FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Antonín Jareš

Simplifying access to linked data using tabular views

Department of Software Engineering

Supervisor of the master thesis:	RNDr. Jakub Klímek, Ph.D.
Study programme:	Informatics
Study branch:	Artifical Intelligence

Prague 2021

I would like to thank my supervisor, RNDr. Jakub Klímek, Ph.D., for the provided guidance, constructive feedback and productive discussions. Also for the patience towards the constant delays and how long this whole process took.

I would also like to thank my family, people who supported me, and people whom I have neglected, for supporting and dealing with me over the very long process of working on this thesis.

Title: Simplifying access to linked data using tabular views

Author: Antonín Jareš

Department: Department of Software Engineering

Supervisor: RNDr. Jakub Klímek, Ph.D., Department of Software Engineering

Abstract: The goal of this thesis is to design and implement a front-end application allowing users to create and manage custom views for arbitrary linked data endpoints. Such views will be executable against a predefined SPARQL endpoint and the users will be able to retrieve and download their requested data in the CSV format. The users will also be able to share these views and store them utilizing Solid Pods. Experienced SPARQL users will be able to manually customize the query. To achieve these goals, the system uses freely available technologies – HTML, JavaScript (namely the React framework) and CSS.

Keywords: dataset, linked data, LOD, RDF, CSV

Contents

1 Introduction 4
1.1 What is Linked Data? 4
1.2 What is a Solid Pod? 5
1.3 Goals 6
1.4 Thesis structure 6

2 Analysis 8
2.1 Target audience 8
2.2 User roles 8
2.3 Requirements 9
2.3.1 Functional requirements 9
2.3.2 Non-functional requirements 12
2.4 Use cases 13
2.4.1 Administrator 14
2.4.2 User 15
2.5 Application inputs 16
2.5.1 SPARQL endpoint 16
2.5.2 Data schema 16
2.6 Existing solutions 18
2.6.1 Comparison criteria 18
2.6.2 Solution comparison 19

3 Design 23
3.1 Type of application 23
3.1.1 Web application advantages: 23
3.1.2 Desktop application advantages: 24
3.1.3 Conclusion 24
3.2 Storage options 24
3.2.1 Local file system 24
3.2.2 Cloud solutions 25
3.2.3 Solid Pods 25
3.2.4 Chosen solution 26
3.3 Language 26
3.3.1 Libraries & Frameworks 26
3.4 Application architecture 27
3.4.1 View layer 28
3.4.2 Application logic 28
3.4.3 Persistent data handling, Solid Pods 29
3.4.4 SPARQL Editor 29
3.4.5 Overview 29
3.5 Mockups 30
3.5.1 Title page 31
3.5.2 User information 31
3.5.3 Data area 32
3.5.4 Query editor 34

4 Implementation **35**
4.1 Limited mobile experience 35
4.2 State handling with Ramda 36
4.3 SPARQL Proxy 36
4.4 Interesting implementation tasks 37
4.4.1 SPARQL generation 37
4.4.2 Cartesian product detection 41
4.4.3 Optionality cycles 43
4.5 Room for improvement 44
4.5.1 Graph tool 44
4.5.2 Query improvements 45

5 Documentation **48**
5.1 User documentation 48
5.1.1 Solid Pod setup 48
5.1.2 UI Elements 50
5.1.3 Graph interface 59
5.1.4 List view 64
5.1.5 Examples 70
5.2 Administrator documentation 92
5.2.1 Prerequisites 92
5.2.2 Data schema creation 93
5.2.3 SPARQL proxy 96
5.2.4 Deployment 97
5.2.5 Application parameters 98
5.3 Programmer documentation 99
5.3.1 Prerequisites 100
5.3.2 Local development setup 100
5.3.3 Overview 100
5.3.4 Implementation examples 102
5.3.5 Automatically generated documentation 105

