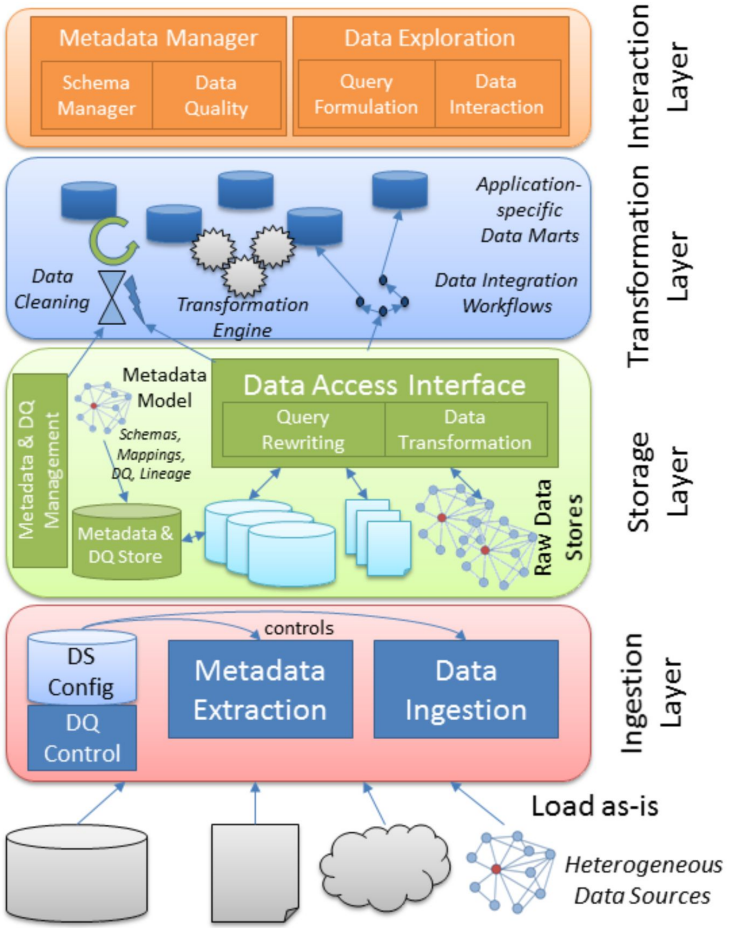
Semantic Data Lake

- Members:
  1. Sayed Hoseini
  2. Tobias Claas
  3. Maher Fallouh
  4. Abdullah Zaid
  5. Muhammad Noman

Table of Contents

Introduction

- Big Data
- What is a data lake?
- Metadata Management
- Data Lake Architecture
  - Storage Layer
  - Interaction Layer
  - Transformation Layer
  - Ingestion Layer



M. Jarke and C. Quix. "On warehouses, lakes, and spaces – the changing role of conceptual modeling for data integration.". In: Conceptual Modeling Perspectives (2017), pp. 231–245.

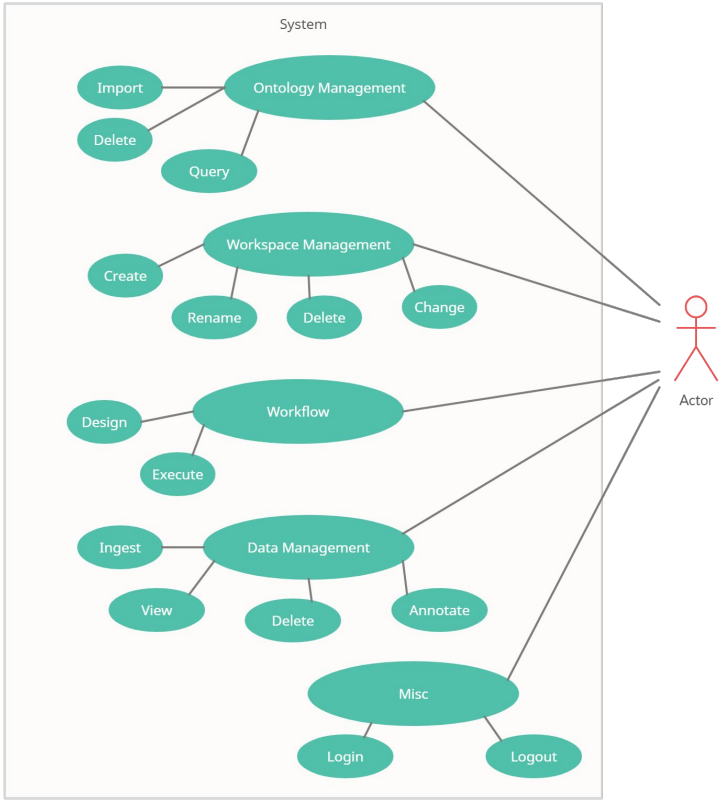
Prototype

- Took over Prototype
- Functionality
  - Backend in Python with Flask
  - Frontend written in Angular, decided to start from scratch in React
  - Using Docker Containers to launch Spark- & Hadoop-Cluster and DataBases (Postgres + MongoDB)
  - MetaData Model -> DataMart Abstraction
  - Authentication
  - Mostly Ingestion Layer

