Introduction to Semantic Systems - Final Report

Group 01

Cem Bicer (01425692)
Helmuth Breitenfellner (08725866)
László Király (09227679)
Gerald Weber (00125536)

14 February 2020

Task 1: Project Idea

Our project idea is to create a mobile app (platform independent, using HTML technology) for students, developers and data scientists.

Based on data from developer surveys, repositories and university lectures, it will answer questions about location, income, lectures and programming languages.

Specifically we want it to be able to answer the following questions:

- [**Study**] As a student I want to learn language *(programming language)*. Which courses should I take?
- [**Location**] As a *(profession)* I live in *(country)* and I can program in *(programming language)* and I want at least *(amount)*. Should I stay our should I go?
- [**Money**] As a student, I live in *(country)*. I want to learn a language which offers the most salary prospect in my country. Which languages shall I consider?
- [**Jobs**] I want to practice language *(programming language)*. Which repositories should I look at, as they are most popular and written in this language?

The app displays the four core questions as four tiles. When selecting a tile, the user is asked for details, like their knowledge, location, or their desired income level. Then the app will submit these details and provide the answer to the question.

Figure 1 shows a mockup of the start screen of the mobile app.

Task 2: Data Collection

We have the following four data sources:

- **Kaggle User Survey** (László Király)
  Data Scientists, Country, Job Role, Programming Language, Income
- **StackOverflow User Survey** (Cem Bicer)
  Software Developer, Country, Job Role, Programming Language, Income
- **GitHub Repositories Data** (Helmuth Breitenfellner)
  Repository URL, Popularity, Programming Language, Issues
- **TISS Lectures** (Gerald Weber)
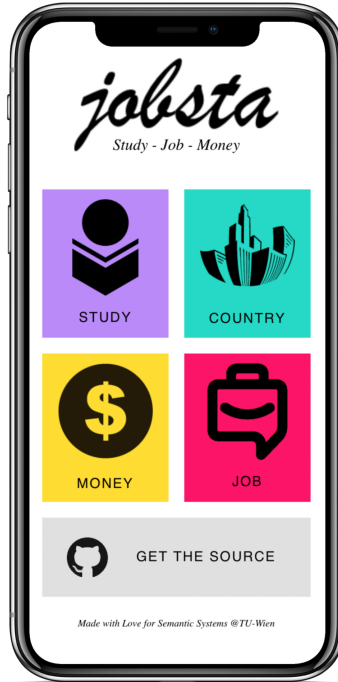 Lectures, Lecturer, Description, Programming Language

Figure 1: Mockup of Mobile Start Screen

Data 1: Kaggle ML & DS Survey

The Kaggle survey[1] is from 2019 and contains almost 20,000 individual and anomymized responses to a set of multiple choice and free text questions.

Since Kaggle is a competition platform for challenging machine learning problems, it comes to no surprise that the survey itself was presented within a competition with a prize pool of $30,000 for the most original data stories about the survey.

The multiple choice answers were reflected in 245 columns, where some of the answers had one column, e.g. age (range), gender, country and income, whereas others were one hot encoded, e.g. progamming languages.

We aligned this dataset with the StackOverflow dataset regarding gender, programming languages, and job role. Income was depicted as ordinal ranges in Kaggle but as continous value in StackOverflow.

The analysis of the metadata was done in `preprocess_kaggle.ipynb` *Part I - Analyse and Generate Metadata*.

Data 2: StackOverflow User Survey

The 2018 survey results are publicly available[2] where each row represents a single survey instance of a specific user. Each column describes a question of the survey and the rows contain the answer of the user. The original StackOverflow user survey 2018 results consists of a CSV file with over 90,000 entries and over 40 columns. Obviously, there were lots of data we did not need for our application use case, therefore we removed unused columns such as *"Do you program as a hobby?"*. Also, for the sake of simplicity and efficiency we stripped the 90,000+ entries down to around 700, which we decided to do so because it makes no difference for the ontology how many instances we use.

As we had another similar developer survey (Kaggle), we had to harmonize these data. For instance, we had the developer role "Software Engineer" in Kaggle and "Front-end developer" in StackOverflow. We decided to merge them to a single "Software Engineer" to have a simpler and better understandable ontology.

