sources.list.d/cassandra.sources.list
```

Anschließend müssen die Paketquellen mittels *apt update* aktualisiert werden. Schlussendlich kann das Paket *cassandra* mithilfe von *apt install* installiert werden.

Auch beim Datenbankmanagementsystem Stardog muss, bevor das eigentliche Paket installiert werden kann, eine neue Paketquelle hinzugefügt werden. Kongruent zum vorhergehenden Befehl geschieht dies wie im Folgenden zu sehen.

```
curl http://packages.stardog.com/stardog.gpg.pub | sudo
apt-key add
echo "deb http://packages.stardog.com/deb/ stable main"
| sudo tee -a
/etc/apt/sources.list.d/stardog.sources.list
```

Nach einer erneuten Aktualisierung der Paketquellen kann das Paket heruntergeladen werden. Dabei sollte die Version mit angegeben werden.

```
sudo apt install -y stardog=7.3.1
```

Damit sind nun alle notwendigen Datenbankmanagementsysteme installiert.

Stardog kann als *systemd* Service bedient werden, oder es kann über das Terminal gesteuert werden. Die Empfehlung lautet hier, Stardog als Service zu betreiben, damit es bequem möglich ist, das Datenbankmanagementsysteme über *systemctl* zu steuern. Dies ist mit dem heruntergeladenen Paket möglich. Es ist zu beachten, dass für Stardog eine Lizenz zum Betrieb benötigt wird. Eine solche Lizenz kann auf der Internetseite des Stardog-Projekts (s. <https://www.stardog.com/docs>) erhalten werden. Hat man eine passende Lizenz, so ist die entsprechende Datei zum Ort der *STARDOG_HOME* Umgebungsvariablen zu kopieren. Die Konfiguration von Stardog und dessen Umgebungsvariablen ist in der Datei */etc/stardog.env.sh* einzusehen. Standardmäßig ist die *STARDOG_HOME* Variable auf den Wert */var/opt/stardog* gesetzt. Dieses sollte nicht ohne genauere Prüfung geändert werden.

Stardog muss im letzten Schritt für die weitere Verwendung mit der Wikinormia initialisiert werden. Es existiert dazu eine Initialisierungsmethode `triple.insertDefaultsIntoWikinormia()` im SDSD-Quelltext, die ausgeführt werden muss. Sie könnte beispielsweise im `testRunner` in `ApplicationLogic` ausgeführt werden.

Bezüglich Cassandra ist zu beachten, dass der standardmäßige Zugriff zunächst über den Nutzer `cassandra` erfolgt. Es wird aus Sicherheitsgründen hier dringend empfohlen, diesen Benutzer abzuändern. Eine passende Struktur wäre beispielsweise die Nutzung eines `dba` Benutzers für die Verwaltung der Datenbanken mit vollen Zugriffsrechten, sowie der Einsatz eines Nutzers `sdsd`, der nur Zugriff auf die SDSD-Daten hat. Um Nutzer zu kreieren sei hier auf die Dokumentation von Cassandra verwiesen (<https://cassandra.apache.org/doc/latest/operating/security.html>, <https://cassandra.apache.org/doc/latest/cql/security.html#>). Ein wichtiger Hinweis dazu ist, dass zunächst in der `cassandra.yaml` die Authentifizierung aktiviert werden muss; dies ist im ersten Link beschrieben. Weiterhin ist bei Cassandra zu beachten, dass der Betrieb von SDSD es erforderlich macht, dass eine grundlegende Struktur angelegt wird. Dazu sind die folgenden CQLSH Skripts auszuführen. Dies kann mithilfe des bei Cassandra mitgelieferten Terminal Programms `cqlsh` geschehen. Beim Aufruf des Programms sollten selbstverständlich die IP-Adresse des Servers, der Port und der entsprechende Benutzername sowie das Passwort angegeben werden. Ist die Verbindung erfolgreich, müssen die folgenden CQLSH Anweisungen auf der Datenbank ausgeführt werden:

```
--DROP TABLE sdsd.position_generic;
CREATE TABLE sdsd.position_generic (
  user text,
  file text,
  name text,
  time timestamp,
  altitude int,
  latitude int,
  longitude int,
  PRIMARY KEY ((user, file, name), time) )
WITH CLUSTERING ORDER BY (time DESC);
--DROP TABLE sdsd.timelog_generic;
CREATE TABLE sdsd.timelog_generic (
  user text,
  file text,
  name text,
  value_uri text,
  time timestamp,
  value bigint,
  PRIMARY KEY ((user, file, name, value_uri), time) )
WITH CLUSTERING ORDER BY (time DESC);
--DROP TABLE sdsd.grid_generic;
```



```
CREATE TABLE sdsd.grid_generic (
  user text,
  file text,
  name text,
  value_uri text,
  north_min int,
  east_min int,
  east_size int static,
  north_size int static,
  value bigint,
  PRIMARY KEY ((user, file, name, value_uri), north_min,
  east_min) )
WITH CLUSTERING ORDER BY (north_min ASC, east_min ASC);
--DROP TABLE sdsd.position_keys;
CREATE TABLE sdsd.position_keys (
  user text,
  file text,
  name text,
  PRIMARY KEY ((user, file), name) )
WITH clustering ORDER BY (name ASC);
--DROP TABLE sdsd.timelog_keys;
CREATE TABLE sdsd.timelog_keys (
  user text,
  file text,
  name text,
  value_uri text,
  PRIMARY KEY ((user, file), name, value_uri) )
WITH clustering ORDER BY (name ASC, value_uri ASC);
--DROP TABLE sdsd.grid_keys;
CREATE TABLE sdsd.grid_keys (
  user text,
  file text,
  name text,
  value_uri text,
  PRIMARY KEY ((user, file), name, value_uri) )
WITH clustering ORDER BY (name ASC, value_uri ASC);
```

Damit sind die Datenbankmanagementsysteme installiert und für die weitere Verwendung bereit.

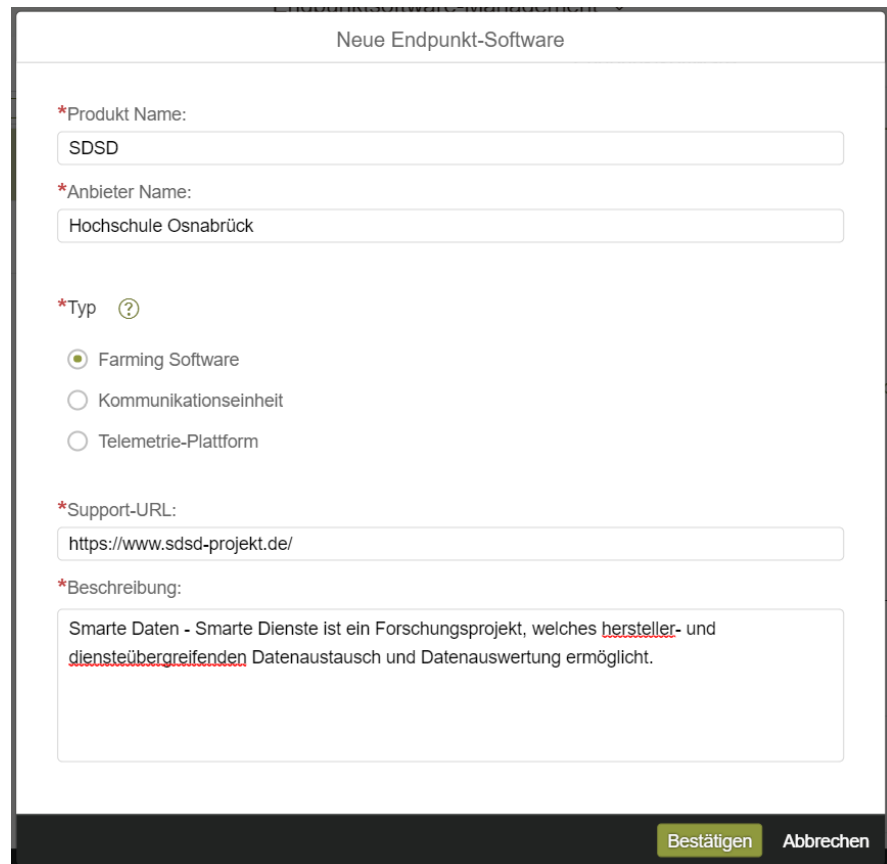
1.2 Erstellung einer eigenen SDSD Konfigurationsdatei

In diesem Abschnitt wird beschrieben, wie eine eigene SDSD Konfigurationsdatei erstellt wird. Dieser Schritt wird benötigt, um den Start und die Bereitstellung des Plattform-Servers zu ermöglichen.

Die Konfigurationsdatei ist mit *settings.json* benannt. Es handelt sich dabei um eine Datei, die alle Zugangsdaten und Passwörter enthält, die für den Betrieb der SDSD-Plattform und des Servers benötigt werden. So sind neben den Zugangsdaten der Datenbankmanagementsysteme auch die Zugangsdaten des

Agrirouters, sowie das Administratorpasswort der SDSD-Plattform inkludiert.


Es wird damit begonnen, die notwendigen Schritte zum Erhalt der Zugangsdaten des Agrirouters zu erläutern. Zunächst ist sich dazu auf der Internetseite als Entwickler zu registrieren (**Achtung:** „Anmelden als Entwickler“ Schaltfläche muss für Registrierung genutzt werden). Anschließend erfolgt ein Login und man befindet sich in der „agrirouter für Entwickler“ Ansicht. Dort ist eine neue Endpunkt Software anzulegen. Wichtig ist dabei, dass als Typ „Farming Software“ gewählt wird; die ist in der folgenden Abbildung zu sehen.



Neue Endpunkt-Software

*Produkt Name:
SDSD

*Anbieter Name:
Hochschule Osnabrück

*Typ 
☒ Farming Software
☐ Kommunikationseinheit
☐ Telemetrie-Plattform

*Support-URL:
https://www.sdsd-projekt.de/

*Beschreibung:
Smarte Daten - Smarte Dienste ist ein Forschungsprojekt, welches hersteller- und diensteübergreifenden Datenaustausch und Datenauswertung ermöglicht.

Bestätigen Abbrechen

Anschließend muss die angezeigte Fehlermeldung beseitigt werden. Dazu wird die Endpunktsoftware erneut bearbeitet. Es muss die URL-Umleitung eingetragen werden. Diese besteht aus der Domain und /rest/onboard. Der Schlüssel wird über Schlüsselpaar generieren erzeugt. Der private Schlüssel muss kopiert werden, um ihn später in der Konfigurationsdatei einsetzen zu können.

Sicherheit

*URL umleiten:

<https://app.sdsd-projekt.de/rest/onboard>

*Öffentlicher Schlüssel:

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA9a6jHEE3IBDprg9mqn2r
HpRBNf6p6x9yr2weHywDkigRktBvt05Eg8TcuFDvF6+mII+P8hULvynBWGF0cbFW
sU9gl/QiQoJf72LL35E2n5C6U9cerOuFgktubu3bbw0G8B5tjDhxSz0Uvctkpngp
TxLesXdFoyCoLNhOjtxlGTnFhTV9BpP7Q7toOqkr0lcrcRCSf/DgFgPcu7klDI
```