Our Addition

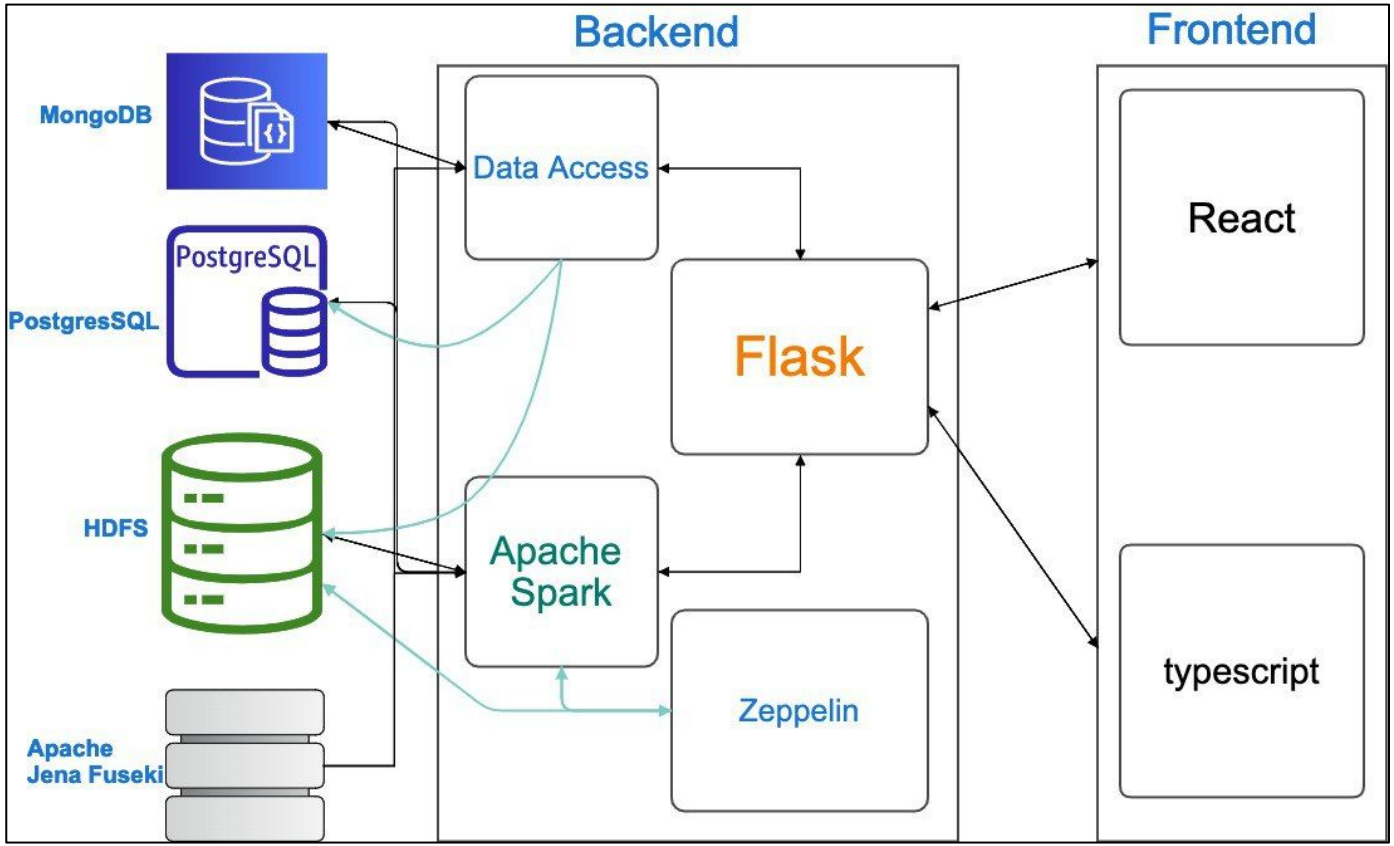
- Add Semantic Information to Data
- Graphical WorkFlow for Data Transformations written with ReactFlow
- New Frontend written with React and Material UI
- Workspace Abstraction
- Integrated Apache Zeppelin to the System
- Datamart Deletion
- ER-Diagram and Documentation