---

[1]Kaggle survey: https://www.kaggle.com/c/kaggle-survey-2019, last seen on 2020-02-13

[2]StackOverflow survey description: https://insights.stackoverflow.com/survey/2018, download page: https://insights.stackoverflow.com/survey, last seen on 2020-02-10

Data 3: GitHub Repositories Data

To obtain the data we were considering two options:

- gathering live data from GitHub, using e.g. the GraphQL API
- downloading collected data from e.g. `ghtorrent.org`.

Both options have their advantages and disadvantages.
For using the live data to build the knowledge graph we would have used *GraphQL* queries like this, as can be tried out at `https://developer.github.com/v4/explorer`:

```
query{
  search(type: REPOSITORY, query: "language:python", first:10) {
    userCount
    edges {
      node {
        ... on Repository {
          name
          url
          stargazers {
            totalCount
          }
          owner{
            login
          }
        }
      }
    }
  }
}
```

At the end we went for downloading collected data, as this required less manual work (compared to performing one query per language) and also makes more data available for other questions which might be asked.

The biggest issue was dealing with the large amount of data from `ghtorrent.org`. The download consists of a file with size 100GB compressed, which then had to be extracted and analysed.
The download contains the following files:

```
-rw-rw-r-- 1 helmuth idc                310 Jun  1  2019 ORDER
-rw-rw-r-- 1 helmuth idc               5326 Jun  1  2019 README.md
-rw-rw-r-- 1 helmuth idc         1033941154 Jun  1  2019 commit_comments.csv
-rw-rw-r-- 1 helmuth idc        27874983212 Jun  1  2019 commit_parents.csv
-rw-rw-r-- 1 helmuth idc       137449918096 Jun  1  2019 commits.csv
-rw-rw-r-- 1 helmuth idc         1118734835 Jun  1  2019 followers.csv
-rwxrwxr-x 1 helmuth idc               2228 Jun  1  2019 ght-restore-mysql
-rw-rw-r-- 1 helmuth idc                703 Jun  1  2019 indexes.sql
-rw-rw-r-- 1 helmuth idc         7464558601 Jun  1  2019 issue_comments.csv
-rw-rw-r-- 1 helmuth idc         9437001225 Jun  1  2019 issue_events.csv
-rw-rw-r-- 1 helmuth idc          489917235 Jun  1  2019 issue_labels.csv
-rw-rw-r-- 1 helmuth idc         5862007798 Jun  1  2019 issues.csv
-rw-rw-r-- 1 helmuth idc           25594106 Jun  1  2019 organization_members.csv
-rw-rw-r-- 1 helmuth idc       116067628357 Jun  1  2019 project_commits.csv
-rw-rw-r-- 1 helmuth idc         6189106041 Jun  1  2019 project_languages.csv
-rw-rw-r-- 1 helmuth idc          663446623 Jun  1  2019 project_members.csv
-rw-rw-r-- 1 helmuth idc           23548935 Jun  1  2019 project_topics.csv
-rw-rw-r-- 1 helmuth idc        23464280056 Jun  1  2019 projects.csv
-rw-rw-r-- 1 helmuth idc         6029885297 Jun  1  2019 pull_request_comments.csv
```

```
-rw-rw-r-- 1 helmuth idc     5059804548 Jun  1  2019 pull_request_commits.csv
-rw-rw-r-- 1 helmuth idc     7720141155 Jun  1  2019 pull_request_history.csv
-rw-rw-r-- 1 helmuth idc     2715930046 Jun  1  2019 pull_requests.csv
-rw-rw-r-- 1 helmuth idc    11886216368 Jun  1  2019 repo_labels.csv
-rw-rw-r-- 1 helmuth idc              0 Jun  1  2019 repo_milestones.csv
-rw-rw-r-- 1 helmuth idc          18833 Jun  1  2019 schema.sql
-rw-rw-r-- 1 helmuth idc     2767031027 Jun  1  2019 users.csv
-rw-rw-r-- 1 helmuth idc     5769651559 Jun  1  2019 watchers.csv
```

Relevant for our use case are the files `projects.csv` and `issues.csv`.

As a first step, the data was filtered and merged, using an *R* script. This script is called `transform.R`. Only original repositories (not forked ones) were taken into account, and only those which have been forked more than 50 times (as a measure of *popularity*) were looked at. Similarly the issues per repository were counted. Only repositories with at least one issue are considered.