6 Tests **106**
6.1 Unit testing 106
6.1.1 Libraries 106
6.1.2 Covered code 107
6.2 Manual test scenarios 107
6.2.1 Preliminaries 108
6.2.2 T1 - Visualization and simple configuration 108
6.2.3 T2 - Language selection 109
6.2.4 T3 - Offline capabilities 111
6.2.5 T4 - Data handling 112
6.2.6 T5 - SPARQL Capabilities 115
6.2.7 T6 - Graph interactions 116

7 Evaluation **117**
7.1 Goal fulfillment 117
7.2 System usability scale results 117

8 Conclusion **119**

Bibliography **120**

List of Figures **121**

List of Tables **125**

A Solid Pod troubleshooting **126**
A.1 Logging in with Solid Pod 126
A.2 Enabling trusted apps 126
A.3 Creating a save destination 127

B Resources **129**

1. Introduction

Today's day and age provides us with unprecedented access to information. Never in the history of humanity has such a vast amount of data been at our fingertips, ready to be inspected and sifted through at a moment's notice.

When navigating around this data, humans contextualize by nature without any additional effort exerted and make connections between topics and resources seamlessly. For example, should you ask a human in Great Britain what they think about London, all of them would be talking about the country's capital, not London, Ontario. While we see this as completely obvious, this is not the same for computers. Computers require all of the information to be either presented unambiguously, or additional non-trivial logic has to be implemented for the computer to be able to distinguish the information and make proper decisions.

There are methods, however, that aim to alleviate this and make the data more accessible to computers at large, one of them is called Linked Data.

This chapter describes what Linked Data is, the main goal of this thesis, the motivation behind it and the overall structure of this text.

1.1 What is Linked Data?

Formally, the term Linked Data was introduced by Tim Berners-Lee in 2006, see Menday and Mihindukulasooriya [2015]. It represents a method of publishing structured data that are interlinked in a unified manner.

In short, it is a unified way of providing data and its relations, which in turn create a literal Web of data called Semantic Web, see Berners-Lee et al. [2001], to make it more easily readable for computers.

Its four main principles are:

- Entities should be identified by distinct URIs.
- HTTP URIs should be used so that these entities are searchable
- When an entity is looked up, useful information should be provided using the open standards such as RDF, SPARQL.
- When referring to other linked entities on the Web their HTTP URI should be used.

Following the example of London, with Linked Data, the entity of London itself would be specified by an IRI that would be directly accessible with additional information available at the resource's location. For example Wikidata[1] contains entries for both, London, Great Britain[2], and London, Ontario[3]. If a computer accesses such a resource, it can then look for the country property[4] that is listed as a property on the resource itself, and check whether the London in

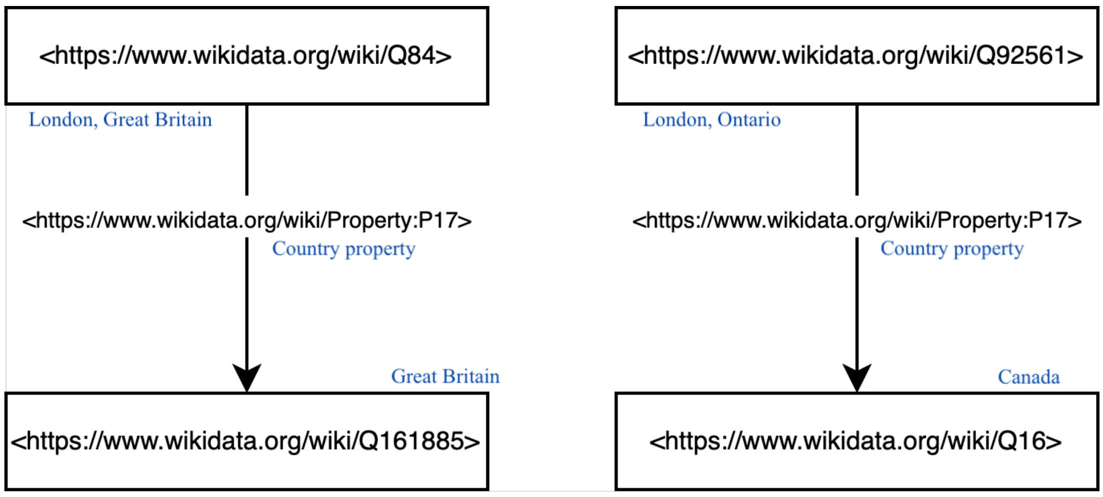
[1]`https://www.wikidata.org/wiki/`
[2]`https://www.wikidata.org/wiki/Q84`
[3]`https://www.wikidata.org/wiki/Q92561`
[4]`https://www.wikidata.org/wiki/Property:P17`

