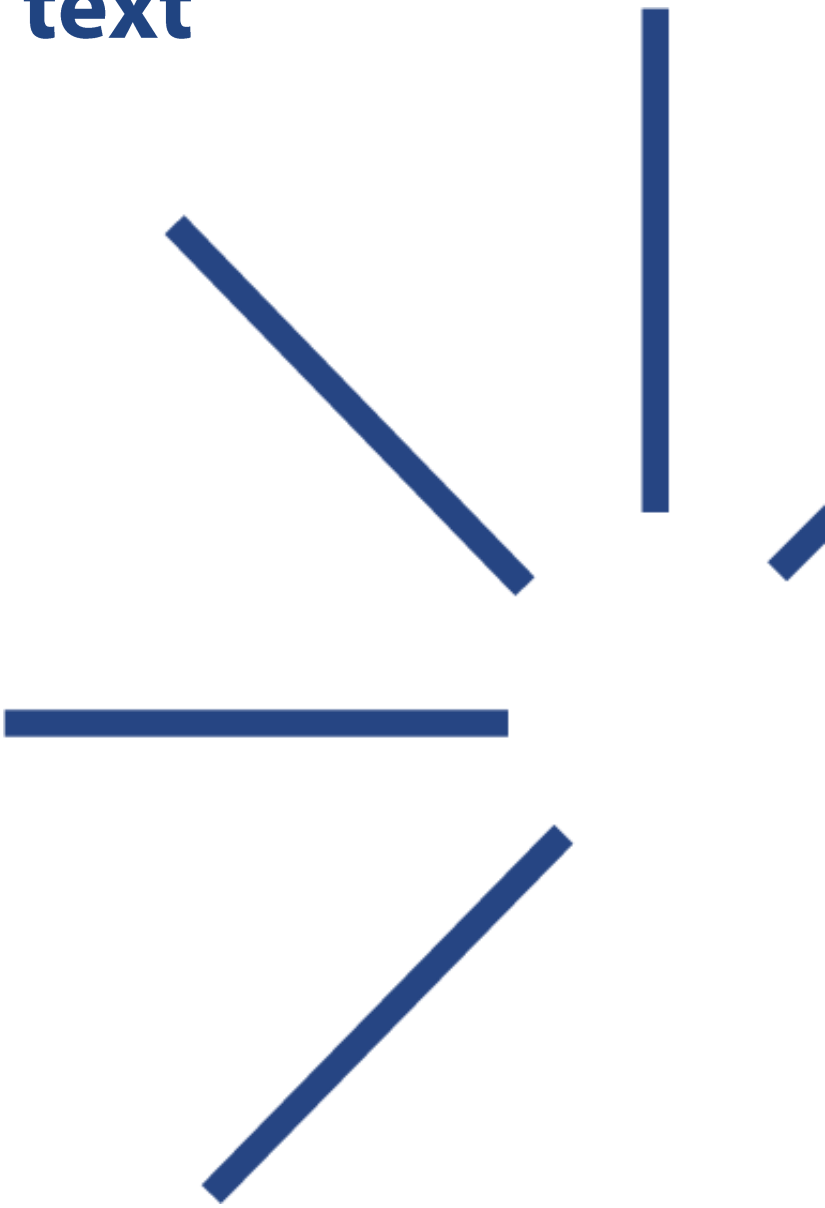
D16 – Query model for filtering legal text



Project funded by the European Union's Next Generation EU through the Slovak Republic's Recovery and Resilience Plan under project no. 09I05-03-V02-00049.

Financované Európskou úniou NextGenerationEU
PLÁN OBNOVY
ÚRAD PODPREDSEDU VLÁDY SLOVENSKEJ REPUBLIKY PRE PLÁN OBNOVY A ZNALOSTNÚ EKONOMIKU
VAIA VÝSKUMNÁ A INOVAČNÁ AUTORITA
VÝSKUMNÁ AGENTÚRA

INTRODUCTION

The project **"Automation of Legal Text Analysis based on Machine Learning"** (hereinafter "**ALTAML**") represents a key initiative aimed at integrating innovative approaches in the field of data processing and subsequent analysis, with a focus on the field of law, which also includes information and communication technology law. The aim of this project is to develop and validate effective methods for the automated analysis of legal texts using machine learning techniques. This involves, 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 legislation and other attributes.