As an output the script created a combined file, `repos_issues.csv`. Here some sample lines from this script:

```
id,url,description,language,forks,issues
3,https://api.github.com/repos/matplotlib/basemap,,C++,211,515
6,https://api.github.com/repos/cocos2d/cocos2d-x,cocos2d for C++,C++,5715,19559
```

Overall, 95576 repositories from GitHub have been created as output in CSV format.

To keep the submission size reasonable, only the resulting `repos_issues.csv` file has been provided. From the original source a small filtered subset is in the submission (`projects_sample.csv` and `issues_sample.csv`)

Data 4: TISS Lectures

The data has been parsed from TISS curriculum *Data Science* of winter term 2019 and consists of lecture descriptions, ECTS, lecturer information and occuring programming languages (extracted from the lecture description). The entry point of the parse task is the list of available curriculums [3] which is not available as simple HTML file. The endpoint presents a HTML based *shell* which loads the data from the backend via REST calls. To parse the information we used Python with Selenium to interact with the browser and extract the data. Parsing consists of lookups - for HTML tags with specific class names and content - and clicks - navigate to the interesting content - to extract the required information. Finally, each linked lecture in the data science curriculum is clicked, loaded and parsed to get the information.
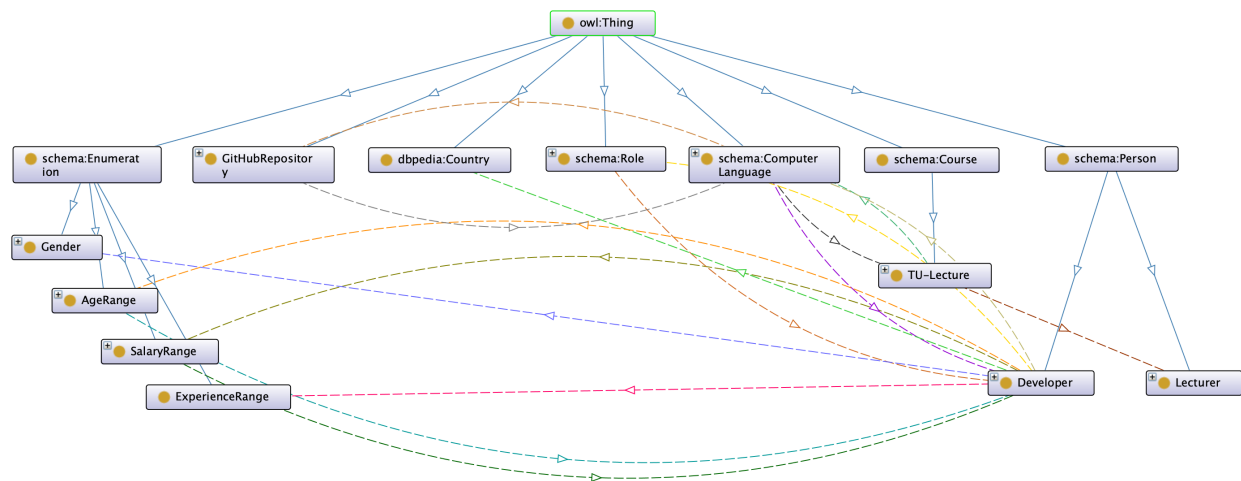
The data set does not contain specific information about programming languages which are dealt with in the lectures. To link this information we have used a set of regular expressions for identifying which languages are dealt with in which lecture.

Task 3: Ontology

When designing the ontology we were reusing existing vocabulary as much as possible.

- `schema:Enumeration`: Our enumerations (e.g. `Gender`, `AgeRange`, or `SalaryRange`) are subclasses of `schema:Enumeration`.
- `schema:gender`: The (functional) attribute specifying the gender of a person is taken the `schema` vocabulatory.
- `schema:homeLocation`: The (functional) attribute specifying the home country of a person is taken from the `schema` vocabulatory.
- `schema:instructor`: The (functional) attribute specifying the lecturer of a lecture is taken from the `schema` vocabulatory.
- `schema:name`: The data property specifying the name of an instance is taken from the `schema` vocabulatory.

---

[3] https://tiss.tuwien.ac.at/curriculum/studyCodes.xhtml?locale=en, last seen on 2020-02-10

Figure 2: Ontology Diagram

- `dbpedia:Country`: Countries are matched with corresponding countries from `dbpedia`.
- `schema:ComputerLanguage`: For the programming languages we use entities of `schema:ComputerLanguage`.
- `schema:Course`: The class `TU-Lecture`, representing a lecture at TU Wien, is modelled as a subclass of `schema:Course`.
- `schema:Person`: The two types of person we deal with, `Developer` and `Lecturer`, are modelled as subclasses of `schema:Person`.
- `schema:Role`: The developer and data scientiest roles are modelled as subclasses of `schema:Role`.