question is located in Great Britain[5] or Canada[6]. Great Britain and Canada are again specified by their IRIs with a list of their own properties, providing further information. This way, the computer can navigate seamlessly through the data unambiguously and with each step process more and more data.

The format used for Linked Data is called RDF, see Schreiber and Raimond [2014], which consists of triples of `<subject> <predicate> <object>`. All of this information can be imagined as a graph, with subjects and objects represented by nodes in the graph and edges representing appropriate relationships.

For the example of the two London cities, the graph could look like this:

Figure 1.1: London Linked Data example


<https://www.wikidata.org/wiki/Q84>
London, Great Britain
<https://www.wikidata.org/wiki/Property:P17>
Country property
Great Britain
<https://www.wikidata.org/wiki/Q161885>
<https://www.wikidata.org/wiki/Q92561>
London, Ontario
<https://www.wikidata.org/wiki/Property:P17>
Country property
Canada
<https://www.wikidata.org/wiki/Q16>

To be able to retrieve information from this graph notation, the users have to have an understanding of a specific language called SPARQL, see Seaborne and Harris [2013]. Queries written in SPARQL are then executed against the endpoint containing the data presented in the RDF format and the resulting data is retrieved. For obvious reasons, adoption of such an approach to the general public poses challenges and would require users interested in working with linked data to learn this language and its intricacies. To alleviate this, the users could be able to view the data as a graph directly and by interacting with the graph, they could be able to choose what data they would like to see.

The goal of this work is to provide the users with a visualization tool which would allow them to query basic data without the need to learn SPARQL or understand RDF themselves.

1.2 What is a Solid Pod?

Throughout the course of this work, an online data storage solution called Solid Pod[7] is mentioned multiple times as it is the option of choice for handling the application data. In short, a Solid Pod (personal online data store) is a place for users to store their data that also allows them to authorize with supporting

[5] `https://www.wikidata.org/wiki/Q161885`
[6] `https://www.wikidata.org/wiki/Q16`
[7] `https://solidproject.org/users/get-a-pod`

applications. More detailed description can be found further in the text under subsection 3.2.3.

1.3 Goals

The goal of this thesis is to design and implement a solution that will simplify users' interaction with Linked Data, allowing them to create, persist, and share their custom created views for given data sets. Utilizing such views, users can then query arbitrary data sets and receive results in the CSV format.

The desired features could be summed up as follows:

- Schema visualization

  The application will display specified linked data schemas in a graphical notation, providing users with an overview of the dataset and UI elements for further interaction.

- Defining tabular views

  Users will be able to intuitively create specific views against data sets that would yield only desired data.

- View storage

  Users will be able to save and import such views utilizing an existing cataloguing tool.

1.4 Thesis structure

The rest of the thesis is structured as follows:

Chapter 2 describes the given problematic in more detail. We discuss the proposed solution's requirements, different actor roles, the specific use cases of the proposed solution and provide overview of similar existing solutions.

Chapter 3 focuses on solution design. The technologies chosen to implement the proposed solution and the reasons behind these choices are described in this chapter alongside an overview of the architecture of the proposed solution.

Chapter 4 describes additional implementation details that were not or could not be taken into account during the design phase. These details include library specific implementations and further description of application layers interacting with each other. This chapter also contains some interesting problems that needed to be solved during the implementation phase and concludes with description of what could be improved.

Chapter 5 contains documentation for users, programmers and administrators. Users can find examples of usage in this chapter with description of all application elements. Administrators and programmers can read through more detailed description of the implementation and possibilities of application deployment and its further extension.

In chapter 6 we focus on the solution's quality assurance. Automatic tests put in place and manual test scenarios for users are described here.