Use Case Diagram



Slide Credit: Maher Fallouh

Architecture



Slide Credit: Abdullah Zaid

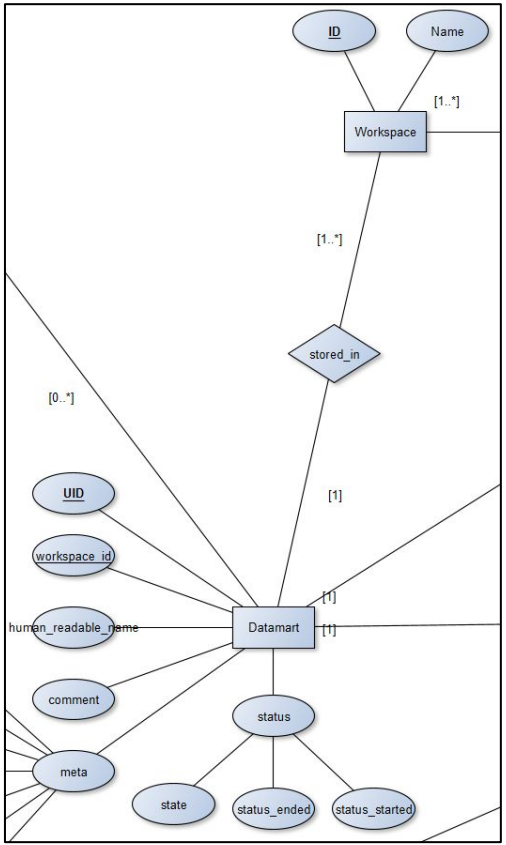
Docker Containers

- Hadoop Cluster (7 Containers)
- Spark Cluster (3 Containers: Master + 2 Workers)
- MongoDB
- Postgres
- Fuseki
- Zeppelin (4,11 GB)
- Data Lake Application (1 Container, 2 GB)

Total: 15 Containers, ~15 GB

Workspaces

- Projects are separated via Workspace
- Each DataMart is associated to a single Workspace
- Workspaces are physically separated as well, i.e. one HDFS directory, MongoDB-, Postgres- and Fuseki-Database per Workspace
- Users can create Workspaces and have access to Data associated to their Workspaces only



GET http://127.0.0.1:5000/workspaces/60dad3807e930175aa99c261/datamarts

Annotations

Ontologies - General

- Collection of Descriptions and labels
  - Usually designed for a specific Domain and Things
  - Can be looked up online
    - Shareable & Common understanding
- Easier understanding of the Data
  - Due to detailed descriptions and additional information like units
- Allows for automated processing or merging of datasets
  - E.g. Merging Datasets

Ontologies

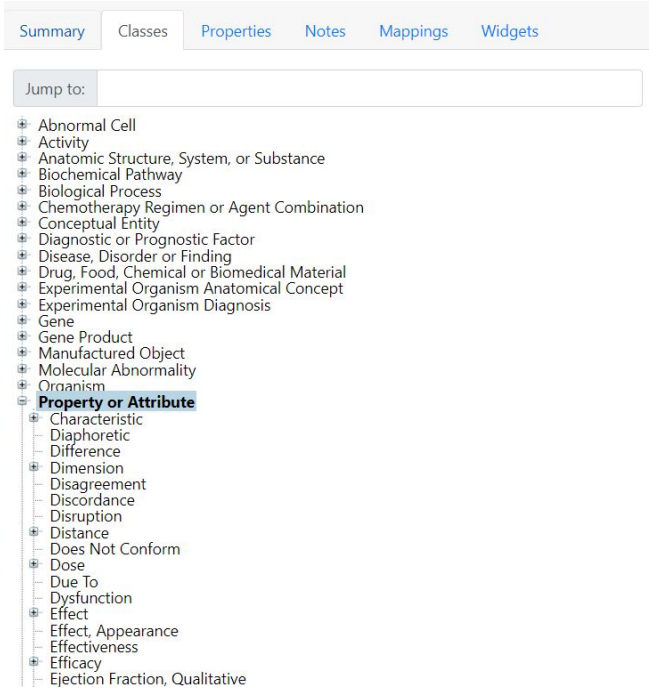
- System needs Ontologies
- Stored in Apache Jena Fuseki
  - Workspaces has local Ontologies
  - Each Ontology is stored in a Named Graph
    - To enable deletion
  - Default Graph is a Union-Graph
    - Ease of computation
- Standard Ontology
  - Good starting point
  - Requirements
    - Basic Vocabulary
    - Not to big or specific

Standard Ontologie

- Subset of National Cancer Institute Thesaurus(NCIT)
  - Entire Ontology very broad and with special focus
    - See figure on the right
  - Property or Attribute Subset
- Can be deleted if not desired
- Treelike Class Hierarchy