We have specified attributes like inverse, functional or range and domain for all relations.
The ontology developed is depicted in Figure 2. The resulting RDF file is stored in `group1.rdf`.

Task 4: Knowledge Graph

Kaggle ML & DS Survey

The semi-automatic generation of the schema data was done in `preprocess_kaggle.ipynb` *Part I - Analyse and Generate Metadata.*
In order to deal with white spaces in URIs we decided to convert blanks to underscores, and other special characters have been url encoded. The job roles, programming languages and genders have been mapped to aligned ontology as designed in `mappings.md`. The income/salary was modeled as ordinal ranges. The resulting RDF instances for the salary ranges were manually added to the ontology file `group1.rdf`.

The Jupyter Notebook `preprocess_kaggle.ipynb` also contains the fully automatic generation of individuals to Turtle N-Quad format in *Part II - Generate Instances.* Some country names have been mapped to match dbpedia, e.g. *United States of America* to *United States.*

StackOverflow User Survey

Every script used for the StackOverflow survey is located in `stackoverflow-survey-2018/individuals/scripts`. There is a `main.py` script that generates RDF-like files, as XML fragments, for e.g. country individuals, developer role individuals and age ranges. While these files cannot be loaded directly into a triple store, we have manually added the corresponding lines into the main RDF file, loaded it into ProtÃ©gÃ©, and then exported the resulting `group1.rdf` file.

The following files were inserted in this way into `group1.rdf`:

- `generated_age_ranges.xml`
- `generated_comp_languages.xml`
- `generated_genders.xml`

The other generated XML fragments were not actually inserted into the knowledge graph as they were not needed.

There is also a script `csv2ttl.py` that generates a Turtle file of the CSV file (filename: `stackoverflow_individuals.ttl`). It creates statements for each row in the results file. This Turtle file can then be imported into the Apache Jena triple store.

GitHub Repositories Data

The CSV file created from the data gathering and compilation, as described in the Task 2 description, is processed using a Python script called `RDFize.py` to create a Turtle RDF file. Below the RDF Turtle representation of two repositories:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix schema: <http://schema.org/>
@prefix group1: <http://www.semanticweb.org/sws/ws2019/group1#>
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>

<https://api.github.com/repos/matplotlib/basemap> rdf:type group1:GitHubRepository ;
    group1:isDevelopedIn group1:Cplusplus ;
    schema:name "matplotlib/basemap"^^xsd:string ;
    group1:issues "515"^^xsd:integer ;
    group1:popularity "211"^^xsd:integer .

<https://api.github.com/repos/cocos2d/cocos2d-x> rdf:type group1:GitHubRepository ;
    group1:isDevelopedIn group1:Cplusplus ;
    schema:name "cocos2d/cocos2d-x"^^xsd:string ;
    group1:issues "19559"^^xsd:integer ;
    group1:popularity "5715"^^xsd:integer .
```

The most tedious task of the cleanup was to make the programming languages of GitHub match with these from the other data sources. E.g. what in GitHub is written as `C++` appears in the other data sources as `Cplusplus`.

TISS Lectures

The script which is extracting the data is directly exporting into Turtle format via RDFLib[4] and writes the results to disk (file: `tuwel-data-science.ttl`). This output was directly imported into Apache Jena/Fuseki server via the provided web interface.

Two example lectures extracted and exported into TTL format:

```
@prefix ns1: <http://schema.org/> .
@prefix ns2: <http://www.semanticweb.org/sws/ws2019/group1#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<https://tiss.tuwien.ac.at/course/courseDetails.xhtml?courseNr=188399&semester=2019W>
    a ns2:TU-Lecture ;
    ns1:name "Introduction to Semantic Systems" ;
```

---

[4]https://github.com/RDFLib/rdflib, last seen on 2020-01-10

```
    ns2:courseId "188.399" ;
    ns2:description "2019W, VU, 2.0h, 3.0EC" ;
    ns2:ects 3.0 ;
    ns2:instructor ns2:Ekaputra_Fajar_Juang,
        ns2:Kiesling_Elmar,
        ns2:Sabou_Reka_Marta,
        ns2:Tjoa_A_Min .

