WEB INFORMATION SYSTEMS

Link Rec Verslag

Kobe Werner (1952857)
Senne Wertelaers (2158088)
Wolf Moyaers (2158257)

Year 2024-2025

1 Introduction and requirements

1.1 Project goals

With LinkRec, we want to create a web service that can be used for HR teams in Belgium. Companies can post job vacancies on the platform. The platform then uses its job-matching algorithm to show the vacancy to the relevant users of the platform. Users can enlarge their professional network by making connections to other users on the platform.

1.2 Usage scenarios

This section outlines the usage scenarios of the LinkRec system. The following actors are used in the use cases.

- John: New user creating and updating profile, seeking jobs
- Mary: Existing user connecting with others, seeking jobs
- Emazon: Employer posting job, searching for candidates
- Ethan: Malicious actor attempting unauthorized access

1.2.1 Authentication

- User Registration: John visits the LinkRec website and creates a new account using his email address and a secure password.
- User Login: Mary logs into her LinkRec account using her username and password.
- Password Recovery: John forgets his password and requests a password reset via email. He follows the instructions in the email to set a new password.

1.2.2 Personal Profile

- Create Profile: After registering, John creates his personal profile by adding his name, profile picture, website, living area, phone numbers, and email addresses.
- Update Profile: Mary updates her professional experience section by adding her latest job. She also removed an old email address that she no longer uses.

1.2.3 Authorization

- View Profile: Ethan tries to view Mary's private profile information, but the system denies access because Ethan is unauthorized.

1.2.4 User-to-User Connections

- Send Connection Request: Mary discovers John's profile and sends him a connection request.
- Accept Connection Request: John receives a notification about Mary's connection request and accepts it. Mary has now been added to John's connections list.

1.2.5 Vacancy Postings

- Create Job Posting: Emazon creates a new job posting for a web developer position, specifying the job title, description, location, required skills, and application deadline.
- Update Job Posting: Emazon updates the job posting to extend the application deadline by a week.

1.2.6 Job Matching Algorithm

- Employer Searches for Candidates: Emazon uses the LinkRec system to search for candidates matching their web developer job posting. The system returns a list of matching profiles, including John's.
- User Searches for Jobs: Mary, who is seeking a new job opportunity, uses LinkRec to search for job postings matching her skills and experience. The system returns a list of relevant vacancies.

1.3 Contributions

- Kobe
  - Ontology
  - Job matching algorithm
  - Inferencing
  - Phase 2 Report
- Senne
  - Backend
  - Interface between Fuseki and GraphQL
  - Phase 1 Report
- Wolf
  - Ontology
  - Interface between Fuseki and GraphQL
  - Phase 1 Report

2 Summary of chosen technologies and platforms

- Fuseki -> RDF database
- Apollo webserver for the backend
- Typescript as language
- SPARQL HTTP client
- GraphQL library from node_modules

3 Ontology

Geonames were used to get location data from Belgium. These locations include the districts (Flanders, Wallonia, Brussels), the provinces, the arrondissements, and the cities. Each of these is stored in a hierarchy. This makes it possible to infer if a city is in a certain province, so we used this dataset.

Wikidata also provided us with a lot of data. From Wikidata, we took the languages (Q33742). They also have a large dataset of possible professions (Q28640); these also have subclasses of the main profession class. With these subclasses, we can infer that different jobs are in the same categories. Lastly, we also obtained the list of master (Q183816) and bachelor (Q163727) degrees. The only inference that could be made about the degrees is whether they are master's or bachelor's degrees.

Besides reusing existing ontologies, we also wrote our own. The first is our user; this has a couple of properties related to the info needed to be stored. These include the ID given by the server, an email, first and last name, a phone number, a web page, and a gender. All these personal properties are linked with the matching foaf properties. Furthermore, it includes several CV properties such as the user's location, the languages they speak, whether or not they are looking for a job, any prior job experiences, and their education. All these properties are used to infer whether a user matches a job vacancy.