Schlüsselpaar generieren

Im nächsten Schritt wird eine Endpunktsoftware Version angelegt. Dort sind alle Nachrichtentypen auszuwählen und die Version muss eingereicht werden. Nach dem erfolgreichen Genehmigungsprozess kann die Version genutzt werden.

Nun wird mit der eigentlichen Konfigurationsdatei fortgefahren. Die Struktur ist im Folgenden zu sehen.

```
{
  "agrirouter": {
    "endpointIdPrefix": "urn:sdsd:",
    "host": "https://goto.my-agrirouter.com",
    "onboardingUrl": "https://agrirouter-registration-
service.cfapps.eu1.hana.ondemand.com/api/v1.0/registration/onboard",
    "applicationId": "???",
    "certificationVersionId": "???",
    "capabilities": [
      { "technicalMessageType": "iso:11783:-10:taskdata:zip",
"direction": 2 },
      { "technicalMessageType": "iso:11783:-
10:device_description:protobuf", "direction": 2 },
      { "technicalMessageType": "iso:11783:-
10:time_log:protobuf", "direction": 2 },
      { "technicalMessageType": "shp:shape:zip", "direction": 2
    },
      { "technicalMessageType": "img:bmp", "direction": 2 },
      { "technicalMessageType": "img:jpeg", "direction": 2 },
      { "technicalMessageType": "img:png", "direction": 2 },
      { "technicalMessageType": "vid:avi", "direction": 2 },
      { "technicalMessageType": "vid:mp4", "direction": 2 },
      { "technicalMessageType": "vid:wmv", "direction": 2 },
      { "technicalMessageType": "doc:pdf", "direction": 2 }
    ],
    "appPrivateKey": "???",
    "arPublicKey":
: "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAY8xF9661acn+iS+QS+9Y3HvTfUVc
ismzbuvxHgHA7YeoOUFxyj3lkaTnXm7hzQe4wDEDgwpJ5GAzxIIYSUXe8EsWLorg500tRexx5S
P3+kj1i83DATBJCXP7k+bAF4u2FVJphC1m2BfLxelGLjzxVAS/v6+EwvYaT1AI9FFqW/a2o92I
sVPOh9oM9eds3lBOAbH/8XrmVIEhofw+XbTH1/7MLD6IE2+HbEeY0F96nioXArdQWxcjUQsTch
+p0p9eqh23Ak4ef5oGcZhNd4ypY8M6ppvIMiXkgWSPJevCJjhXRJRmndY+ajYGx7CLePx7wNvx
XWtkng3yh+7WiZ/YqwIDAQAB"},
    "agrirouter-qa": {
      "endpointIdPrefix": "urn:sdsd:",
      "host": "https://agrirouter-qa.cfapps.eu10.hana.ondemand.com",
      "onboardingUrl": "https://agrirouter-registration-service-
hubqa-eu10.cfapps.eu10.hana.ondemand.com/api/v1.0/registration/onboard",
      "applicationId": "???",
      "certificationVersionId": "???",
      "capabilities": [
        { "technicalMessageType": "dke:other", "direction": 2 },
        { "technicalMessageType": "iso:11783:-10:taskdata:zip",
"direction": 2 },
        { "technicalMessageType": "iso:11783:-
10:device_description:protobuf", "direction": 2 },
        { "technicalMessageType": "iso:11783:-
10:time_log:protobuf", "direction": 2 },
        { "technicalMessageType": "shp:shape:zip", "direction": 2
      },
    },
  },
}
```

```

        { "technicalMessageType": "img:bmp", "direction": 2 },
        { "technicalMessageType": "img:jpeg", "direction": 2 },
        { "technicalMessageType": "img:png", "direction": 2 },
        { "technicalMessageType": "vid:avi", "direction": 2 },
        { "technicalMessageType": "vid:mp4", "direction": 2 },
        { "technicalMessageType": "vid:wmv", "direction": 2 },
        { "technicalMessageType": "doc:pdf", "direction": 2 }
    ],
    "appPrivateKey": "???",
    "arPublicKey" : "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIB
BCgKCAQEAy8xF9661acn+iS+QS+9Y3Hv
TfUVcismzbuvxHgHA7YeoOUFxyj3lkaT
nXm7hzQe4wDEDgwpJSGAzxIIYSUXe8EsWLorg500
tRexx5SP3+kj1i83DATBJCXP7k+bAF4u2FVJphC1m2BfLxe1
GLjzxVAS/v6+EwvYaT1AI9FFqW/a2o92IsVP0h9oM9eds3lB0AbH/8XrmVieHofw+Xb
TH1/7MLD6IE2+HbEeY0F96nioXArdQWxcjUQsTch+p0p9eqh23Ak4ef5oGcZhNd4ypY8M6ppvI
MiXkgWSPJevCJjhXRJRmndY+ajYGx7CLEPx7wNvxXWtkng3yh+7WiZ/YqwIDAQAB"

    },
    "redis": {
        "address": "redis://127.0.0.1:6379/"
    },
    "mongodb": {
        "address": "???",
        "user": "sdsd",
        "database": "sdsd",
        "password": "???"
    },
    "stardog": {
        "update": "???/sdsd/update",
        "query": "???/sdsd/query",
        "user": "???",
        "password": "???"
    },
    "cassandra": {
        "address": "???",
        "port": 9042,
        "keyspace": "sdsd",
        "user": "???",
        "password": "???"
    },
    "adminPassword": "???"
}

```

Alle mit „???“ gekennzeichneten Werte sind auszutauschen. Zu beachten ist, dass hier bei den Datenbanken die bereits konfigurierten Zugangsdaten zu nutzen sind. Wahrscheinlich wird

zunächst nur Zugang zum Agrirouter QA gewährt. Diese Zugangsdaten sind unter *agrirouterQA* einzutragen. Der *appPrivateKey* ist der im letzten Schritt kopierte Schlüssel. Die *ApplicationId* ist am erstellten Software Endpunkt zu finden.



Um die *CertificationVersionId* zu erhalten, muss bei der genehmigten Version die angezeigte ID kopiert werden.



Die erstellte Konfigurationsdatei ist an dem Pfad abzulegen, der im nächsten Schritt als *WorkingDirectory* für den Server genutzt werden soll.