<https://tiss.tuwien.ac.at/course/courseDetails.xhtml?courseNr=188995&semester=2019W>
    a ns2:TU-Lecture ;
    ns1:name "Data-oriented Programming Paradigms" ;
    ns2:courseId "188.995" ;
    ns2:dealsWith ns2:Python ;
    ns2:description "2019W, VU, 2.0h, 3.0EC" ;
    ns2:ects 3.0 ;
    ns2:instructor ns2:BÃ¶ck_Sebastian,
        ns2:Hanbury_Allan,
        ns2:Kiesling_Elmar,
        ns2:Piroi_Florina_Mihaela .
```

Task 5: Triple Store

For storing the data we were using Jena. We have installed an instance on the Internet for easier collaboration and app deployment.

The upload to Apache Jena Fuseki server was performed using the web interface, which turned out to be easy. Biggest problem here was error reporting. Fuseki is not very verbose in reporting errors on the UI, even though a helpful error message could often be found in the logfiles.

Construct Queries

Internal Enrichment

We created `CONSTRUCT` queries to enrich the knowledge graph with inverse relations to the ones created by the scripts.

Here one example:

```
PREFIX group1: <http://www.semanticweb.org/sws/ws2019/group1#>

CONSTRUCT {
  ?ageGroup group1:ageHasDevelopers ?developer
}
WHERE {
  ?developer a group1:Developer ;
             group1:hasAgeRange ?ageGroup .
  ?ageGroup a group1:AgeRange .
}
LIMIT 25
```

A `CONSTRUCT` query creates triplets, however these triplets are not stored back into the knowledge graph but only returned as results. For persisting the enriched knowledge one can use a *SPARQL 1.1* `INSERT DATA` query:

```
PREFIX group1: <http://www.semanticweb.org/sws/ws2019/group1#>

INSERT {
```

```
  ?ageGroup group1:ageHasDevelopers ?developer
}
WHERE {
  ?developer a group1:Developer ;
             group1:hasAgeRange ?ageGroup .
  ?ageGroup a group1:AgeRange .
}
```

External Enrichment

We were linking with `dbpedia` to get the name of the countries in English and German.

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX schema: <http://schema.org/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

CONSTRUCT
{ ?country rdfs:label ?countryName }
WHERE {
  { SELECT distinct ?country
    WHERE {
      ?developer schema:homeLocation ?country .
    }
  }
  SERVICE <https://dbpedia.org/sparql> {
    ?country rdfs:label ?countryName .
    FILTER (lang(?countryName) = 'en' || lang(?countryName) = 'de')
  }

}
```

A similar `INSERT` query has been used to store the additional triplets in the knowledge graph.

Uploads to Jena Fuseki

We created a persistent dataset on Apache Jena Fuseki, called `jobsta`. All data was uploaded into the default graph. The following files were uploaded, in this order, into the Triple Store:

- `group1.rdf` - the ontology and salary ranges, total 538 triples
- `kaggle-survey-2019/kaggle_individuals.ttl` - Kaggle survey data, total 147,574 triples
- `stackoverflow-survey-2018/individuals/generated/stackoverflow_individuals.ttl` - StackOverflow survey data, total 8,650 triples,
- `github-data/repos.ttl` - GitHub repositories, total 477,800 triples
- `tuwel-scraping/tuwel-data-science.ttl` - TISS lectures, total 1,946 triples

In addition, the following queries were inserted into the *query* text box of Apache Jena Fuseki:

- `internal-enrichment.qry` - creation of triples for inverse properties
- `external-enrichment.qry` - creation of triples for country names

The resulting folder on the server had a size of 133MB, and the dataset contains 787,400 triples.

Task 6: SPARQL Queries

We have developed the following set of queries. Unfortunately we worked as a team here, and it's not possible to assign queries to specific participants.

Query 1: Which languages are dealt with in lectures at TU Wien?

```
PREFIX group1: <http://www.semanticweb.org/sws/ws2019/group1#>

SELECT distinct ?language
WHERE {
  ?lecture group1:dealsWith ?language .
}
ORDER BY ?language
LIMIT 25
```

Query 2: As a student I want to learn language *(programming language)*. Which courses should I take?