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

As part of work package KPB3**, D16 – query model for filtering legal text** using a combination of extracted attributes, full-text and semantic search – was developed. The aim of this output is to define a method for searching legal texts, particularly court decisions and legislation, whilst the proposed model can also be extended to other types of legal documents. The model allows for the combination of:

- full-text search within the texts of court decisions and legislation,
- searching by extracted references to legislation,
- searching by extracted references to other court decisions,
- searching by automatically identified key terms,
- filtering by categorised attributes (e.g. area of law, sub-area of law, court or date of decision),
- semantic search using vector representations of court decisions and legislation,
- hybrid search combining full-text, attribute and semantic search.

The proposed query model is based on the use of Elasticsearch Query DSL and builds on deliverable **D15 – Repository Design**, in which Elasticsearch was defined as the final search layer of the ALTAML system.

Higher layers of user interaction and intelligent assistance in query formulation can subsequently be built on top of this query model. This functionality is addressed separately within the follow-up project output **D17 – Intermediary search method**, whose task is to transform user requests formulated in natural language into queries over legal documents.

1. Elasticsearch Index Configuration

In the Elasticsearch environment, an index represents a logical repository of documents intended for searching and filtering data, which can be likened to a table in relational SQL databases. Each index contains documents of the same type, along with a definition of their structure (*mapping*) and the method of processing text fields (*settings* and analysers). The index configuration determines the data types of individual fields, the method of text processing, and the supported search types.

The proposed query model utilises separate Elasticsearch indices for individual types of legal documents. Their configuration is presented in the following sub-sections.

1.1. Configuration of the court decisions index

The court decisions index contains individual court decisions represented as separate documents. The documents contain full-text fields, structured attributes and vector representations used in various types of search. The index configuration consists of *settings* and *mappings* sections.

The *settings* section defines how text is processed during indexing and searching. An important part of this configuration are the analysers, which determine how text is tokenised and how text transformations are applied, such as conversion to lowercase or lemmatisation.

The following exampleExample 1 presents the configuration of the *settings* section used for the court decisions index:

```
"settings": {
  "analysis": {
    "tokenizer": {
      "standard": { "type": "standard" }
    },
    "filter": {
      "lowercase": { "type": "lowercase" },
      "hunspell_sk": {
        "type": "hunspell",
        "locale": "sk",
        "dictionary": "/usr/share/elasticsearch/config/hunspell/sk"
      }
    },
    "analyzer": {
      "slovak_lemmatizer": {
        "type": "custom",
        "tokenizer": "standard",
        "filter": ["lowercase", "hunspell_sk"]
      }
    }
  },
  "number_of_shards": 1,
  "number_of_replicas": 1
```

```
}
```

Example 1– Configuration of the settings section of the court decisions index

For full-text fields, a custom analyser called `*slovak_lemmatizer*`, based on the Slovak **Hunspell** dictionary, is used. Using the Hunspell analyser allows for matching even between the same words in different grammatical forms, which is important when searching Slovak texts.

The *mappings* section (Example 2) defines individual document fields, their data types and how they are used in searches. Data types determine whether a field will be used for full-text search, filtering, date queries or semantic search using vector representations.

```
"mappings": {
 "properties": {
  "document_pretty": {
   "type": "text",
   "analyzer": "slovak_lemmatizer"
  },
  "date_of_decision": {
   "type": "date",
   "format": "yyyy-MM-dd"
  },
  "area_of_law": {
   "type": "keyword"
  },
  "text_chunks": {
   "type": "nested",
   "properties": {
    "part": {
     "type": "keyword"
    },
    "text": {
     "type": "text",
     "analyzer": "slovak_lemmatizer"
    },
    "embedding": {
     "type": "dense_vector",
     "dims": 384,
     "index": true,
     "similarity": "cosine"
    }
   },
  //other attributes...
  }
 }
}
```

Example 2 – Configuring the mappings section of the court decisions index

In the index configuration, data types are assigned to individual fields according to how they are used in searches (Table1). The **text** type is intended for full-text fields analysed during searches, the **keyword** type for exact filtering and aggregations, and the **date** type for date fields supporting range queries. The **nested** type is used for nested structured objects, such as references to legislation or other court decisions. For semantic search, fields of **the dense_vector** type are used, which contain vector representations of legal documents.

Table1 - Attributes of the court decision index, their data types and purpose in searching and filtering

Attribute	**Type**	**Purpose**
document_pretty	text	Full-text search of the decision text
date_of_decision	date	filtering by date
ECLI	keyword	exact filtering by ECLI identifier of the decision
case_number	keyword	exact filtering by case number
type_of_decision	keyword	filtering by form of decision
area_of_law	keyword	filtering by area of law
sub-area_of_legislation	keyword	filter by legal sub-area
laws	nested	filtering by identifiers of referenced legislation
dockets	nested	filtering by referenced decisions
key_phrases	keyword	filtering by extracted keywords
case_name	keyword	filter by court name
court_type	keyword	filter by court type
judge_name	text	full-text search by judge's name
text_chunks	nested	textual parts of the decision used in semantic search
text_chunks.part	keyword	filtering by decision section
text_chunks.text	text	full-text search within specific parts of the decision
text_chunks.embedding	dense_vector	semantic search based on semantic similarity to parts of decisions

As court decisions can be lengthy, the vector representation is not created for the entire decision, but for smaller text segments (chunks). Each chunk contains the text itself, information about the part of the decision from which it originates (e.g. header, introduction, operative part, reasoning or instructions), and its vector representation, which enables semantic search for similar parts of decisions.