Metrics

Classes	165,673
Individuals	0
Properties	97
Maximum depth	21
Maximum number of children	3,235
Average number of children	6
Classes with a single child	11,365
Classes with more than 25 children	1,152
Classes with no definition	36,836



Standard Ontology - Extraction

- Property or Attribute Subset contains
  - All Subclasses and
  - all labels of referenced Classes
  - Degree 1 of referencing
- Reduced size from 600MB to 6MB

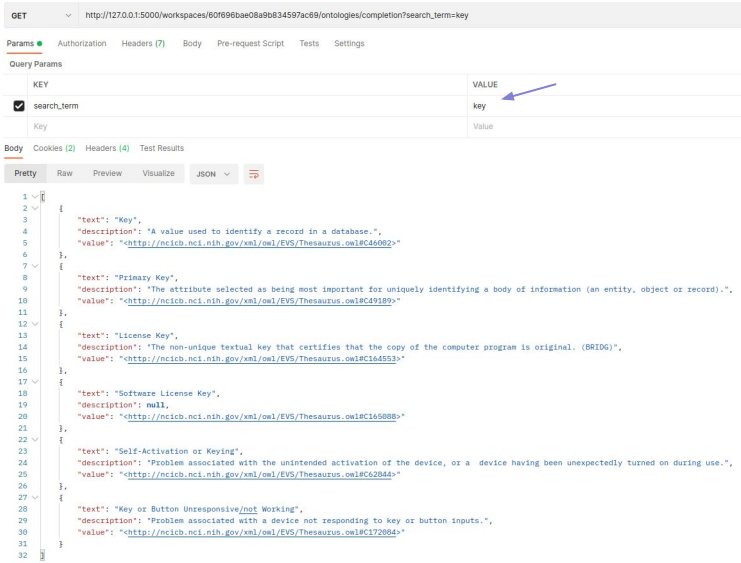
```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ncid: <http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#>
CONSTRUCT {
    ?x ?a ?b .
    ?a rdfs:label ?l1 .
    ?b rdfs:label ?l2 .
} WHERE {
    {
        ?x (rdfs:subClassOf)* ?poa ;
            ?a ?b .
        OPTIONAL {?a rdfs:label ?l1 .}
        OPTIONAL {?b rdfs:label ?l2 .}
        FILTER(?poa = ncid:C20189)
    }
}""")
```

```
167964  ncit:C20189 a owl:Class ;
167965      rdfs:label "Property or Attribute" ;
167966      ncit:A8 ncit:C90259 ;
167967      ncit:NHC0 "C20189" ;
167968      ncit:P106 "Conceptual Entity" ;
167969      ncit:P108 "Property or Attribute" ;
167970      ncit:P207 "C1514495" ;
167971      ncit:P322 "NICHD" ;
167972      ncit:P366 "Properties_or_Attributes" ;
167973      ncit:P90 "Property or Attribute" ;
167974      ncit:P97 "A distinguishing quality or prominent aspect of a person, object, action, process, or substance." ;
167975      owl:disjointWith ncit:C20633,
167976          ncit:C22187,
167977          ncit:C22188,
167978          ncit:C26548,
167979          ncit:C28428,
167980          ncit:C3910,
167981          ncit:C43431,
167982          ncit:C7057,
167983          ncit:C97325 .
167984
167985  ncit:C48470 a owl:Class ;
167986      rdfs:label "Potency Unit" ;
167987      ncit:NHC0 "C48470" ;
167988      ncit:P106 "Quantitative Concept" ;
```

Auto-Completion

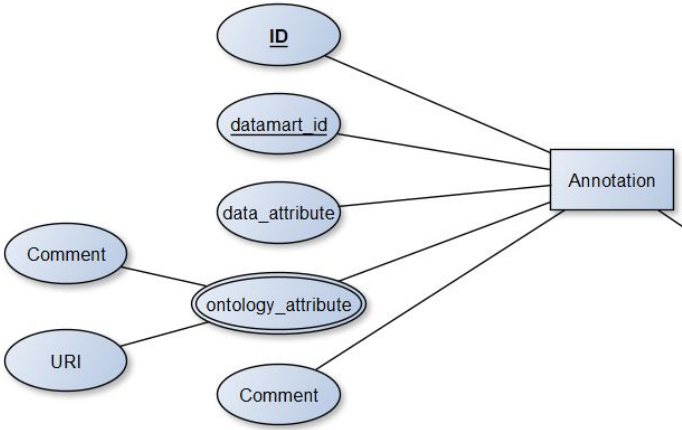
- Suggestion feature for Annotating with Uploaded Ontologies
- Function that will query the Uniongraph in Fuseki
  - Will look for Attributes with the Label
  - Or Classes with these Name

```
querystring = """
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
    PREFIX ncit: <http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#>
    SELECT ?subject ?label ?desc
    WHERE {
      ?subject a ?x ;
        rdfs:label ?label .
      OPTIONAL {?subject ncit:P97 ?desc .}
      FILTER (regex(str(?subject), '#""" + search_term + """', 'i') ||
        regex(?label, '""" + search_term + """', 'i'))
    }
    ORDER BY strlen(?label)
    LIMIT 20 """
p = post('http://localhost:3030/' + workspace_id,
         auth=(settings.Settings().fuseki_storage.user, settings.Settings().fuseki_storage.password),
         data={'query': querystring})
```