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX group1: <http://www.semanticweb.org/sws/ws2019/group1#>
PREFIX schema: <http://schema.org/>

SELECT ?name
WHERE {
  ?lecture schema:name ?name .
  ?lecture group1:dealsWith ?programming_language .
  FILTER (?programming_language = ${language})
}
ORDER BY ?name
LIMIT 25
```

In this query `${language}` has to be replaced with the URI of a specific programming language, e.g. `group1:Python`.

Query 3: Which developer roles are most frequently specified in the surveys?

```
PREFIX group1: <http://www.semanticweb.org/sws/ws2019/group1#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT distinct ?role ?name
WHERE {
  ?developer group1:hasRole ?role .
  ?role rdfs:label ?name
}
GROUP BY ?role ?name
ORDER BY DESC(COUNT(?developer))
LIMIT 25
```

Query 4: Which countries are either in the European Union, or are most frequently specified in the surveys?

```
PREFIX group1: <http://www.semanticweb.org/sws/ws2019/group1#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX schema:<http://schema.org/>
PREFIX dbpedia:<http://dbpedia.org/resource/>

SELECT distinct ?name ?country WHERE {
  {
    SELECT distinct ?name ?country WHERE {
```

```
      ?developer schema:homeLocation ?country .
      ?country rdfs:label ?name .
      FILTER (lang(?name) = 'en')
    }
    GROUP BY ?name ?country
    ORDER BY DESC(COUNT(?developer))
    LIMIT 10
  }
  UNION
  {
    SELECT distinct ?name ?country WHERE {
      {
        SELECT distinct ?country WHERE {
          ?developer schema:homeLocation ?country .
        }
      }
      ?country rdfs:label ?name .
      SERVICE <https://dbpedia.org/sparql> {
        ?country <http://purl.org/dc/terms/subject> dbpedia:Category:Member_states_of_the_European_Unior
      }
      FILTER (lang(?name) = 'en')
    }
    GROUP BY ?name ?country
    ORDER BY ASC(?name)
  }
}
ORDER BY ?name
```

Query 5: Which programming languages are most often specified in the surveys?

```
PREFIX group1: <http://www.semanticweb.org/sws/ws2019/group1#>

SELECT distinct ?language
WHERE {
  ?developer group1:developsIn ?language .
}
GROUP BY ?language
ORDER BY DESC(COUNT(?developer))
LIMIT 25
```

Query 6: As a *(profession)* I live in *(country)* and I can program in *(programming language)* and I want at least *(amount)* USD per year. Can I achieve this in my home country?

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX group1:<http://www.semanticweb.org/sws/ws2019/group1#>
PREFIX schema:<http://schema.org/>
PREFIX dbpedia:<http://dbpedia.org/resource/>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX sc:<http://purl.org/science/owl/sciencecommons/>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>

ASK
WHERE {
```

```
?developer a group1:Developer .
?developer schema:homeLocation ?country .
?developer group1:developsIn ?language .
?developer group1:hasRole ?role .
{
  SELECT ?country (AVG(?avgRange) as ?averageK)
  WHERE {
      ?developer a group1:Developer .
      ?developer group1:hasRole ?role .
      ?developer schema:homeLocation ?country .
      ?developer group1:developsIn ?language .
    ?developer group1:hasSalaryRange ?salaryRange .
    ?salaryRange group1:minSalary ?minSalary .
    ?salaryRange group1:maxSalary ?maxSalary.
    BIND ((?minSalary + ?maxSalary)/2 AS ?avgRange)
  }
  GROUP BY ?country
}
{
  SELECT ?country (AVG(?salaryValue) as ?averageS)
  WHERE {
    ?developer a group1:Developer .
    ?developer group1:hasRole ?role .
    ?developer schema:homeLocation ?country .
    ?developer group1:developsIn ?language .
    ?developer group1:salary ?salaryValue .
  }
  GROUP BY ?country
}
BIND ((?averageK + ?averageS)/2 as ?average)
FILTER (?language = ${language} && ?country = ${country} && ?average > "${salary}"^^xsd:integer && ?r
}
GROUP BY ?country ?average
```

In this query `${language}` has to be replaced with the URI of a specific programming language,
e.g. `group1:Python`, `${country}` with the URI of a specific country, e.g. `dbpedia:Austria`,
`${salary}` with a yearly salary in USD, e.g. 50000, and `${role}` with the URL of a specific role,
e.g. `group1:Software_Engineer`.

