# KnowledgeGraphsJava

Java project where the core functionalities of the KGs generation and PII identification pipeline are implemented. Also encompasses modules that establish communication with the rest of the KGs component’s modules (Abbrev Expansion and Bertmap), as well as other components of Work Package 4, mainly the Preprocessing Tool (T4.1 – CERTH) and the Kafka Message Broker (TRUSTUP). In short, it includes:

* The input and output points of the KGs’ component
* The management of the input datasets (download, storage, configuration files)
* The main steps of the KGs generation pipeline:
  + Preparation of input datasets (such as parsers and basic clean up)
  + Putative Ontology Extraction, referred to as “Input Standardization Module” in D4.1
  + Parts of the “Mapping Generation Module” (D4.1)
  + The “Ontology Refinement and KG Generation” (D4.1)
* Parts of the PII identification pipeline

Functionalities per package:

## A\_Coordinator

* **Inputs**: Manages the **input** point of the KGs by establishing communication with the preprocessing tool to receive data files uploaded by the data owners along with associated metadata through a sequence of API requests. This subpackage implements API endpoints to accomplish two main tasks: (1) receive a **POST** request, notifying the KGs that the preprocessing of an uploaded file is finished, and (2) download the file by executing a **GET** request towards the preprocessing tool. These API endpoints are implemented using the Spring Boot framework.
* **Kafka**: The **output** of the PII identification process performed by the KGs follows a JSON format. This subpackage contains the Kafka **producer** responsible for publishing this JSON file to a topic ('piisTopic') in the Kafka Message Broker. *(At the side of the API Node -see Trustup repo- this json is consumed and then stored as a document in the ‘piis\_collection’ in the MongoDB – a shared databased for all the components)*

*Read more about the interaction between the KGs and other WP4 components at Deliverable 5.2.*

* **Config**: This class manages configuration files for the KnowledgeGraphsJava project. This includes parameters about:
  + The input dataset: The data owner uploads files one-by-one to the UI, but the KGs need to accumulate uploaded files belonging to the same dataset to create a complete KG (e.g., it wouldn’t make sense to create a KG for a single table in a relational database). At the moment, files are grouped to datasets according to the use case (fintech, health) and their datatype (csv, excel, dcm).
  + The outputs of the KGs e.g., do you want to upload the generated KG to a GraphDB repo?
  + Selected domain ontology for semantic annotation of the dataset and “hyperparameters” for the Mapping Selection.

*Predefined config files for each dataset of the health and fintech use cases can be found as “./KnowledgeGraphsData/KnowledgeGraphJava/ConfigFiles/${UseCase}\_${filetype}\_Config.json”*

* **Pipeline**: This class coordinates the KG generation and PII identification pipelines by executing their steps by calling a sequence of other classes and methods.

## B\_InputDatasetProcessing

When a file is downloaded by making a GET call to the preprocessing tool, it is saved at the “Downloaded\_Data” subdirectory of its dataset.

This package prepares the downloaded files for subsequent steps of the pipeline. Different approaches are followed according to the file’s type.

* **Tabular:**This subpackage manages files structured in a tabular format, such as CSVs, Excel files, and SQL databases. The schema for these datasets is represented by the **RelationalDB** class, comprising objects of the **RTable** class, with each object representing an individual table. In the case that the input dataset is:
  + - An **SQL database**, the .Connectors.**SQLConnector** class is called to connect to the DB (JDBC) and retrieve info about the schema (i.e., table names, column names and datatypes, primary/foreign key connections). Each relational table in the DB will become an object of the RTable class.
    - A set of **CSV-like files** stored at the Downloaded\_Data, the **TabularFilesReader** is used to prepare and clean up the files (like, handle duplicate headers, determine column types etc.). The results are saved at Processed\_Data directory. The set of files is represented by an object of the **RelationalDB** where each file is represented as an object of the **RTable** class.

🡪 *We consider each CSV file as a table that is part of a relational database e.g., RelationalDB { tables = account (from account.csv) , person (from person.csv) }.*

* **DICOM:** KGs are generated using the metadata contained within DICOM files, which adhere to a format similar to that of a JSON structure. The complexity of these metadata ranges from simple key-value pairs (found in low-complexity DICOM file types like MRI and CT) to nested structures (as seen in highly complex DICOM file types such as an XRAY dosage file). To simplify the handling of this structure, the metadata from a DICOM file is extracted into a JSON file (referred to as DSON from hereon). The structure of this JSON file aligns with the modeling patterns of the Semantic DICOM ontology (**DICOM2SediJSON**.java), which is the chosen domain ontology for modeling the DICOM dataset's KG. (On the other hand, an earlier version of this transformation process, independent of the selected domain ontology, can be found at .temp.replaced.DICOM2JSON).
* **Medical.AbbreviationsDictionary.java:** This class invokes the API endpoint of the AAExpansion service (<http://knowledge-graphs-python:7531/start_aaexpansion>). It sends as input an ordered list of headers from a tabular file in the health use case, such as the patient record Excel file. The response is the result of the **AAExpansion** process, where each header containing abbreviations is expanded to its full form. These expanded forms are integrated into the Putative Ontology as supplementary annotations of the ontology's resources, using the skos:altLabel property.