Evaluation of the implemented solution can be found in chapter 7. It contains the overall goal fulfillment and additional feedback from external users in the form of System Usability Scale forms, as per Brooke [1996].

This work concludes with chapter 8 which contains author's look back on the work and thoughts about the whole process and its results. Following it are various lists and attachments this text references.

2. Analysis

This chapter provides closer look on the chosen problem area. It covers description of the target audience as well as the possible user roles in the application. The formal requirements are specified in the part following, ending with an overview of the current existing solutions and their comparison against the proposed solution of this work with regards to the requirements specified.

2.1 Target audience

The people this application is targeted at could be separated into three main groups as follows:

Data schema curators

For some use cases, the data schemas might provide data that would not be necessary. The curators would aim to manually edit the data schema presented in the application to create curated subsets of the provided data schemas that can be further used by other users without polluting their experience with arbitrary data that would be of no use.

For example a biology data schema could be split up into two parts regarding botany and zoology to allow users to view data schemas corresponding only to the botanical or animal kingdom, respectively.

People without RDF knowledge

The common users this application aims at are ”clerks” and people without RDF knowledge who might use this tool without the need to understand RDF or SPARQL. These people should be introduced to the application and should be explained some basic functionalities, which they can also achieve by themselves following the subsection 5.1.5 and reading through the section 5.1. They should also have an understanding of what the data schema they will be working with represents.

Result set consumers

When the resulting queries are catalogued or shared, some people might use the application only via these means for fetching the result sets. In this case, there is no need of understanding RDF, nor how the application works, as these users would only understand what the result set represents and would work with it accordingly.

2.2 User roles

From the application perspective, there are three possible types of users as follows:

Anonymous

A user who is in no way authenticated with the application. By default every user visiting the application falls under this category. These users shall be able to utilize the basic features of the application which don't require authentication.

Authenticated - Solid Pod user

A user who has authenticated with the application via a Solid Pod. Such users shall have additional features at their disposal, such as data persistence and file handling in their Solid Pod, and shall be recognized as the same user by the application across different sessions on different machines.

Administrator

A user managing and/or deploying the application. They shall be able to change the parameters of the application, allowing them to distribute the application with different data schemas on its initialization.

2.3 Requirements

This section describes formal granular requirements imposed on the application.

To evaluate the fulfillment of these requirements in the final application, we can follow Section 6.2 which cover all of the requirements specified.

Because of the nature of the development process for the implementation part of this work, the list of requirements laid out below had been updated multiple times over the course of the development phase, such requirements are marked with 🕒.

2.3.1 Functional requirements

This section describes the functional requirements, denoted by *F*, explaining what should be directly accessible for the users in the application, and its capabilities.

The requirements specified in this section and their details are a product of discussions held during the meetings with the supervisor of this work.

F1 - Schema visualization

The application shall provide the user with a overview of the provided data set in a graphical notation. Such visualization shall preserve all relationships specified in the data schema and shall provide simple, easy to navigate overview over it.

F2 - Application configuration

- F2.1 - SPARQL Endpoint

  The user shall be able to change the SPARQL endpoint against which the generated queries shall be run.

- F2.2 - Data schema

  The user shall be able to provide the URL of a data schema to be loaded in the application

- F2.3 - Human readable labels

  When provided by the endpoint, the application should allow the user to toggle between human readable names, provided by the endpoint, for entities and their direct IRI description.

- F2.4 - Interface language selection

  The user shall be able to select the application language. The publishing of this work shall come with English and Czech versions of the application included.

- F2.5 - Property language selection

  The user shall be able to select preferred language order for queried linked data properties supporting language selection. At the time of the publication of this work, this is **rdf:langString**[1].

- F2.6 - Resource label selection

  The user shall be able to select preferred language for human readable descriptions of properties and entities.

- F2.7 - Limited offline usability

  The user shall be able to use most of the features regarding data manipulation and interactions even when offline.

F3 - Generate SPARQL SELECT query

The user shall be able to select properties to be included in the resulting query by interacting with the graphical interface, which in turn would generate an appropriate SPARQL SELECT query representing the selection.