1.3 Start und Bereitstellung des Servers

Nachdem die Datenbankmanagementsysteme erfolgreich installiert worden sind und erste Konfigurationen vorgenommen wurden, kann der eigentliche Server der SDSD-Plattform gestartet werden. Mit dem initialen Start werden die verbleibenden Datenbank-Schemata automatisch konfiguriert.

Um die Administration und Verwaltung des Servers zu vereinfachen, wird ein Service erstellt. Damit kann die Steuerung bequem über *systemctl* erfolgen. Um den Start ausführen zu können, werden die SDSD JAR Archive benötigt. Außerdem sollte

eine adäquates Java Development Kit (JDK11 oder höher) installiert sein.

Der Inhalt der Service Datei ist im Folgenden beschrieben. Es ist zu beachten, dass hier das *WorkingDirectory* immer auf */home/sdsd/website* verweist. Dieses ist nach eigenen Wünschen anzupassen. Das gewählte Verzeichnis muss außerdem den *view* Ordner enthalten.

```
[Unit]
Description=SDSD Website
After=redis.service
#After=redis.service mongod.service cassandra.service
stardog.service

[Service]
ExecStart=/usr/bin/java -jar
# Expecting the user is called sdsd
/home/sdsd/website/website.jar -H "https://app.sdsd-
projekt.de"
ExecStartPost=/bin/sleep 30
# Required on some systems
WorkingDirectory=/home/sdsd/website
Restart=always
# Restart service after 10 seconds if server crashes
RestartSec=10
# Output to syslog
StandardOutput=journal
StandardError=journal
SyslogIdentifier=sdsd-website
User=sdsd
#Group=sdsd

[Install]
WantedBy=multi-user.target
```

Dieser Service sollte, wie bekannt, entweder systemweit unter */etc/systemd/system* oder für den Benutzer unter */etc/systemd/user* gespeichert werden. Als Name wird empfohlen, *sdsdserver.service* zu setzen. Nach einem Neustart des Computers oder des *systemctl* Daemons kann der Server nun verwendet werden.

1.4 Inbetriebnahme der SSL Verschlüsselung

In diesem Abschnitt wird beschrieben, wie die von der SDSD-Plattform verwendete SSL Verschlüsselung mithilfe der Tools *HAProxy*, *Let's Encrypt* und *Certbot* realisiert wird.

Um die Verschlüsselung zu realisieren, muss als Erstes das Programm Certbot installiert werden. Certbot dient als kostenloses, quelloffenes Programm dazu, SSL und TLS Zertifikate der Zertifizierungsstelle *Let's Encrypt* zu verwalten. Um SDSD nutzen zu können, muss nicht zwingend *Let's Encrypt* verwendet werden. Für eine Produktivumgebung ist ein eigenes erstelltes Zertifikat zu bevorzugen. Im Folgenden wird *Let's Encrypt* beschrieben.

Die Installation erfolgt mithilfe der im Folgenden gezeigten Kommandos. Es wird zunächst die passende Paketquelle hinzugefügt und es werden die Paketquellen aktualisiert. Dann kann das Paket *certbot* installiert werden.

```
sudo add-apt-repository ppa:certbot/certbot
sudo apt-get update
sudo apt-get install certbot
```

Nachdem die Installation erfolgt ist, muss das eigentliche Zertifikat erstellt werden. Zunächst wird der *haproxy* Service gestoppt. Anschließend wird *certbot* mit den passenden Parametern genutzt, um das Zertifikat zu erstellen. Dazu werden die folgenden Befehle ausgeführt.

```
sudo service haproxy stop

sudo certbot certonly --standalone --preferred-challenges
http --http-01-port 80 -d app.sdsd-projekt.de
```

Nachdem das Zertifikat generiert worden ist, muss nun die Konfiguration von *HAProxy* am Pfad */etc/haproxy/haproxy.cfg* geändert werden. Die einzufügende Konfiguration ist im Folgenden ersichtlich.

```
global
    log /dev/log      local0
    log /dev/log      local1 notice
    chroot /var/lib/haproxy
    stats socket /run/haproxy/admin.sock mode 660
level admin
    stats timeout 30s
    user haproxy
    group ssl-cert
    daemon

    # Default SSL material locations
    ca-base /etc/ssl/certs
    crt-base /etc/ssl/private

    # Default ciphers to use on SSL-enabled listening
sockets.
```



```
# For more information, see ciphers(1SSL). This
list is from:
# https://hynek.me/articles/hardening-your-web-
servers-ssl-ciphers/
ssl-default-bind-ciphers
ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES128:D
H+AES:ECDH+3DES:DH+3DES:RSA+AESGCM:RSA+AES:RSA+3DES:!aNULL
L:!MD5:!DSS
ssl-default-bind-options no-sslv3
tune.ssl.default-dh-param 2048

defaults
    log      global
    mode     http
    option    httplog
    option    dontlognull
    option    redispatch
    option    contstats
    retries  3
    backlog  10000
    timeout  connect 5s
    timeout  client  50s
    timeout  server  50s
    timeout  tunnel  57600s
    timeout  http-keep-alive 1s
    timeout  http-request  15s
    timeout  queue    30s
    timeout  tarpit    60s
    default-server inter 3s rise 2 fall 3
    errorfile 400 /etc/haproxy/errors/400.http
    errorfile 403 /etc/haproxy/errors/403.http
    errorfile 408 /etc/haproxy/errors/408.http
    errorfile 500 /etc/haproxy/errors/500.http
    errorfile 502 /etc/haproxy/errors/502.http
    errorfile 503 /etc/haproxy/errors/503.http
    errorfile 504 /etc/haproxy/errors/504.http

frontend www
    bind      *:80
    mode      http
    redirect  scheme https code 301 if !{
ssl_fc }

frontend wwws
    bind      *:443 ssl crt
/etc/ssl/private/sdsd.pem
    reqadd    X-Forwarded-Proto:\ https
    acl acme   path_beg /.well-known/acme-
```

```
challenge
    use_backend certbot if acme
    default_backend sdsd

backend sdsd
    mode http
    stats enable
    stats uri /haproxy
    option forwardfor
    server java_sdsd localhost:8081 weight 1
maxconn 8192 check

backend certbot
    server certbot localhost:54321
```

Das Programm *HAProxy* muss nun noch der Usergruppe *ssl-cert-group* hinzugefügt werden. Dazu müssen die folgenden Befehle genutzt werden.

```
sudo usermod -a -G ssl-cert haproxy
sudo service haproxy start
```

Im nächsten Schritt wird ein Skript erstellt, mit dessen Hilfe ein Zertifikat bei Ablauf erneuert werden kann. Dazu werden die im Folgenden zu findenden Befehle in eine aussagekräftig benannte Datei mit der Endung *.sh* gespeichert. Als Pfad wird seitens der Verfasser dieses Dokuments */usr/local/bin/renew.sh* vorgeschlagen.

```
#!/bin/sh

cd /etc/letsencrypt/live/app.sdsd-projekt.de
cat fullchain.pem privkey.pem > /etc/ssl/private/sdsd.pem
service haproxy reload
```

Dieses erstellte Skript muss schlussendlich noch unter Verwendung des Werkzeuges *chmod* ausführbar gemacht werden und es müssen die entsprechenden Privilegien festgelegt werden, um auf die Zertifikatsdateien zugreifen zu können.

```
sudo chmod u+x /usr/local/bin/renew.sh
sudo /usr/local/bin/renew.sh
```

```
sudo chmod 640 /etc/ssl/private/sdsd.pem
sudo chown root:ssl-cert /etc/ssl/private/sdsd.pem
```

Für die Vergabe der Rechte werden die Befehle *chown*, sowie *chmod* genutzt.

Im letzten Schritt wird das Programm *Certbot* so konfiguriert, dass es das erstellte Skript nutzt, um automatisch das Zertifikat zu aktualisieren. Dafür muss die *Certbot* Konfiguration mit einem Editor abgeändert werden. Diese ist am Pfad */etc/letsencrypt/renewal/app.sdsd-projekt.de.conf* zu finden. Der folgende Wert ist abzuändern.

```
http01_port = 54321
```

Anschließend wird der Prozess mit der erstellten Konfiguration getestet. Dies geschieht mithilfe des folgenden Befehls.

```
sudo certbot renew --dry-run
```

Um die Konfiguration abzuschließen muss nun ein Symlink zum erstellten *renew.sh* erstellt werden. Dazu wird der folgende Befehl genutzt.

```
sudo ln -s /usr/local/bin/renew.sh /etc/letsencrypt/renewal-hooks/deploy/sdsd
```

Mit der Ausführung des letzten Schrittes ist die Konfiguration beendet.

2 Parser- / Dienstentwicklung

In diesem Kapitel wird beschrieben, wie die aufgesetzte SDSD-Plattform erweitert werden kann. Dazu wird im Folgenden auf die Parserentwicklung sowie die Dienstentwicklung eingegangen.