In a semantic query, the user's expression or question is first converted into a vector representation using the same language model and then compared with the vectors of individual chunks stored in Elasticsearch. In this way, it is possible to search for semantically similar decisions or parts thereof even in cases where the user's query does not contain the same words as the document itself. At the same time, it is possible to restrict the search to specific parts of the decisions, for example, only to the reasoning or the operative part of the decision. An example of such a semantic query is presented in the sub-section '2.1 '.

1.2. Configuration of the legislation index

As legal regulations have a different structure to court decisions, they are stored in a separate Elasticsearch index.

The configuration of the *settings* section, including the *slovak_lemmatizer* analyser, is handled in a similar way for the legislation index as for the court decisions index. For the sake of clarity, the example of the *mappings* configuration is not repeated here, and the individual index attributes are summarised directly in the following tableTable2 .

Table2 - Attributes of the legislation index, their data types and purpose in searching and filtering

Attribute	Type	Purpose
law_id	keyword	precise filtering by legal act identifier
law_titles	text	full-text search by hierarchical headings of the legal act
valid_from	date	filtering by effective date
valid_to	date	filter by end date
decision_type	keyword	filter by form of decision
text	text	full-text search in the text of the legislation
key_phrases	keyword	filtering by extracted keywords
embedding	dense_vector	semantic search based on semantic similarity to legal provisions

Unlike court decisions, legal regulations do not involve dividing the text into chunks. Semantic search is performed using a single vector representation created for the entire legal regulation and stored in the embedding array.

2. Types of supported queries

The proposed query model allows for various types of searches to be performed on court decisions and legislation, and for these to be combined within a single Elasticsearch DSL query. As individual document types are stored in separate Elasticsearch indices, queries are executed on the relevant index, for example by calling GET /court_decisions/_search or GET /laws/_search

The following sub-sections present examples of selected query types.

2.1. Examples of queries on court decisions

The following examples present selected query types demonstrating the capabilities of the proposed query model for court decisions.

Query listed at the Example3 searches for court decisions that either contain the term “unfair competition” or contain an extracted reference to Section 49 of Act No. 513/1991 Coll. Commercial Code, which defines bribery.

The query uses the *'should'* condition, which represents a logical OR operation between full-text search and filtering by extracted legal provisions. *The 'minimum_should_match'* parameter specifies the minimum number of conditions in the *'should'* section that must be met. A value of 1 means that the document must meet at least one of the specified conditions.

```
{
  "query": {
    "bool": {
      "should": [
        {
          "match": {
            "document_pretty": "unfair competition"
          }
        },
        {
          "nested": {
            "path": "laws",
            "query": {
              "bool": {
                "must": [
                  {
                    "term": {
                      "laws.law": "513/1991"
                    }
                  },
                  {
                    "term": {
                      "laws.paragraph": "49"
                    }
                  }
                ]
              }
            }
          }
        }
      ],
      "minimum_should_match": 1
    }
  }
}
```

Example3 - Elasticsearch query searching for decisions containing the term "unfair competition" or Section 49 of the Commercial Code

TheExample4 presents a query searching for court decisions to which at least one of the key terms "misleading advertising" or "trade secret" has been automatically assigned. The *terms* operator allows you to search for documents containing at least one of the specified values. Unlike full-text search, in this case the search is not performed directly within the text of the decision, but within a structured attribute (key_phrases) containing extracted terms.

```
{
  "query": {
```

```
                        "terms": {
                          "key_phrases": [
                            "misleading advertising",
                            "trade secret"
                          ]
                        }
                      }
                    }
```

Example4 - Elasticsearch query searching for decisions based on the keywords "misleading advertising" and "trade secret"

