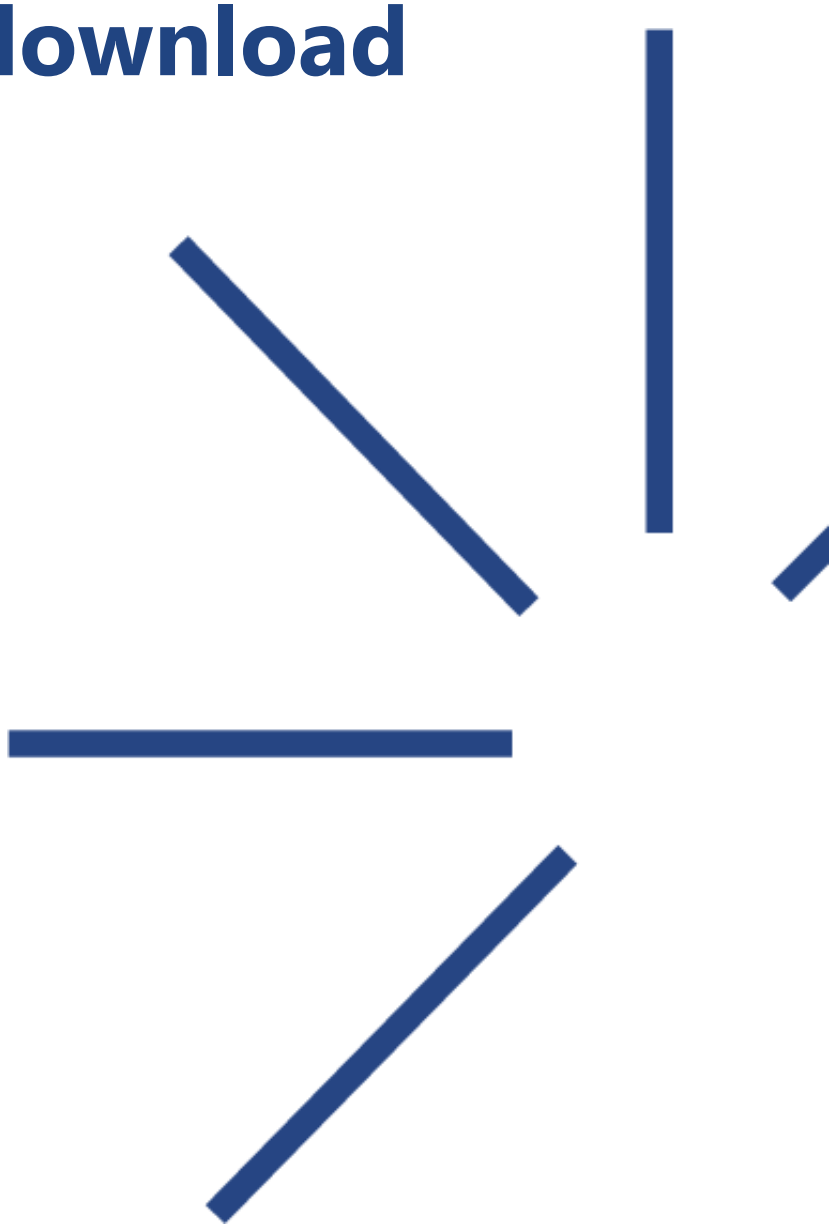
D3 - Methods for continuous download



Project funded by the European Union – Next GenerationEU through the Recovery and Resilience Plan of the Slovak Republic under project No. 09I05-03-V02-00049.



INTRODUCTION

The project **"Automation of Legal Text Analysis Based on Machine Learning"** (hereinafter referred to as **"ALTAML"**) represents a key initiative aimed at integrating innovative approaches in the field of data processing and subsequent analysis, specifically within the legal domain, which also includes the law of information and communication technologies. The objective of this project is to develop and validate effective methods for the automated analysis of legal texts using machine learning techniques. This includes, in particular, the development of tools that facilitate the processing and analysis of large volumes of data in the form of various legal documents, as well as the extraction of relevant information (attributes) from such documents, including the identification of key terms, references to other legal acts, and other attributes.

The ALTAML project thus aims to contribute to a more efficient access to legal information and to the acceleration of legal processes, thereby ensuring a higher degree of legal certainty and improving the accessibility of legal texts, the results of their analysis, and relevant legal information for both professionals and the general public.

Within Work Package KPB1, **Deliverable D3 – Methods for continuous download** – was developed. This document presents notes on a highly resilient system for continuous downloading of court decisions, implemented in Python using a microservices architecture and communication via RabbitMQ. The decision texts and metadata are converted into JSON format and stored in a MongoDB database. For fast search capabilities, selected attributes are also stored in a relational MySQL database. The system is designed to be resistant to failures of both external and internal components and can be deployed using Docker and Docker Compose technologies.

1. ALTAML Decisions downloader

Decisions downloader is a Python-based project designed to download the latest court decisions, transform them into our custom JSON format, and save them into MongoDB and MySQL databases.

The following sections describe the main functionalities, system requirements, configuration, and deployment procedures.