Query 7: As a *(profession)* I can program in *(programming language)* and I want at least *(amount)* USD per year. In which countries can I achieve this, which has the highest prospect?

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX group1:<http://www.semanticweb.org/sws/ws2019/group1#>
PREFIX schema:<http://schema.org/>
PREFIX dbpedia:<http://dbpedia.org/resource/>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX sc:<http://purl.org/science/owl/sciencecommons/>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>

SELECT ?name
WHERE {
  ?developer a group1:Developer .
```

```
    ?developer schema:homeLocation ?country .
    ?developer group1:developsIn ?language .
    ?developer group1:hasRole ?role .
    {
      SELECT ?country (AVG(?avgRange) as ?averageK)
      WHERE {
        ?developer a group1:Developer .
        ?developer group1:hasRole ?role .
        ?developer schema:homeLocation ?country .
        ?developer group1:developsIn ?language .
        ?developer group1:hasSalaryRange ?salaryRange .
        ?salaryRange group1:minSalary ?minSalary .
        ?salaryRange group1:maxSalary ?maxSalary.
        BIND ((?minSalary + ?maxSalary)/2 AS ?avgRange)
      }
      GROUP BY ?country
    }
    {
      SELECT ?country (AVG(?salaryValue) as ?averageS)
      WHERE {
        ?developer a group1:Developer .
        ?developer group1:hasRole ?role .
        ?developer schema:homeLocation ?country .
        ?developer group1:developsIn ?language .
        ?developer group1:salary ?salaryValue .
      }
      GROUP BY ?country
    }
    ?country rdfs:label ?name .
    BIND ((?averageK + ?averageS)/2 as ?average)
    FILTER (?language = ${language} && ?average > "${salary}"^^xsd:integer
            && ?role = ${role} && lang(?name) = 'en')
}
GROUP BY ?country ?average ?name
ORDER BY DESC(?average)
LIMIT 1
```

In this query `${language}` has to be replaced with the URI of a specific programming language, e.g.
`group1:Python`, `${salary}` with a yearly salary in USD, e.g. 50000, and `${role}` with the URL of a specific
role, e.g. `group1:Software_Engineer`.

Query 8: As a student, I live in *(country)*. I want to learn a language which offers the most salary prospect in my country. Which languages shall I consider?

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX group1:<http://www.semanticweb.org/sws/ws2019/group1#>
PREFIX schema:<http://schema.org/>

SELECT distinct ?language
WHERE {
  ?developer a group1:Developer .
  ?developer schema:homeLocation ${country} .
  ?developer group1:developsIn ?language .
  ?developer group1:hasSalaryRange ?salaryRange .
```

```
  ?salaryRange group1:minSalary ?minSalary .
  ?salaryRange group1:maxSalary ?maxSalary .
  BIND ((?minSalary + ?maxSalary)/2 AS ?avgRange)
}
GROUP BY ?language
ORDER BY DESC(AVG(?averageK))
LIMIT 25
```

In this query `${country}` has to be replaced with the URL of a specific country, e.g. `dbpedia:Austria`.

Query 9: Which programming languages are most often used in GitHub repositories?

```
PREFIX group1:<http://www.semanticweb.org/sws/ws2019/group1#>
PREFIX schema:<http://schema.org/>

SELECT distinct ?language
WHERE {
  ?repository group1:isDevelopedIn ?language .
  ?language a schema:ComputerLanguage .
}
GROUP BY ?language
ORDER BY DESC(COUNT(?repository))
LIMIT 25
```

Query 10: I want to practise language *(programming language)*. Which repositories should I look at, as they are most popular and written in this language?

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX group1:<http://www.semanticweb.org/sws/ws2019/group1#>
PREFIX schema:<http://schema.org/>

SELECT ?name
WHERE {
  ?repository group1:isDevelopedIn ${language} .
  ?repository group1:popularity ?popularity .
  ?repository schema:name ?name .
}
ORDER BY DESC(?popularity)
LIMIT 25
```

In this query, `${language}` has to be replaced with the URI of a specific programming language, e.g. `group1:Csharp`.

Query 11: Which repositories in GitHub have more than 100 issues and are most popular?

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX group1:<http://www.semanticweb.org/sws/ws2019/group1#>
PREFIX schema:<http://schema.org/>

SELECT ?name
WHERE {
  ?repository group1:popularity ?popularity .
  ?repository group1:issues ?issues .
```

```
  ?repository schema:name ?name .
  FILTER (?issues > 100)
}
ORDER BY DESC(?popularity)
LIMIT 25
```

Query 12: For which programming languages exist more than 1,000,000 repositories, counting in the number of forks? How popular were they in the surveys?