- F3.1 - Mark as optional

  The user shall be able to mark selected properties as optional.

- F3.2 - Hide value from result set

  The user shall be able to select whether the query results should display selected property values or omit them.

- F3.3 - Query customization

  The user shall be able to manually edit the SPARQL query, which should persist until the user changes their selection through interaction with the tools.

[1]`https://www.w3.org/TR/rdf-schema/#ch_langstring`

F4 - User data storage

- F4.1 - Login with a WebID

  The user shall be able to login to the application via a Solid Pod of their choice with their WebID.

- F4.2 - List views

  When logged in, the user shall be able to list files saved in their Solid Pod.

- F4.3 - Load view

  The user shall be able to load an application view from a public accessible URL or their Solid Pod when logged in.

- F4.4 - Save view

  When logged in, the user shall be able to save their created data views in their Solid Pod.

- F4.5 - Delete view

  When logged in, the user shall be able to delete views from their Solid Pod.

- F4.6 - Permissions 🕒

  The user shall be able to set private, public read/write permissions on the project files they save utilizing Solid Pods.

- F4.7 - Share capabilities 🕒

  The user shall be able to obtain a single URL containing their encoded selection and endpoint, allowing them to share and repeat the query easily and storing it.

- F4.8 - Direct result URL 🕒

  The user shall be able to obtain a single URL allowing for a direct download of the result set in .csv format.

  Having such a URL at their disposal, the user can then catalogue it as a CSV distribution in a catalogue of their choice, for example by the tool developed in Klímek and Skoda [2018], which is used by `https://data.gov.cz/`.

- F4.9 - Automatic save retries 🕒

  The user's project remote save action shall be automatically retried for a set amount of times, should it fail.

F5 - Run SPARQL query

- F5.1 - Run query

  The user shall be able to execute their generated SPARQL query against the specified endpoint.

- F5.2 - Result format

  The user shall be able to obtain the query results in the CSV format.

F6 - Application interaction

- F6.1 - Notify about cartesian product 🕒

  The user shall be notified if their selection does not create a strongly connected graph, which in turn could result in querying for a cartesian product.

- F6.2 - ”Hide the rest” functionality 🕒

  The user shall be able to hide all non-queried entities with a single action.

- F6.3 - Edge description 🕒

  The user shall be able to view information about edges connecting different entities.

- F6.4 - Entity duplication 🕒

  The user shall be able to duplicate entities in the schema, allowing them to query for recursive properties with a different target entity or differentiating two entity instances in the query.

- F6.5 - Entity deletion 🕒

  The user shall be able to delete entities from the data schema, allowing them to reduce the size of the schema by removing entities of no interest.

- F6.6 - Save edit 🕒

  When opening a remote project file, the user shall be asked to edit the original file, save it to their own location or not save it at the moment.

- F6.7 - Notify about custom SPARQL query 🕒

  The user shall be warned about the SPARQL query being manually edited, to prevent them from accidentaly interacting with the graph and removing this change.

2.3.2 Non-functional requirements

This section describes non-functional requirements, denoted by *N*, which lay out aspects of this work that are not directly related to its functionality and as such can not always be evaluated in a yes or no fashion.

N1 - Deployment

- N1.1 - Documentation

  Administrators shall be able to deploy the application easily by following corresponding installation steps.

- N1.2 - Bundle

  Administrators shall be provided with all necessary resources to deploy the application.

N2 - Demo scenarios

The application shall be accompanied by demo scenarios which the users could follow to get introduced to the basic functionalities of the application.

N3 - Manual test scenarios

The application shall be provided with manual test scenarios, allowing the testers to cover all the functional requirements targeted in subsection 2.3.1.

N4 - Unit tests

The application shall be provided with automatic test suites covering basic various code functionality.

N5 - Code documentation

The application shall be provided with a generated documentation describing the code functionalities.