Features

The key functionality of the project can be summarized in several core processes:

- **Download Data**: Automatically fetch the latest court decisions from the Ministry of Justice of Slovak repulic.
- **Transform Data**: Process and transform the downloaded metadata and PDF files into a custom JSON format suitable for later usage.
- **Store Data**: Save the transformed data into both MongoDB and MySQL databases for querying and analysis.
- **Parallel processing** - The whole process is organized as microservices that communicate using RabbitMQ massage broker. The microservices are easy to replicate to many servers to provide efficient load balancing.

Requirements

Before deployment, several system components must be available to ensure a stable and isolated environment. The project is fully containerized using Docker and Docker Compose, relying on MongoDB and MySQL for persistent data storage.

Required components:

- Docker and Docker compose
- MongoDB (for storing transformed data)
- MySQL (for storing transformed data)

Setup Mongo and MySQL on server

Database initialization and configuration are performed using Docker Compose. Follow these steps to start both databases on the server:

- run docker-compose-databases.yaml
    - it is using environmental variables for users
    - when you have default env file .env run docker compose up -f docker-compose-databases.yaml -d

- when you create env file that is not default you need to run docker-compose --env-file variables.env -f docker-compose-databases.yaml up -d

2. DataDownloader

The DataDownloader component manages the extraction and storage of raw decision metadata in MySQL. It forms the backbone of the data ingestion layer, ensuring consistency and traceability of downloaded decisions.

In MySQL create table raw_decision

```
create table raw_decision (
  guid varchar(75),
  datum_vydania date,
  spisova_znacka varchar(20),
  our_id varchar(150),
  ecli varchar(45),
  file_name_original varchar(150),
  file_name_txt varchar(150),
  downloaded_json datetime,
  saved_json_with_fulltext datetime,
  primary key (datum_vydania, guid)
);
```

RabbitMQ

RabbitMQ serves as the core communication layer between microservices, ensuring reliable message delivery and coordination. Follow the steps below to install and configure RabbitMQ:

- to install RabbitMQ run docker-compose-rabbitmq.yaml
  - for it to run you need to have .ez file for delayed messages plugin which is used
  - you can download specific version here: https://github.com/rabbitmq/rabbitmq-delayed-message-exchange/releases
  - make sure to have .ez file in same folder as dockerfilerabbitmq
    - dockerfilerabbitmq will automatically enable all needed plugins it needs
  - in dockerfilerabbitmq insert/change name of .ez file
- to create all queues, bindings, exchange and stream run docker-compose-config.yaml file

Setup of environment variables

Environment variables define system parameters, connection settings, and per-server configurations. Follow these steps to configure the environment:

- create variables.env file under root directory(e.g. data-collector/variables.env)
    - copy content from variables_example.env to variables.env and replace with your own values
    - if you don't use default env file .env you need to change env file for every docker-compose file
- environmental variables will differ on different servers
- every server will have different value for variable SERVER_NUMBER which specify server
- connection to rabbitmq, mongo and mysql will differ on every server, because if only one server have them running other have to connect to them.
- path to files will remain the same for instance that there is a central storage for them
    - even when there is no central storage, it is not needed to change and can still remain the same
- you can setup file src\no_consumer_services\last_date.txt to write down the first date from which the data will be downloaded from the Ministry of Justice

3. Run on server with docker

Running the system on a server requires no manual installation of dependencies. Everything is handled within Docker containers.

- you don't need to install anything, everything will be installed inside docker container
- you only need to insert every environmental variable that was mention before this section, logins and connection to databases
- if you don't use default env file .env you need to change env file for every docker-compose file

How to run docker-compose files

Docker Compose simplifies the execution and orchestration of services.

- docker compose up -d to run default docker compose file
- docker compose -f docker-compose file up -d to run specific

Consumer services

Consumer services form the backbone of the automated data processing pipeline. They handle downloading, transformation, storage, and monitoring tasks.

- to run every service for whole data flow run docker-compose.yaml file
    - to run only one from the services docker compose up -d - build name use name from docker-compose file
    - if service use volume, specify path from environmental variable
- to run service used for monitoring failures of serviced for data flow run docker-compose-email.yaml file
    - code for sending email is located in email_sender_service.py
    - to configure which mail address should receive emails change receiver email variable
- to run service which will consume from stream and insert decision to MongoDB collection run docker-compose-stream.yaml
- to run service which will get a list of new or updated decision run docker-compose-list.yaml file
    - if on linux, write 0 2 * * Sat .../Start-decision-list.sh inside crontab -e file, write path to Start-decision-list.sh

Administrator

Administrative services support maintenance, recovery, and data correction operations.

- to create an export of failed decision run docker-compose-export.yaml file, to specify from which service export will be created change value of environmental variable inside docker-compose file
- to reset failed decision from MySQL back to RabbitMQ queues run docker-compose-reset.yaml, to specify for which service reset will be done change value of environmental variable inside docker-compose file
- second stream consumer is used by administrator to insert back and old version of decision from stream, to use it run docker-compose-stream-admin-yaml file
    - inside docker-compose file are two environmental variables to specify filtering value and offset value

4. Source files

The project's source code is maintained in the internal UPJŠ GitLab repository. It can be accessed at the following address: https://gitlab.science.upjs.sk/altaml/decisions-downloader