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX group1:<http://www.semanticweb.org/sws/ws2019/group1#>
PREFIX schema:<http://schema.org/>

SELECT ?language ?totalRepos (COUNT(?developer) as ?developerCount)
WHERE {
  { SELECT distinct ?language (SUM(?popularity) as ?totalRepos)
    WHERE {
      ?repository group1:popularity ?popularity ;
                  group1:isDevelopedIn ?language ;
                  group1:issues ?issues .
    }
    GROUP BY ?language
    HAVING (SUM(?popularity) > 1000000)
  }
  ?developer group1:developsIn ?language .
}
GROUP BY ?language ?totalRepos
LIMIT 25
```

Query 13: Who are the top lecturers in regard to the number of ECTS taught?

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX group1: <http://www.semanticweb.org/sws/ws2019/group1#>
PREFIX schema: <http://schema.org/>

SELECT ?name ?url (SUM(?ects) as ?sum)
WHERE {
  ?lecture a group1:TU-Lecture .
  ?lecture group1:instructor ?instructor .
  ?lecture group1:ects ?ects .
  ?instructor schema:name ?name .
  ?instructor schema:URL ?url
}
GROUP BY ?name ?url
ORDER BY DESC(?sum)
LIMIT 25
```

Query 14: What are the top lectures in regard to the number of ECTS?

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX group1: <http://www.semanticweb.org/sws/ws2019/group1#>
PREFIX schema: <http://schema.org/>

SELECT ?name ?lecture ?ects
```

```
WHERE {
  ?lecture a group1:TU-Lecture .
  ?lecture schema:name ?name .
  ?lecture group1:ects ?ects .
}
ORDER BY DESC(?ects) ?name
LIMIT 25
```

Query 15: What are the top languages taught according to the number of lectures that deal with them?

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX group1: <http://www.semanticweb.org/sws/ws2019/group1#>
PREFIX schema: <http://schema.org/>

SELECT ?language (COUNT(?lecture) as ?lecture_count)
WHERE {
  ?lecture group1:dealsWith ?language
}
GROUP BY ?language
LIMIT 25
```

Task 7: App Implementation

The app is a simple mobile-friendly web-based application. We used Bootstrap for the styling, jQuery for the DOM manipulation and `fetch` together with `async/await` for HTTP calls.

The interaction with Apache Jena is using SOH - *SPARQL over HTTP*.

To avoid dealing with *CORS* (Cross-Origin Resource Sharing) issues, we have deployed the app in the same server as Apache Jena. It is accessible at https://jena.helmuth.at/app. For security reasons we have added username and password protection. Please use the following credentials to access the app (https://jena.helmuth.at/app) or the Jena Server (https://jena.helmuth.at/):

- **Username:** group1
- **Password:** gurus2020

Auxiliary Queries

The dropdown select boxes receive their options through separate *SPARQL* queries.

Country Lists

When the list of possible countries is presented, the following logic is used:

- show the 10 countries with the most developers
- show all member states of the European Union

For this, a federated search is used, using the attribute `dbpedia:Category:Member_states_of_the_European_Union` from dbpedia.

Programming Languages & Roles

When showing a select list of programming languages, the values which are present for the main query are displayed, usually sorted by usage and limited to 25 values.
For example, for *Study*, the list of programming languages is restricted to those for which actual lectures exist.

Similarly for roles, the 25 most frequently specified roles (according to the surveys) in the world are offered as options in the dropdown boxes.

Location Query

For the *Location* part of the application we have implemented *two* main queries. The first `ASK` query identifies whether the desired income level in the home location is feasible. If not, it identifies with a `SELECT` query countries with sufficient income level, and from these the one country with the highest income level.

Depending on the results the answer is then one of:

- **You should stay!** - should the desired level be achievable in the home country
- **You should go! I suggest *country* for a try.** - should the desired level only be achievable in another country than the home country
- **Sorry to break the news - that's not realistic anywhere.** - should the desired level not be achievable in any country

Source Code

The source code of the app is in the submission package.
At the end we were not offering the code in a public repository, as the survey data is not licensed for redistribution. Therefore the corresponding button on the app is not working.