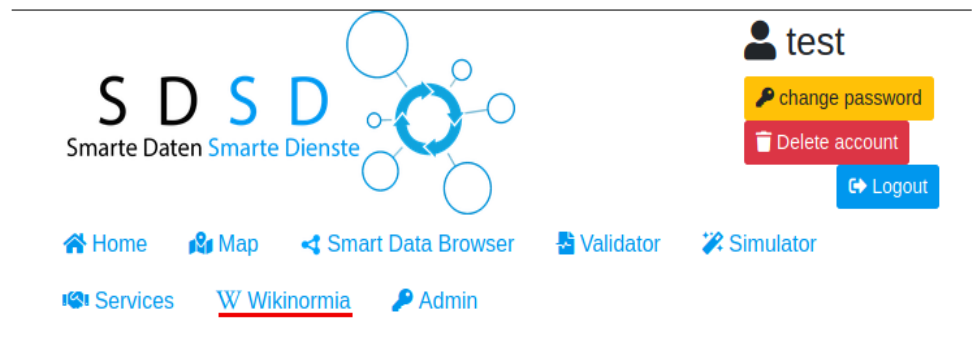
2.1 Parserentwicklung

Im ersten Unterabschnitt steht hier die Parserentwicklung im Vordergrund.

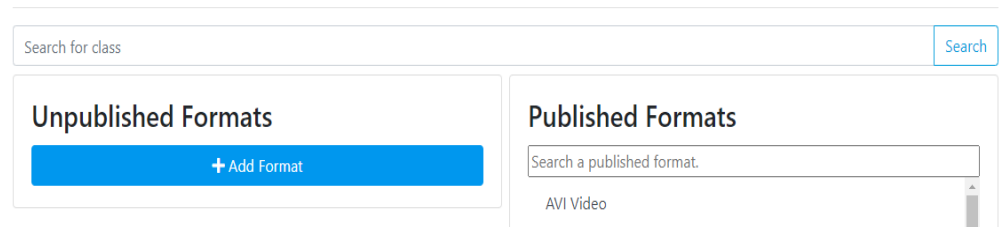
2.1.1 Anlegen von Wikinormia Formaten und Klassen

Zunächst wird hier die Nutzung der Wikinormia erläutert. Die Wikinormia enthält Typen, die mit der SDSD-Plattform genutzt werden können. Soll nun also ein Parser für spezielle Datentypen entwickelt werden, so müssen diese in der Wikinormia angelegt werden. Dazu müssen die folgenden Schritte ausgeführt werden:

1. Wikinormia muss in der Menü-Leiste ausgewählt werden



2. Als Erstes wird nun ein Format angelegt. Dies geschieht durch Hinzufügen eines Unpublished Formats.



3. Nun gelangt man zu einer Ansicht, in der die Daten des Formats in ein Formular eingegeben werden müssen.

Create Format

Wikinormia Draft Mode

Title

The title of the element.

Identifier

The unambiguous name of the resource within its context (base class).

Short Description

A short description that can be displayed in a tooltip.

MIME-Type

The MIME-Type of the format.

agrirouter Message Type

The message type to use when sending the a file via agrirouter.

4. Nach erfolgreichem Ausfüllen des Formulars und abspeichern des Formats, wird als Nächstes die Klasse spezifiziert. Dort wird der Typ eingegeben. Hier können auch mehrere zu einem Format zugehörige Klassen angegeben werden. Über die grünen „+“ Schaltflächen Subklassen, Part-Of Beziehungen, Instanzen und Attribute spezifiziert werden.

teds Publish Edit Delete Wikinormia Draft Mode

Title

The title of the element.

Identifier

The unambiguous name of the resource within its context (base class).

Short Description

A short description that can be displayed in a tooltip.

+ **Subclass of**

Set base types to create a more specific subtype.

+ **Part of**

Set a parent type of which this element is a part of.

Attributes

Identifier	Label	Type	Unit
------------	-------	------	------

Classes

Save Create Delete

Instances

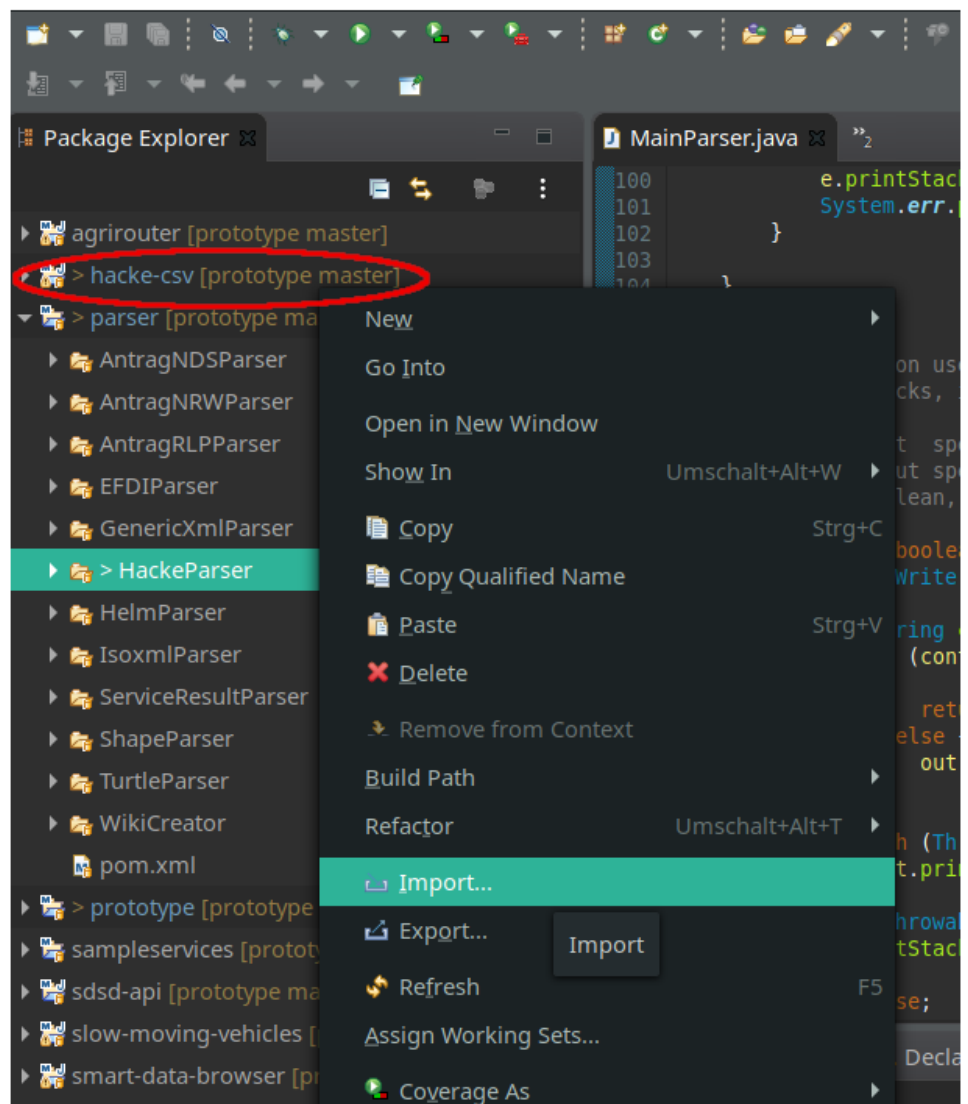
With instances it should be noted that only a **Part of** relationship to another instance may exist if this relationship also exists among the classes of the instances.

Identifier	Label
------------	-------

- Ein Klick auf die „Publish“-Schaltfläche beendet das Anlegen. Sollte das Format damit veröffentlicht werden, so kann es nicht mehr geändert werden. Nutzt man die „Publish“-Schaltfläche nicht, so ist das Format im Draft Mode (engl. „Entwurfsmodus“) gespeichert. Es kann dann später an der Erstellung weitergearbeitet werden.

2.1.2 Erstellung eines Parsers mit Wikinormia Typen

- Im ersten Schritt muss ein neues Maven Projekt im Parser Paket im SDSD Code angelegt werden.
- Wenn ein Ordner erstellt wurde, kann es sein, dass dieser als Maven Projekt in den Eclipse Workspace importiert werden muss. Die ist im folgenden Screenshot markiert.