The job-seeking status of users has three options: actively looking, open to offers, or not looking. The user only gets matched with jobs when they are looking for or open to offers.

For the employer we have a different class, this was needed for security reasons to differentiate between users and employers. These employers also get linked to job vacancies. Apart from the job vacancies, they also have properties for an email, first and last name, phone number, and a webpage. These properties are linked with both the foaf and vcard ontology.

Experiences are composed of multiple properties: the job title the user had, in what profession the user has experience, at which company the user worked, a short description of what the job entailed, and how long the user worked the job. From the time the user worked at the job, the experience level is inferred.

Since the user can have an education, this is also defined in the ontology. An education consists of a title, a degree, and the institution where the title was obtained.

Then, we have the job/vacancy class. As the name suggests, these depict vacancies on the platform. Each vacancy has a title, a location, and a list of requirements. The requirements can contain several different properties. This can be a previous profession with a number of years of experience; this can also be an education or a degree needed. Lastly, it can be a language that the user must speak. Each of these properties can also contain a description providing more information to the user.

4 Matching algorithm

There are two steps to the matching algorithm. The first step is matching a user to a requirement. This is completely done by inferencing the data. The inferencing is done in such a way that all the parts of the requirement are optional, but if it is specified, the user should also match it. Due to the use of geonames and Wikidata, a lot of inferencing is done to provide extra features. An example of this is that the job can have a province as location; due to the inferencing, all cities that are inside that

province also get matched. The second step is the combining of the user with the jobs; this is done on the backend of LinkRec.

5 Implementation & API

The application implements a GraphQL API that interfaces with an RDF triple store, focusing on user profile management and job matching. Here's a breakdown of the core functionalities:

The system uses RDF as its data store, with custom ontologies defined for users, jobs, experiences, and requirements. The data modeling follows semantic web principles, incorporating existing ontologies like FOAF and vCard for standard profile information, while extending with domain-specific predicates for job-seeking status and professional experiences. Wikidata references are used for standardized entities like languages and professions, enabling rich data integration.

The API exposes a GraphQL schema that abstracts the complexity of RDF queries. Core types include User, Experience, Education, and Job, with resolvers that translate between GraphQL operations and SPARQL queries. A custom type system was implemented to handle RDF term resolution, supporting both simple scalar values and complex nested objects. This allows for efficient querying of related entities while maintaining type safety.

The API supports standard CRUD operations through mutations like createUser, updateUser, and deleteUser, while queries enable complex data retrieval with filtering and relationship traversal. Features include user profile management, experience tracking, and job matching based on user qualifications and job requirements.

The system implements a custom resolver pattern that allows each complex type (User, Experience, etc.) to execute its own SPARQL queries, enabling efficient data fetching while maintaining separation of concerns. External services integration, particularly with Wikidata, is handled through SPARQL federation, allowing real-time resolution of standardized entity labels and relationships.

6 Observations on your developments

While quite a lot didn't work as expected some aspects did. One of these was setting up GraphQL. Due to the frameworks used, the API and web interface were easily up and running. This provided us with a solid foundation to further implement the API and an easy to debug environment. This is not something we would change in hindsight.

Something we would change is the user authentication. Currently there are two types of accounts, there are the users and the employers. Since they were implemented as two different types it proved challenging to implement the propper security measures for both. If we had to redo the authentication we would add user roles, this way we only have to implement authentication for one class.

The main thing that didn't work as expected was RDF. We chose to use Fuseki as the RDF database. This was the biggest mistake. The documentation didn't help at all. Furthermore did it not properly work with docker. After all these problems were fixed we could finally start with implementing the ontology.

Implementing the ontology was also not as expected. While RDF tries to provide structure we found it difficult to actually create those structures. We struggled a lot and still don't have the best ontology.

We could definitely have done this differently by first properly thinking and designing the relations between all the classes.

7 Conclusion

The project proved difficult for us all. Furthermore did it not fall within our interests area. We learned that the semantic web is quite difficult to master. We also believe that it isn't really used any more since most of the resources about the topic where for legacy purposes or from a couple years ago.