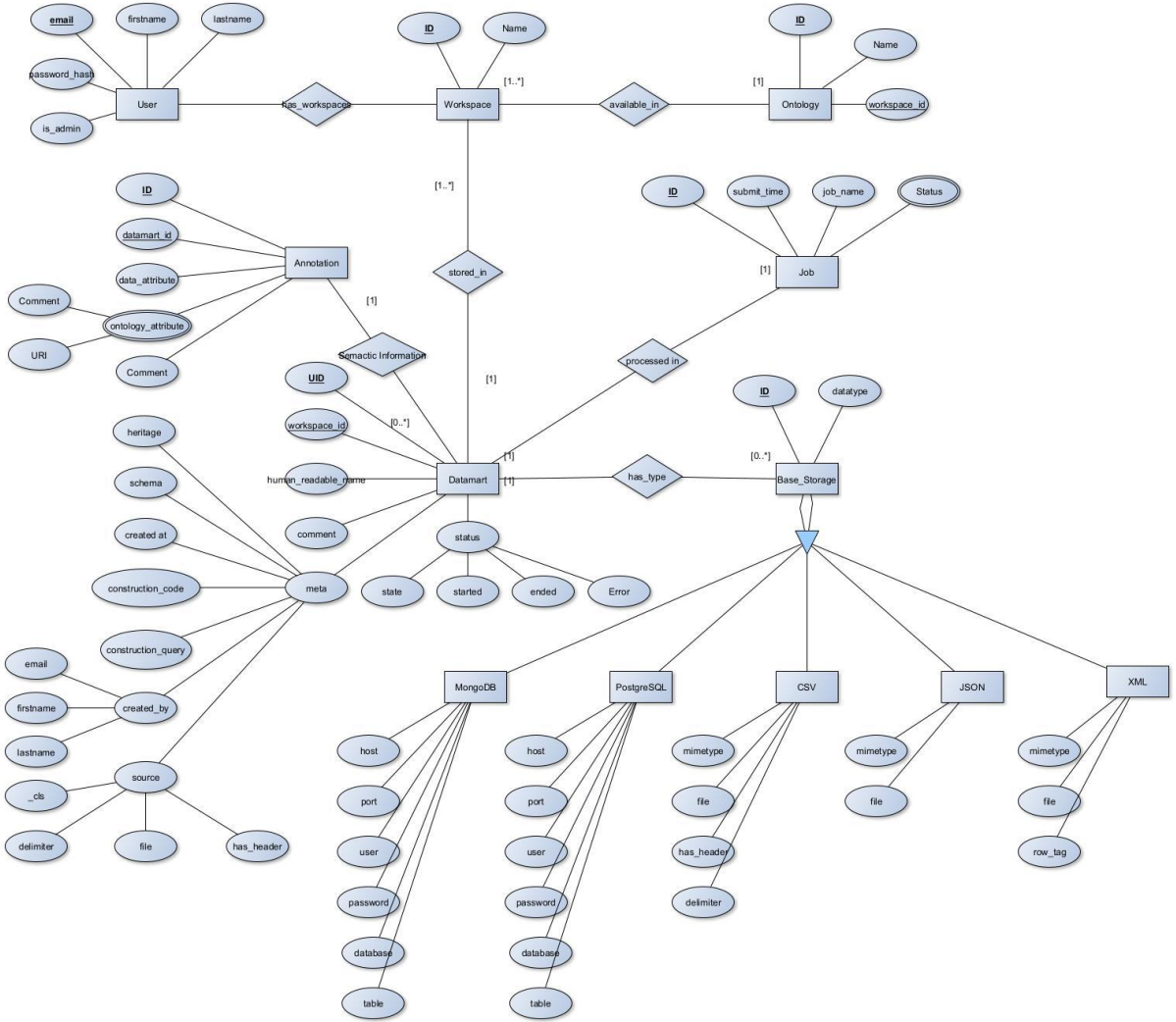


Annotation

- Bring Data and Ontologies together
- Stored in MongoDB
  - Associated to Datamarts & Ontologies
  - Multiple annotations for one data_attribute in one entry
- Integrity Checks
  - Datamart_id with MongoDB referencing
  - Ontology-attribute with ASK-Query
  - Data_attribute exists due to selection in frontend



ER Diagram



Semantic Data Integration - Summer

WorkFlow

What is the Workflow good for?