3. **Achtung:** Wahrscheinlich entstehen Probleme in der Datei *pom.xml*, die behoben werden müssen. Dies ist in den folgenden Schritten beschrieben.
4. Der zu erstellende Parser muss bei *parser/pom.xml* der Datei an der rot-markierten Stelle hinzugefügt werden.

```

parser/pom.xml
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w
3   <modelVersion>4.0.0</modelVersion>
4   <parent>
5     <groupId>de.sdsd.projekt</groupId>
6     <artifactId>sdsd</artifactId>
7     <version>1.0.0</version>
8   </parent>
9
10  <groupId>de.sdsd.projekt.parser</groupId>
11  <artifactId>parser</artifactId>
12  <packaging>pom</packaging>
13  <name>SDSD Parser</name>
14
15  <properties>
16    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
17  </properties>
18
19  <modules>
20    <module>antrag-nds</module>
21    <module>antrag-nrw</module>
22    <module>antrag-rlp</module>
23    <module>efdi</module>
24    <module>genericxml</module>
25    <module>isoxml</module>
26    <module>serviceresult</module>
27    <module>shape</module>
28    <module>ttd</module>
29    <module>wiki-creator</module>
30    <module>hacke-csv</module>
31    <module>helm-csv</module>
32  </modules>
33
34 </project>

```

5. Innerhalb der *pom.xml*-Datei muss die *artifactId* auf den Namen des Moduls gesetzt werden, welches im letzten Schritt hinzugefügt wurde.


```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/POM/4.0.0.xsd">
5      <modelVersion>4.0.0</modelVersion>
6
7      <parent>
8          <groupId>de.sdsd.projekt.parser</groupId>
9          <artifactId>parser</artifactId>
10         <version>1.0.0</version>
11     </parent>
12
13     <artifactId>hacke-csv</artifactId>
14     <packaging>jar</packaging>
15     <name>HackeParser</name>
16
17     <properties>
18         <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
19         <maven.compiler.source>1.8</maven.compiler.source>
20         <maven.compiler.target>1.8</maven.compiler.target>
21     </properties>
22

```

6. Nun muss der Parameter *outputFile* auf einen passenden Wert gesetzt werden.

```

<build>
  <plugins>
    <!-- build fat jar -->
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-shade-plugin</artifactId>
      <version>3.2.1</version>
      <executions>
        <execution>
          <phase>package</phase>
          <goals>
            <goal>shade</goal>
          </goals>
          <configuration>
            <transformers>
              <transformer
                implementation="org.apache.maven.plugins.shade.resource.ServicesResourceTransformer"/>
              <transformer
                implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">
                <manifestEntries>
                  <Main-Class>de.sdsd.projekt.parser.MainParser</Main-Class>
                </manifestEntries>
              </transformer>
            </transformers>
          </configuration>
          <outputFile>${project.build.directory}/hackeParser.jar</outputFile>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>

```

7. Anschließend wird dem Parser Projekt eine *MainParser* Klasse hinzugefügt. Diese muss die folgende Struktur beinhalten:

```

package de.sdsd.projekt.parser;

public class MainParser {
    public static void main(String[] args) throws
    IOException {
        if (args.length > 0) {

```

```

InputStream in = args.length > 1 ?
FileUtils.openInputStream(new File(args[1])) :
System.in;
        OutputStream out = args.length > 2 ?
FileUtils.openOutputStream(new File(args[2])) :
System.out;
        switch (args[0].toLowerCase()) {
        case "parse":
            parse(in, out);
            break;
        case "test":
            System.exit(test(in, out) ? 0 : 1);
            break;
        default:
            System.err.println("No parser
specified ('parse', 'validate', 'test', 'isoxml')");
            break;
        }
    } else
        System.err.println("USAGE: java -jar
parser.jar parse|validate|test|isoxml filepath");
    }
    public static void parse(InputStream input,
OutputStream output) {
        @TODO
    }
    public static boolean test(InputStream input,
OutputStream output) {
        @TODO
    }
}

```

8. Die *MainFunction* verwendet Kommandozeilenparameter, um die zu parsende Datei zu erhalten. Die geparsen Werte werden in *OutputStream* geschrieben. Der *OutputStream* liefert eine ZIP-Datei mit geparsen Daten. Sie können diese Datei verwenden, um zu überprüfen, ob Ihr Parser korrekt arbeitet.
9. Nun wird die Parser Funktion geschrieben. Die Parser API muss verwendet werden, um Daten zu SDSD hinzuzufügen. Die im Java Code verwendeten Modelle müssen mit denen in der Wikinormia erstellten übereinstimmen. Die Fehler müssen in die Fehler Liste geschrieben werden. Dieses wird mithilfe der Funktion *api.setErrors()* realisiert. **Achtung: Geo Daten und TimeLog Daten dürfen NICHT über die *writeTriples* Funktion hinzugefügt werden. Dazu müssen andere Funktionen genutzt werden, die in den anderen Unterabschnitten erläutert werden.** Beispiele für Parser können im Parser Paket gefunden werden. Eine Beispiel Struktur wird im Folgenden gezeigt:

```

public static void parse(InputStream input,

```

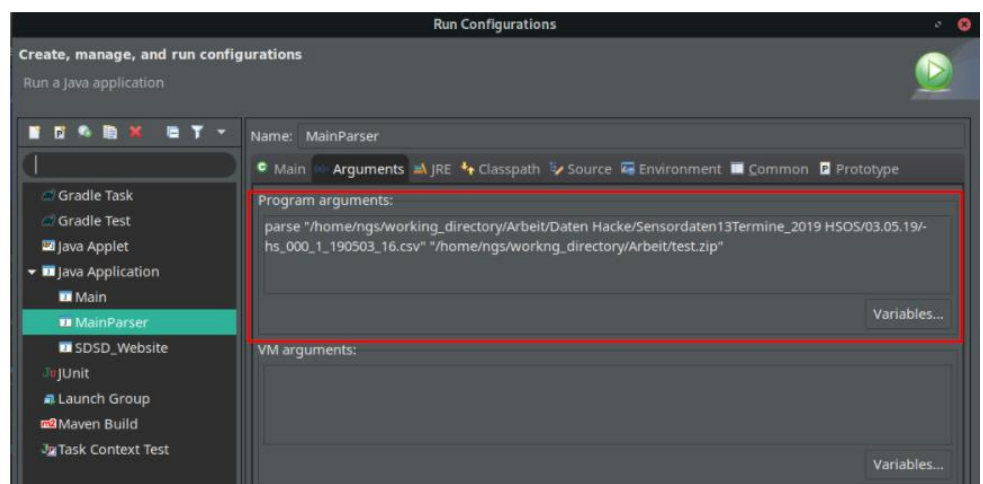
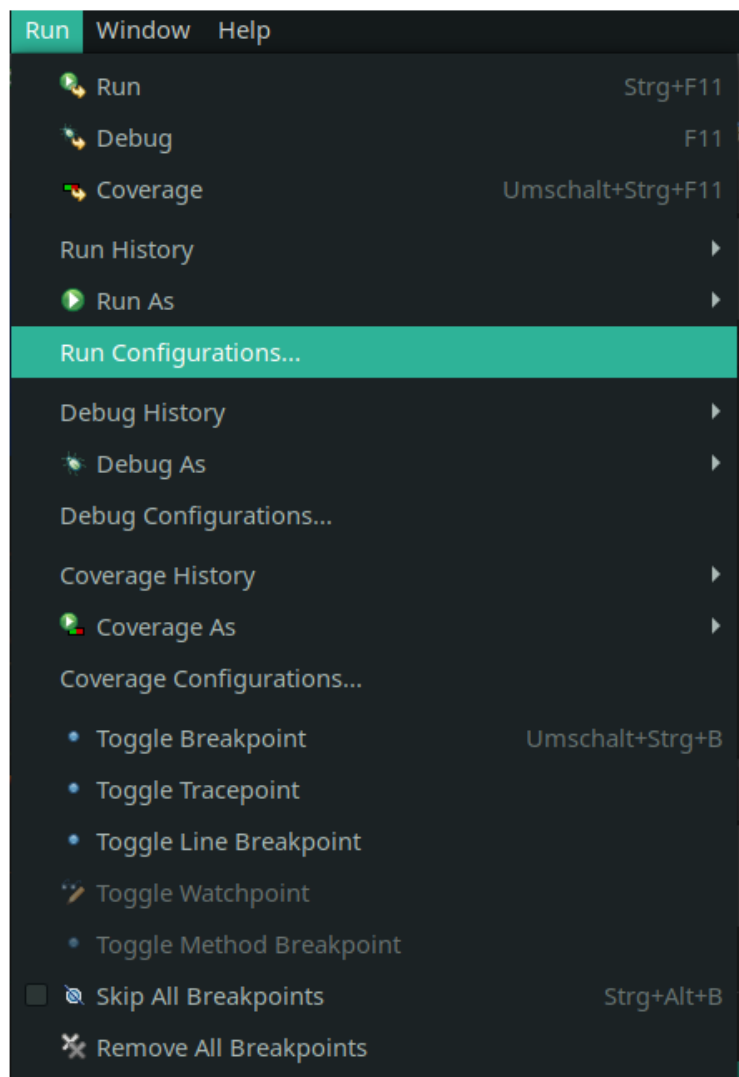
```
OutputStream output) {

    try (ParserAPI api = new ParserAPI(output)) {
        List<String> errors = new ArrayList<>();
        long t1 = System.nanoTime();

        try {
            @TODO
        } catch (Throwable e) {
            e.printStackTrace();
            errors.add(e.getMessage());
        }
        api.setParseTime((System.nanoTime() - t1) /
1000000);
        api.setErrors(errors);
    } catch (Throwable e) {
        e.printStackTrace();
        System.err.println(e.getMessage());
    }

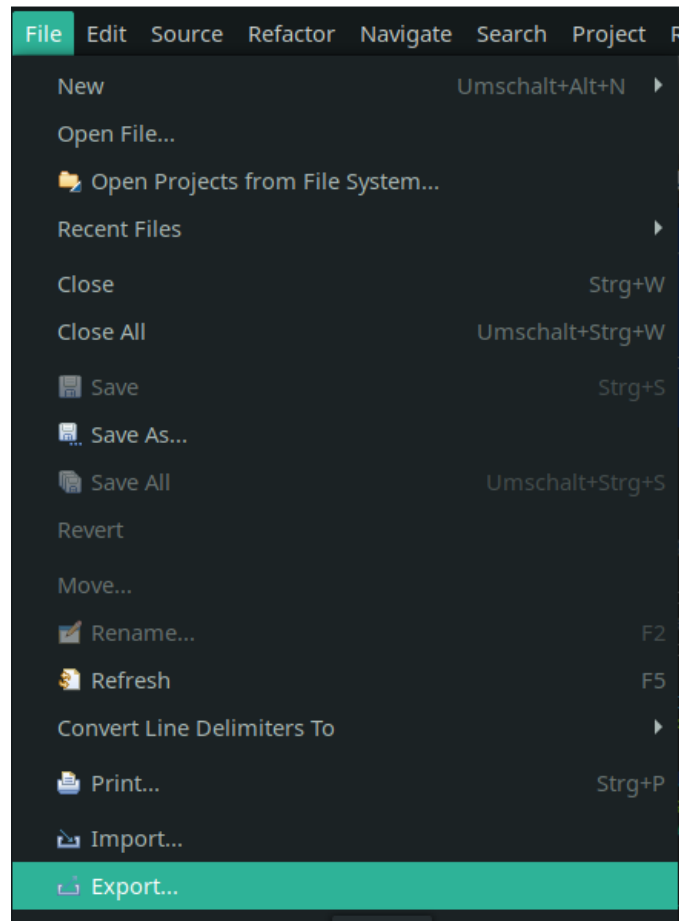
}
```