TheExample 5 presents a semantic query searching for court decisions containing passages **semantically similar** to the term "unfair competition", with the search limited only to the part of the decision constituting the reasoning.

```
    {
      "size": 10,
      "query": {
        "nested": {
          "path": "text_chunks",
          "query": {
            "bool": {
              "filter": [
                {
                  "term": {
                    "text_chunks.part": "reasoning"
                  }
                }
              ],
              "must": [
                {
                  "knn": {
                    "field": "text_chunks.embedding",
                    "query_vector":[0.12,...,-0.03,0.44], /*384-dimensional vector
*/
                    "k": 30,
                    "num_candidates": 300
                  }
                }
              ]
            }
          },
          "inner_hits": {
            "size": 3
          }
        }
      }
    }
```

Example 5 – Elasticsearch KNN query for semantic search in the reasoning section of court decisions

In semantic search, a user query – for example, the term 'unfair competition' – is first converted into a vector representation (*query_vector*) using the same language model that was used to create the text_chunks.embedding array. TheExample 6 demonstrates the creation of a vector representation of a user query in Python using the sentence-transformers library.

```
from sentence_transformers import SentenceTransformer

model = SentenceTransformer("sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2")
query_vector = model.encode("unfair competition").tolist()
```

Example 6 – Creating an embedding vector for a user query using the sentence-transformers library

The dimensionality of the vector depends on the language model used. In the example above, the *paraphrase-multilingual-MiniLM-L12-v2* model generates 384-dimensional vector representations. The language model used can be replaced with another model in the future; however, in such a case, it is necessary to recreate the vector representations of all indexed documents so that the embeddings of documents and user queries are created in the same vector space.

The generated vector representation is then compared using cosine similarity with the vectors of individual text segments of the decisions stored in Elasticsearch. This approach allows for the retrieval of semantically similar passages even in cases where the document does not contain the exact same words as the user query.

Using the text_chunks.part attribute also allows semantic search to be restricted to a specific part of the decision, such as the reasoning, the operative part or the instructions.

The size parameter specifies the maximum number of court decisions returned to the user.

Elasticsearch performs *KNN* queries using Approximate Nearest Neighbour (ANN) search. The *num_candidates* parameter specifies the number of such candidate text chunks shortlisted, from which *the* most similar results are subsequently selected. A higher value *for num_candidates* can improve the accuracy of the results, but also increases the computational complexity of the search.

The *inner_hits* attribute determines the number of text segments from the decision with the highest score that will be returned to the user along with the final decision. In the example given, a maximum of 10 decisions will be returned, and for each decision, a maximum of 3 of the most relevant text chunks.

In addition to separate full-text and semantic search, it is also possible to combine both approaches within a single **hybrid query**. This approach allows not only exact textual matches with the user's query to be taken into account, but also semantic similarity between documents and the user's query.

TheExample 7 demonstrates a hybrid search combining the classic BM25 full-text score with semantic similarity based on vector representations. The resulting document score is derived from a weighted combination of both approaches.

```
{
  "query": {
    "script_score": {
      "query": {
        "match": {
          "document_pretty": "unfair competition"
        }
      },
      "script": {
        "source": """
          0.7 * _score +
```

```
                0.3 * cosineSimilarity(
                  params.query_vector,
                  'embedding'
                )
              """,
              "params": {
                "query_vector": [0.12, -0.03, 0.44]
              }
            }
          }
        }
      }
```

Example 7 - Hybrid search combining BM25 full-text scores and cosine similarity of embeddings

In the example above, the value *_score* represents the BM25 full-text search score, and the *cosineSimilarity* function expresses the semantic similarity between the user query and the document's vector representation. The weights 0.7 and 0.3 determine the relative influence of full-text and semantic search on the resulting document score. A higher weight assigned to the full-text score favours documents containing an exact textual match with the user query, whilst a higher weight for semantic similarity favours documents that are semantically similar to the user query even without the occurrence of the same words.

2.2. Examples of queries on legislation

The proposed query model also supports searching within legislation stored in a separate Elasticsearch index. When searching, it is possible to combine full-text search within the text of legal regulations, filtering by period of validity, searching by extracted terms, and semantic search using vector representations of legal regulations.

The Example 8 presents a query searching for legal regulations containing the term "unfair competition" that were in force as of 1 January 2026.

```
{
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "text": "unfair competition"
          }
        }
      ],
      "filter": [
        {
          "range": {
            "valid_from": {
              "lte": "2026-01-01"
            }
          }
        },
        {
          "range": {
            "valid_to": {
              "gte": "2026-01-01"
```

```
                        }
                    }
                }
            ]
        }
    }
}
```

Example 8 - Elasticsearch query combining full-text search and filtering by the validity period of a legal regulation

Semantic search using vector representations and search by extracted keywords is implemented for legal regulations in a similar way to that used for court decisions. The difference lies mainly in the different structure of the indexed documents and in the use of a single vector representation for the entire legal regulation.