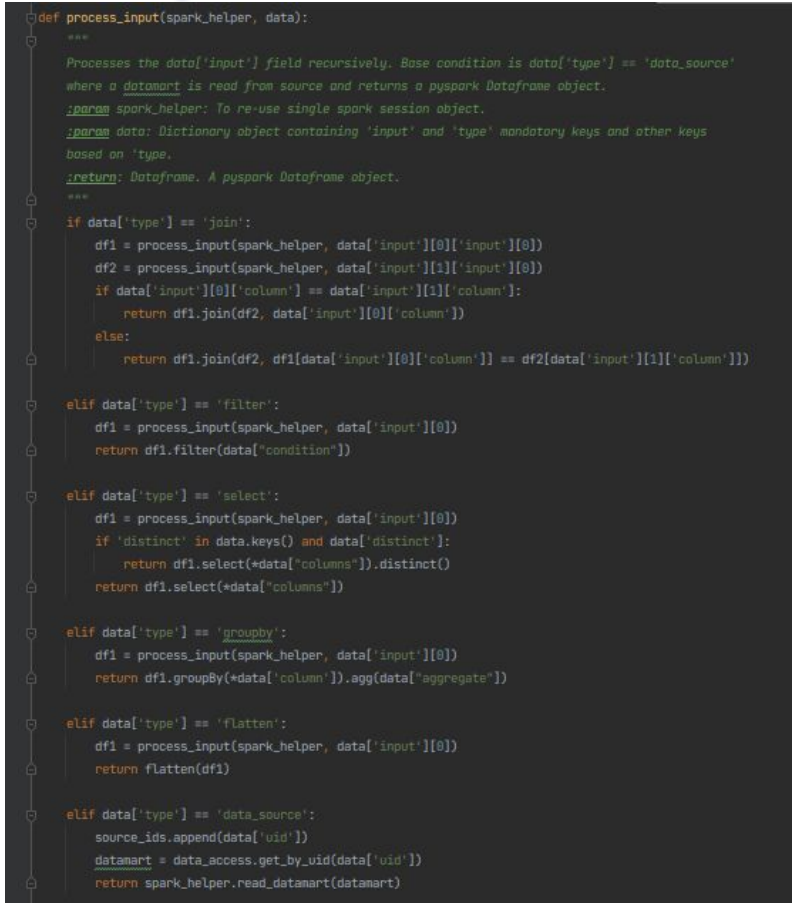
- Graphical interface
- Helpful for Non-technical people
- No SQL knowledge required
- End-to-end pipeline for data transformation

Workflow - Proof of Concept

- Part of Transformation layer
- Backend Data Abstraction: Datamart -> Dataframe (pyspark)
- Workflow diagram converted to JSON
- Execute in single Spark session (recursively)
- The code written is extendable to more operations
- Basic smart suggestion using Annotations (DEMO)

Workflow - Proof of Concept

```
{
  "type":"output",
  "name":"exported.csv",
  "target":"MongoDB",
  "input":[
    {
      "type":"filter",
      "condition":"Identifier= \"9012\" ",
      "input":[
        {
          "type":"select",
          "columns":[
            "Identifier",
            "Access code",
            "Recovery code",
            "First name2",
            "Last name2",
            "Department",
            "Location"
          ],
          "input":[
            {
              "type":"join",
              "input":[
                {
                  "column":"Identifier",
                  "input":[
                    {
                      "type":"source",
                      "id":"e22d832d-4670-4382-b715-836a408fec10"
                    }
                  ]
                },
                {
                  "column":"Identifier",
                  "input":[
                    {
                      "type":"source",
                      "id":"bb0b6f46-36f6-4d14-8d0f-026ebdf6d084"
                    }
                  ]
                }
              ]
            }
          ]
        }
      ]
    }
  ]
}
```



```python
def process_input(spark_helper, data):
    """
    Processes the data['input'] field recursively. Base condition is data['type'] == 'data_source'
    where a datamart is read from source and returns a pyspark Dataframe object.
    :param spark_helper: To re-use single spark session object.
    :param data: Dictionary object containing 'input' and 'type' mandatory keys and other keys
    based on 'type'.
    :return: Dataframe. A pyspark Dataframe object.
    """
    if data['type'] == 'join':
        df1 = process_input(spark_helper, data['input'][0]['input'][0])
        df2 = process_input(spark_helper, data['input'][1]['input'][0])
        if data['input'][0]['column'] == data['input'][1]['column']:
            return df1.join(df2, data['input'][0]['column'])
        else:
            return df1.join(df2, df1[data['input'][0]['column']] == df2[data['input'][1]['column']])

    elif data['type'] == 'filter':
        df1 = process_input(spark_helper, data['input'][0])
        return df1.filter(data["condition"])

    elif data['type'] == 'select':
        df1 = process_input(spark_helper, data['input'][0])
        if 'distinct' in data.keys() and data['distinct']:
            return df1.select(*data["columns"]).distinct()
        return df1.select(*data["columns"])

    elif data['type'] == 'groupby':
        df1 = process_input(spark_helper, data['input'][0])
        return df1.groupBy(*data['column']).agg(data["aggregate"])

    elif data['type'] == 'flatten':
        df1 = process_input(spark_helper, data['input'][0])
        return flatten(df1)

    elif data['type'] == 'data_source':
        source_ids.append(data['uid'])
        datamart = data_access.get_by_uid(data['uid'])
        return spark_helper.read_datamart(datamart)
```