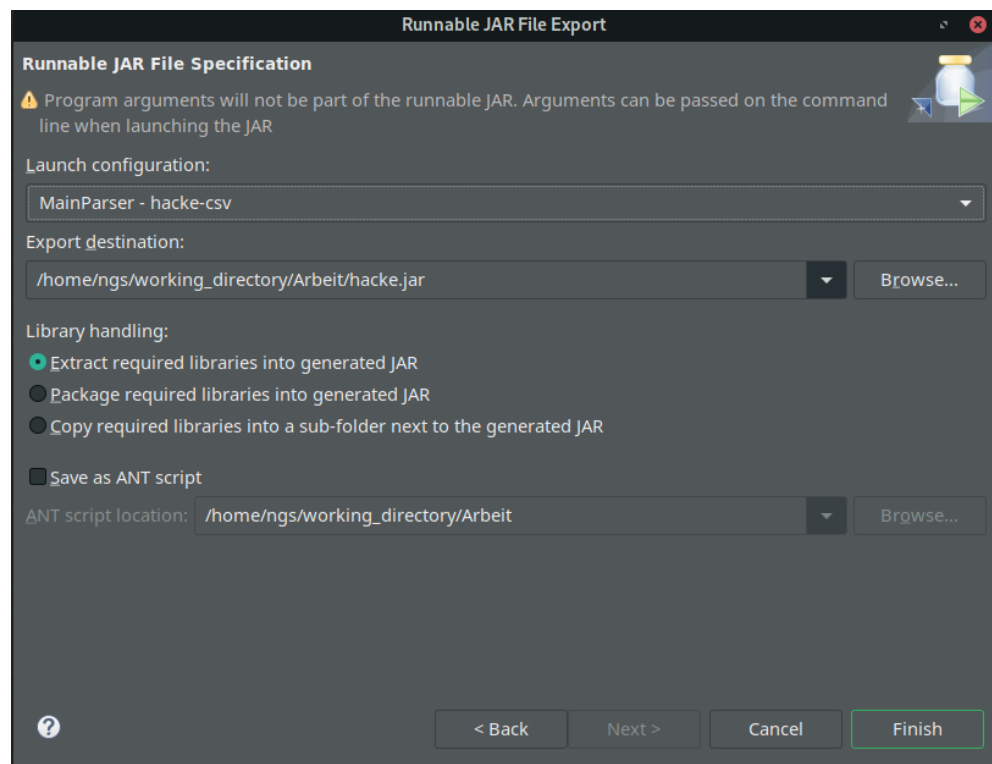
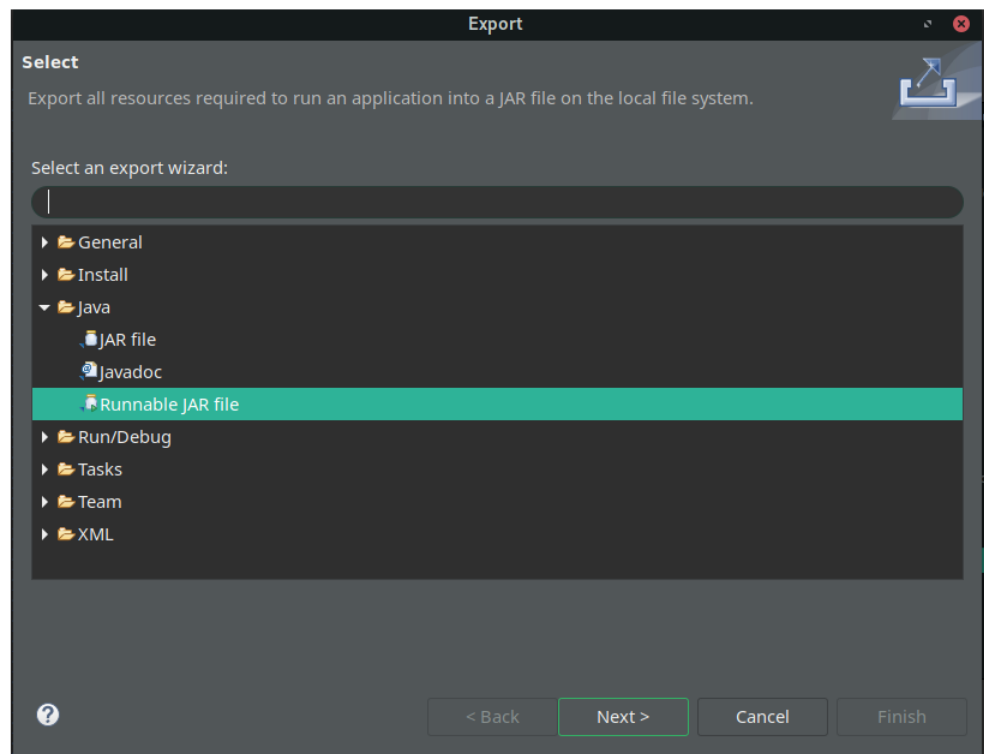
10. Nun muss die Test Funktion geschrieben werden. Diese wird ausgeführt, wenn der SDSD-Server die hochgeladenen Daten nicht erkennen kann. Folglich sollte die Test Funktion die Daten korrekt erkennen.
11. Beide Funktionen müssen nun offline getestet werden. Dazu müssen mithilfe der RunConfigurations in Eclipse die Kommandozeilenparameter gesetzt werden. Um die *parse* Funktion zu testen, muss der Pfad auf "*parse [path to your data] [path to store result ZIP]*" gesetzt werden. Um die Test Funktion zu testen muss der Pfad auf „*test [path to your data]*“ gesetzt werden.



12. Sind die Tests erfolgreich verlaufen, muss der Parser im *JAR*-Format exportiert werden und bei SDSD hochgeladen werden.

Um das Exportieren erfolgreich vorzunehmen, muss innerhalb von Eclipse das File Menu mit der Option Export genutzt werden und der anschließende Assistent ausgeführt werden.





2.1.3 Kreation eines Parsers mit einem Apache Jena RDF Modell

In diesem Abschnitt wird beschrieben, wie ein Apache Jena RDF Modell angelegt werden kann.

Um zu verdeutlichen, wie die Struktur eines RDF Modells aussieht, wird im Folgenden ein kurzes Beispiel gegeben. Es wird angenommen, dass die folgende Datenmenge der Tabelle *Employee* einer Datenbank *CompanyData* in ein RDF Modell konvertiert werden soll.

Id	Name	Age	Birthday	Salary
1	Bob	23	1996-12-24	1420.60

Das zu erzeugende Format in der Wikinormia wäre *CompanyData*. Die korrespondierende Klasse ist *Employee*.

1. Es müssen global alle Wikinormia Ressourcen deklariert werden. Dies geschieht in *WIKI_RESOURCES*. Sie können eine Zuordnung von der Klassenart der Entität zu der entsprechenden Wikinormia-Ressource verwenden. Eine Ressource wird durch den Aufruf von *Util.toWikiResource()* erstellt, die den Namen der Entität an den Wikinormia-Format-Spezifizierer *WIKI_FORMAT* bindet:

```
private static final Resource WIKI_FORMAT =
    Util.toWikiResource("CompanyData");

private static final List<Class<?>> ENTITIES =
    Arrays.asList(Employees.class);

private static final Map<Class<?>, Resource>
    WIKI_RESOURCES = createWikiResources();

private static Map<Class<?>, Resource>
    createWikiResources() {
        HashMap<Class<?>, Resource> wikiResources = new
        HashMap<>();
        for (Class<?> entityClass : ENTITIES) {
            String entityName =
            entityClass.getSimpleName();
            wikiResources.put(entityClass,
            Util.toWikiResource(WIKI_FORMAT, entityName));
        }
        return wikiResources;
    }
}
```

2. Nun wird ein leeres Jena RDF Modell angelegt:

```
import org.apache.jena.rdf.model.Model;
import org.apache.jena.rdf.model.ModelFactory;
Model model = ModelFactory.createDefaultModel();
```

3. Nun wird zufällig eine global zu verwendende URI für jede Zeile der Daten generiert. Da im verwendeten Beispiel nur eine Zeile vorhanden ist, wird nur eine URI generiert. Dieser Ressourcen Identifier ersetzt das ID Attribut der originalen *Employee* Tabelle. Es ist zu beachten, dass der dritte Parameter, der hier *null* ist, eine Beziehung (*DCTerms.isPartOf*) zu einer Elternressource darstellt.

```
Resource entityUriResource =
Util.createRandomUriResource(model, entityResource,
null);
```

4. Hat eine Zeile ein beschreibendes Attribut, kann dieses als ein Label für den Jena Eintrag genutzt werden. Ein solches Label *RDFS.Label* ist an die URI der aktuellen Daten-Zeile gebunden:

```
entityUriResource.addLiteral(RDFS.label,
ResourceFactory.createTypedLiteral("Bob"));
```

5. Iterieren Sie über alle Attribute *attrName* einer Datenzeile und ordnen Sie die Attributwerte *attrValue* den Java-Datentypen zu. Verwenden Sie für numerische Typen die Wrapper-Klassen *Integer*, *Double* usw. anstelle der eingebauten primitiven Typen wie *int* und *double*. Denken Sie daran, dass mit 0 initialisierte Variablen einen akzeptablen Attributwert enthalten können, obwohl der ursprüngliche Wert fehlte. *Null*-Werte hingegen weisen auf ein fehlendes Attribut hin. Versuchen Sie nicht, ein RDF-Literal aus einem fehlenden und daher *null*-wertigen Attribut zu erstellen; überspringen Sie diese stattdessen. Jena wird mit fehlenden Attributwerten korrekt umgehen, wenn Sie diese nicht zum RDF-Modell hinzufügen. Bitte achten Sie auf Zeilenattribute vom Typ Datum, da diese von der überladenen *ResourceFactory.createTypedLiteral()*-Funktion nicht korrekt behandelt werden. Für die gängigen Datentypen kann *Util.lit()* genutzt werden.

```
import org.apache.jena.rdf.model.Literal;
import org.apache.jena.rdf.model.Property;
import org.apache.jena.rdf.model.Resource;
import org.apache.jena.rdf.model.ResourceFactory;

import de.sdsd.projekt.api.Util;
import de.sdsd.projekt.parser.data.Employee;
Resource entityResource =
WIKI_RESOURCES.get(Employee.class);

for (...) {
if (attrValue == null)
continue;
Literal typedLiteral;
```



```

        if (attrValue instanceof Date)
            typedLiteral = Util.lit(((Date)
attrValue).toInstant());
        else
            typedLiteral =
ResourceFactory.createTypedLiteral(attrValue);

        Property wikiProperty =
Util.toWikiProperty(entityResource, attrName);
        entityUriResource.addProperty(wikiProperty,
typedLiteral);
    }

```

Nachdem das Literal `typedLiteral` erfolgreich erstellt wurde, müssen wir eine Wikinormia-Eigenschaft generieren, indem wir die Entitätsressource `Employee` und den Namen `attrName` des aktuellen Attributs verwenden. Im nächsten Schritt verknüpfen wir das Literal `typedLiteral` mit der `wikiProperty` (aka. Tabellenspalte) und fügen dieses (Spalte, Wert) Paar zum URI-Ressourcen-Identifikator `entityUriResource` des aktuellen Jena-RDF-Eintrags hinzu.

Für unser einzeliliges Beispiel aus unserer Mitarbeitertabelle ist der oben beschriebene Prozess nachfolgend dargestellt:

```

Resource entityResource =
WIKI_RESOURCES.get(Employee.class);
Property nameProperty =
Util.toWikiProperty(entityResource, "Name");
Property ageProperty =
Util.toWikiProperty(entityResource, "Age");
Property birthdayProperty =
Util.toWikiProperty(entityResource, "Birthday");
Property salaryProperty =
Util.toWikiProperty(entityResource, "Salary");

Literal nameLit =
ResourceFactory.createTypedLiteral("Bob");
Literal ageLit =
ResourceFactory.createTypedLiteral(23);
Literal birthdayLit = Util.lit(Instant.parse("1996-12-
24"));
Literal salaryLit =
ResourceFactory.createTypedLiteral(1420.60);

entityUriResource.addProperty(nameProperty, nameLit);
entityUriResource.addProperty(ageProperty, ageLit);
entityUriResource.addProperty(birthdayProperty,
birthdayLit);
entityUriResource.addProperty(salaryProperty,
salaryLit);

```

6. Im letzten Schritt kann nun die *ParserAPI* Funktion *writeTriples* zum Schreiben des Jena RDF Modells in einen ZIP Stream genutzt werden.

```
import de.sdsd.projekt.api.ParserAPI;
try (ParserAPI api = new ParserAPI(output)) {
    api.writeTriples(model);
}
```

2.1.4 Geodaten in einem Parser verarbeiten

Wie bereits im vorherigen Unterabschnitt erwähnt, müssen Geodaten, wenn sie auf der SDSD-Plattform eingesetzt werden, auf andere Art und Weise verarbeitet werden.

Werden Geodaten verarbeitet, so sollten sie im GeoJSON Format zur Verfügung stehen. Dabei wird für Repräsentation der Geodaten die JS-Object Notation genutzt. Es existieren in GeoJSON sowohl einfache Geometrien als auch mehrteilige Geometrien. Es können die folgenden Datentypen repräsentiert werden:

- Punkte (Point)
- Linien (LineString)
- Polygone (Polygon)
- MultiPoint
- MultiLineString
- MultiPolygon

Wird GeoJSON verwendet, so ist außerdem möglich, in GeoJSON *Properties* für ein GeoJSON Objekt zu setzen. Damit können zusätzliche Informationen gespeichert werden. Für die Anzeige mit der SDSD-Plattform dürfen Properties nicht verschachtelt werden. Weiterhin ist bei der Repräsentation in GeoJSON zu beachten, dass die Koordinaten im WGS84 Format vorliegen.

Um die Geodaten, nun an SDSD senden zu können muss die Klasse *GeoWriter* genutzt werden. Diese stellt die Funktionen *writeFeature* und *writeGeometries* zur Verfügung. Es ist wichtig, dass erzeugte *GeoWriter* Objekt am Ende der Ausführung wieder zu schließen. Ein aus dem *AntragsparserNDS* entnommenes Beispiel kann im Folgenden eingesehen werden.

```
try (GeoWriter geo = api.writeGeo()) {
    for(JSONObject feature : features) {
```

```

        geo.writeFeature(feature, ElementType.Field,
        feature.getString("id"), feature.getString("label"));
    }
}

```

Es wird nun dargestellt, wozu der *ElementType* eingesetzt wird. Dieser muss passend zum Einsatz der Geodaten gewählt werden. SDSD stellt dafür die folgenden Typen zur Verfügung:

- Other: Kein anderer Typ kann spezifiziert werden
- TimeLog: Sollte zur Parserprogrammierung vernachlässigt werden. Es handelt sich um einen von SDSD intern genutzten Elementtypen.
- Field: Angabe von Feldgrenzen
- TreatmentZone: Für Applikationskarten
- GuidancePattern: Ein Guidance Pattern gibt an, wo eine Fahrspur gestartet wird. Die weitere Route ergibt sich dann aus weiteren Parametern, wie Feldbreite, etc.
- FieldAccess: FieldAccess gibt an, wo sich ein befahrbarer Zugang zum genutzten Feld befindet.