Workflow - Operations

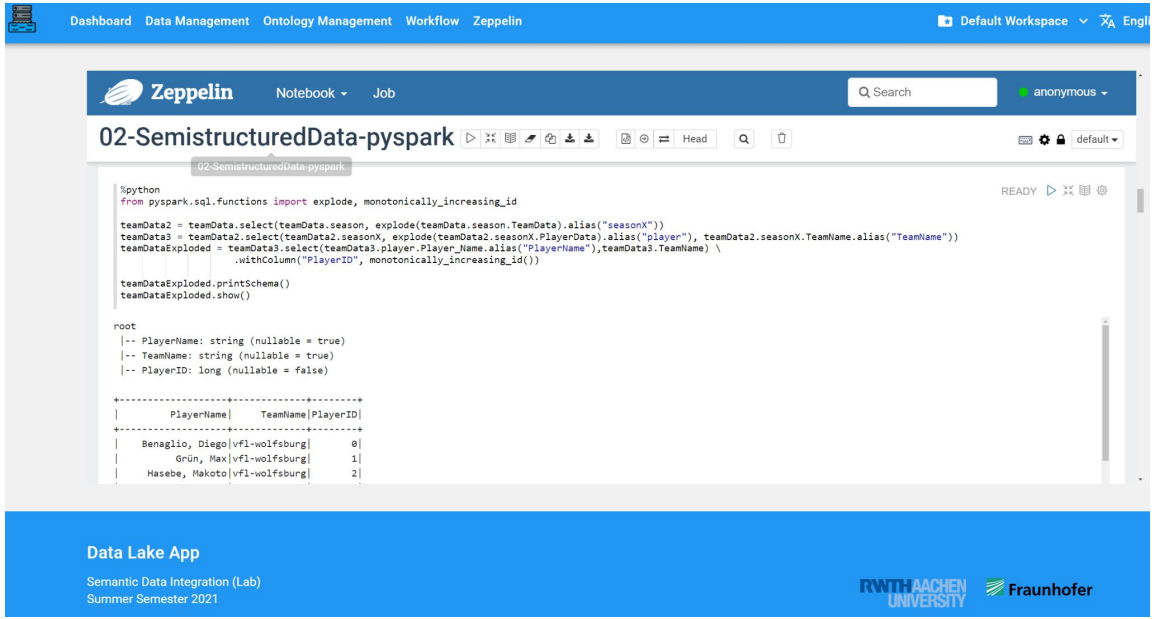
- Join
- GroupBy
- Select
- Filter
- Flatten (for JSON)

Workflow - Outlook

- More Transformation Nodes
- Possibly an Integration of Team 3 App Store
  - Apps directly integrated(e.g. Data Cleaning)
- Transformations regarding semi-structured data (JSON, XML) in particular
- Basic Machine Learning Nodes from Spark ML (Clustering, Decision Trees, etc.)
- Visualization

Apache Zeppelin

- Integrated Zeppelin UI into Frontend



- Zeppelin 0.9.0 does not support latest Spark 3.x - Workaround with PySpark

Documentation

- Backend: PyDocs
- APIs: Postman Collection
- Frontend: React Documentation comment
- ER-Diagram
- Wiki and ReadMe on GitLab

database.data_access.ontology_data_access

Modules

os settings

Functions

add(name, file, workspace_id) -> database.models.ontology.Ontology
Adds new ontology to Fuseki and adds an entry in MongoDB.
:param name: name of the ontology.
:param file: file that contains the ontology.
:param workspace_id: the workspace the file is to be added in Fuseki.
:returns: the newly created Ontology entity in MongoDB.