Weiterhin ist die korrekte Verwendung einer URI zu beachten. Die URI ist ein Pflichtparameter. Sie stellt den Bezug zum TripleStore her. Folglich sollte die URI des Elements angegeben werden, dass die entsprechende Verknüpfung im TripleStore darstellt.

2.1.5 TimeLogs im Parser verarbeiten

In diesem Abschnitt wird erläutert, wie TimeLog Werte in einem Parser eingesetzt werden können.

Bei TimeLog Werten handelt es sich um zeit- und positionsbezogene Werte. Eng mit TimeLog Werten verknüpft sind ValueInfos. Diese sind am besten wie folgt zu charakterisieren: Stellt man sich die TimeLog Werte als Tabelle vor, so sind die ValueInfos als Spaltenbeschreibungen zu charakterisieren.

Um TimeLog Werte in einem Parser zu verarbeiten, sind mehrere Schritte auszuführen.

1. Im ersten Schritt muss ein TimeLog Objekt erstellt werden. Dies kann entweder anhand eines bereits bestehenden Objektes erfolgen, oder ein neues Objekt wird angelegt. Im folgenden Code Ausschnitt ist die Erstellung eines neuen Objektes zu sehen. Es stammt aus dem für die Beispiele verwendeten HackeParser. Der Name muss eindeutig sein. Außerdem sollte ein Start- und Endzeitpunkt angegeben

werden. Weiterhin sollte die Anzahl der Zeilen angegeben werden.

```
TimeLog timelog = new TimeLog(Util.createRandomUri(),
    TLGNAME, Instant.ofEpochMilli(hdp.get(0).getTime()),
    Instant.ofEpochMilli(hdp.get(hdp.size() -
    1).getTime()), hdp.size());
```

2. Nun muss ein ValueInfos Objekt erstellt werden. Dieses benötigt neben einer URI, dem zugehörigen TimeLog und einem Designator einen Scale, einen Offset, Number of Decimals und eine Einheit. Der Scale, der Offset und die Number of Decimals dienen dazu, aus einem Integer Wert, einen Fließkommawert herzustellen. Die Formel dazu lautet:

$$((IntegerWert + Offset) \cdot scale)$$

Dies kann beispielsweise von Vorteil sein, wenn eine Temperatur eingegeben wird. Der Integer Wert stellt dabei die Temperatur in der Einheit Kelvin dar. Die obenstehende Formel kann dann in der Kombination mit korrektem Offset und Scale zur Berechnung des zugehörigen Wertes in Grad Celsius dienen. Auch hier ist der folgende exemplarische Code Ausschnitt dem Hacke Parser entnommen.

```
ValueInfo mocot = new ValueInfo(Util.createRandomUri())

    .addTimeLog(timelog)

    .setDesignator(MOCOT)

    .setScale(SCALE)

    .setNumberOfDecimals(4)

    .setUnit("%");
```

3. Im letzten Schritt muss nun, wie aus den vorherigen Beispielen bekannt der *TimeLogWriter* genutzt werden. Dieser wird erstellt und es kann mithilfe von *tlw.write* die entsprechende Zeile geschrieben werden.

```
try (TimeLogWriter tlw = api.addTimeLog(timelog,
    Arrays.asList(mocot, dicot))) {
    Long[] values = new Long[2];
    for (HackeData hd : hdp) {
        if (hd.getTime() != null && hd.getLat() !=
        null && hd.getLon() != null && hd.getAlt() != null) {

            values[0] = toValue(hd.getMocot());

            values[1] = toValue(hd.getDicot());
```

```

        tlw.write(Instant.ofEpochMilli(hd.getTime()),
hd.getLat(), hd.getLon(), hd.getAlt(), values);

    }
}

```

Analog funktioniert das Hinzufügen von Grids.

2.2 Entwicklung eines Dienstes

In diesem Abschnitt wird die Entwicklung eines Dienstes für die SDSD-Plattform beschrieben. Ein Dienst bezeichnet eine externe Anwendung. Diese kann Zugriff auf Daten eines Benutzers der Plattform erhalten und außerdem spezielle Ergebnisse zurückschreiben. Somit bilden die Dienste „Apps“ zur Erweiterung der Plattform.

Es wird zunächst damit begonnen, den Lebenszyklus eines Dienstes zu erläutern. Am Anfang des Lebenszyklus wird ein Dienst durch den Benutzer aktiviert. Der Dienst wartet anschließend auf die Eingabe von Parametern und eventuellen Zugriffsberechtigungen. Ist die Eingabe getätigt, wechselt der Dienst in den Status aktiv. Führt er seine Arbeit erfolgreich aus, wechselt er in den Zustand beendet. Wird er jedoch vom Benutzer gelöscht oder abgebrochen, wechselt er in einen entsprechenden Zustand.

Um nun einen Dienst zu erstellen, muss dieser zunächst auf der SDSD-Website angelegt werden, um einen ServiceToken zu erhalten. Dies geschieht durch Auswahl von Services im Menü der SDSD-Plattform. Dort wird „+ Own Services“ ausgewählt.

 Services

[+ Own Services](#)

Auf der folgenden Seite kann nun ein Service über das untenstehende Formular angelegt werden. Der Token ist nach erfolgreichen Anlegen über die „Copy Token“ Schaltfläche zu erhalten.

+ Create Service

Service Name:

+ Parameter

Set parameters the server need to know from the user.

+ Access

Set the Wikinormia data types the service needs to access.

+ Create

Nun wird sich der eigentlichen Programmierung des Dienstes zugewandt. Es sollte sich dazu am *ExampleService* Paket orientiert werden. Die Datei *Main.java* kann direkt in den selbst programmierten Dienst übernommen werden. Dort ist **nur** der erzeugte Service Token einzutragen. Der restliche Inhalt des Programmcodes dieser Datei bleibt unverändert. Das verwendete *system.in.read()* dient in der *Main.java* Datei dazu eine Endlosschleife zur Ausführung des Dienstes zu erzeugen.

Anschließend wird sich der eigentlichen Service Datei zugewandt. Diese sollte ähnlich der *ExampleService.java* aufgebaut sein. Der verwendete Konstruktor muss beim Parameter *local* auf *false* setzen. Sollte dies nicht passieren, wird angenommen, dass die SDSD-Plattform auf dem lokalen Klienten läuft. In der Funktion *runTaskForInstance* wird spezifiziert, was der Dienst ausführen soll. Über *instSetError* ist es möglich dem Nutzer im Fehlerfall Fehlermeldungen anzeigen. Abschließend wird *inst.complete* ausgeführt, um die Ausführung des Dienstes zu beenden. Es muss generell bei der Entwicklung beachtet werden, dass Dienste auch parallel ausgeführt werden können und damit der Zugriff auf die Daten entsprechend geregelt worden sein muss.

3 Resultate

Zusammenfassend lässt sich sagen, dass in diesem Dokument erläutert wurde, wie die SDSD-Plattform in Betrieb genommen werden kann und erweitert werden kann.

Zur Inbetriebnahme sind auf einem dedizierten Server die unterschiedlichen Datenbankmanagementsysteme MongoDB, Redis, Cassandra und Stardog zu installieren. Außerdem müssen für die einzelnen Systeme Konfigurationen vorgenommen werden. Zum Beispiel müssen für Cassandra CQLSH Skripts ausgeführt werden, die dazu dienen in der Datenbank die für die SDSD-Plattform benötigten Strukturen anzulegen. Ist die Konfiguration der Datenbankmanagementsysteme abgeschlossen, so muss eine eigene SDSD-Konfigurationsdatei erstellt werden. Diese dient dazu, die eigenen Zugangsdaten bereitzuhalten. Ist dies geschehen, kann der Server mit der entsprechenden JAR-Datei gestartet werden. Abschließend wird eine SSL-Verschlüsselung in Betrieb genommen, um die Kommunikation absichern zu können. Um die Plattform zu erweitern können nun Parser und Dienste entwickelt werden. Die Parser können auf unterschiedliche Art und Weisen angelegt werden, die in diesem Dokument behandelt wurden.

Bei den Diensten handelt es sich um externe Programme, die Zugriff auf die Daten eines Benutzers erhalten und Aktionen damit ausführen, sowie Ergebnisse zurückschreiben. Zur Erstellung eines solchen Dienstes sollte sich idealerweise am ExampleService Paket orientiert werden.