create_query_string(graph_name: str, keyword: str)
This methods generates the query string for the keyword-search in put.
:param graph_name: graph to be queried, default is "default graph",
like "<http://localhost:3030/60d5c79a7d2c38ee678e87a8/60d5c79d7d2c38ee678e87a9>"
:param keyword: keywords to search for or when search-bool is false the query itself
:returns: the query

delete(workspace_id, graph_id)
Delete ontology in Fuseki and the entry in MongoDB.
:param workspace_id: id of workspace in MongoDB and Fuseki.
:param graph_id: name of named graph.
:return:

get_all(workspace_id) -> [<class 'database.models.ontology.Ontology'>]
Get all ontologies in a specific workspace.
:param workspace_id: id of workspace.
:returns: all ontologies in the workspace.

get_suggestions(workspace_id, search_term)
This function provides multiple suggestions for a auto-completion of ontology-attributes in fuseki. The search_term
can either be the rdf:label or the name of the class itself(after the #).
:returns: a list of maximum 20 suggestions which fit the requirements ordered by the length of the label.

Poster

Semantic Data Integration - Summer

Semantic Data Integration 2021
Data Lake Team 2

RWTH AACHEN UNIVERSITY

Fraunhofer FIT

Supervised by Prof. Dr. rer. nat. Christoph Quix, Dr. Christoph Lange

RWTH Aachen, Fraunhofer FIT

TEAM-MEMBERS

- Sayed Husani
- Maher Fallouh
- Abdullah Zaid
- Tobias claas
- Muhammad Noman

ABSTRACTS

We present what we have done in Lab Semantic Data Lake and what specific and masterful works we have added in Data Lake. Where we made a four-layered architecture, Which includes an ingestion, a storage, a transformation and an interaction layer.For this functionality we have created different structures,In which different form of data structures are extracted from different sources, It is transferred to Datamart for integration and to be stored in different repositories. Where Data interaction, data workflow is accessable in frontend UI.The final result of the project is a user interface supports with different source systems and target systems. which we consider to be proposed as efficient way of using data in different state of the art real time applications.

INTRODUCTION

- In Introduction we include what we have done. What unique implementations we have done as a team. Therefore Semantic Data lake is new and better way of accessing, storing, extracting and handling the big data ,in this project Data lake we have worked in many things but in particular meta data is in particular important to mention, where we have made different Datamarts. Our main motivation to create Data mart is Easy access to frequently needed data of different type. we can easily Creates a collective view by a group of distinct select of data. Which eventually Improves response time for user at the end, With Ease of creation.In Functionality we popoulate data mart with data from source systems, and then accordingly retrieve and fetch data.

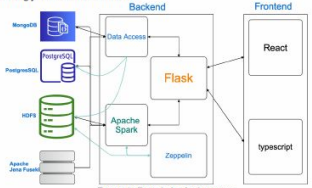


Figure 1: Data Lake Architecture.

METHODOLOGIES

- We include all technologies we have used in creation and functionalities of Data lake.
- Extracts data and metadata from heterogeneous sources.
- Transform and stores the metadata in an expandable Datamart.
- Provides extra explanation of semantic data from Datamart .
- New type of data source is easily integrated, stored and transformed.
- Store data in it's original form in mongoDB, Postgres and HDFS.
- For Deployment, Implementation and scaling Docker is used.

Table 1: Table explaining Methods and technologies.

Interface	Technologies	Purposes
Backendend	Python, Java	Store, Transform
Frontend	React, Typescript, Javascript	Extract, Ingest
Data Handling	Apache, Docker, Postman, Zeppelin	Data Processing

METADATA OVERVIEW

1. **Data Lake is created in structured way with the aim that Metadata is efficient way of matching customer's precise demand.**
2. **This make a Data Lake Modern solution for big data, rather then becoming a Data Lake swamp**

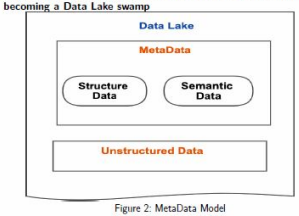


Figure 2: MetaData Model

FUNCTIONALITIES

- Integration of data sources is performed in **Dataset-Management**, where different types of data files with parameters are uploaded. Data file can be modified or deleted according to requirement.

Figure 3: Meta Data(Workflow)

- Then data is stored in **DataMart**, which can be easily accessed and modified by **Metadata**. This Function is easily been approached by workflow Diagram.
- Interactive Data analytics is performed by **Zeppelin** notebook, which helps in data-driven documents and codes.

Figure 4: Zeppelin

CONCLUSION

1. We have put an effort to build Masterful Data Lake.That ingest Different kind of data from heterogeneous sources.
2. Furthermore Data Lake transform and store the Metadata in Datamart. We propose it is complete solution for big data handling.

DEMO

Thank you for Your Attention