N6 - Demo

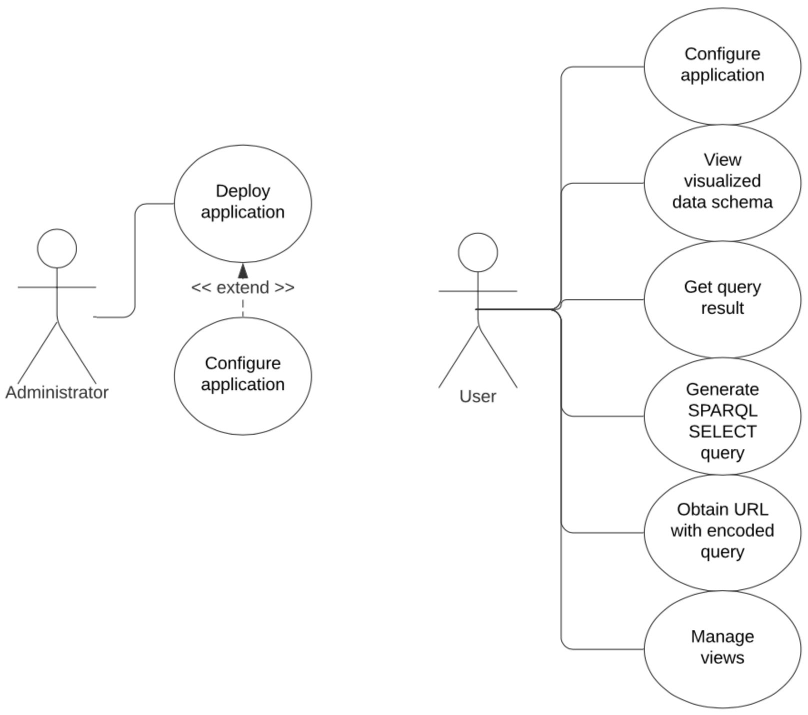
The application shall be hosted and available to users to test out without their need of deployment.

2.4 Use cases

The following section describes possible use cases for the application users and its administrators. Overview is provided by one use case diagram[2] accompanied by textual descriptions of each separate use case.

[2]`https://www.uml-diagrams.org/use-case-diagrams.html`

Figure 2.1: Use case diagram



2.4.1 Administrator

Administrators will take care of deploying the application and providing access to the end user. They will able to change the application configuration, resulting in different data sets being displayed to the user.

Deploy application

Administrators will be able to deploy the application utilizing common means of deployment for the chosen technology.

In the case of a web application, such an option could be adding an information about the data to the link directing the users to the application. The application would in turn parse the information during initialization and display the required data, effectively allowing the administrators to show users different data loaded just by a simple configuration change.

Configure application

A necessary requirement for the application to work properly is providing input data schema and endpoint against which the generated SPARQL queries will be run. The administrators can provide these encoded in the URL for the links which would be directing users to the application. The administrators can also opt deploy the application without this information, since the required inputs can also be edited inside the application.

- Endpoint

  A valid endpoint URL against which generated SPARQL queries can be run. E.g. `https://linked.opendata.cz/sparql`

- Data schema

  Data schema describing the catalogue in a specific format required by this application.

  An example of such a schema can be found at `https://jaresan.github.io/simplod/dataSchema.html`.

For simplicity, in the rest of the text, we will consider every mention of the application as if it were properly provided with these parameters, unless stated otherwise.

2.4.2 User

Users will care mostly about requesting the appropriate data and about being able to persist these selection. They might also be interested in using the application with their own setup of data schema and endpoint.

Configure application

Allows users to change the endpoint for the SPARQL queries and provide the data schema URL if the application is not provided with one.

View visualized data schema

Allows every user, regardless of their role, to be able to see the data schema provided visualized for easier navigation.

Generate SPARQL SELECT query

Users are able to create a SPARQL select query by utilizing visual tools in the application, there is no necessity of users to directly write SPARQL queries. We refer to every such set of criteria created by the user as a *view*.

Get query result

Users are able to execute their generated SPARQL query against the selected endpoint and retrieve the results of the generated SPARQL SELECT query in different formats.

Obtain URL with encoded query

Users are able to export their generated query and endpoint in a single URL to share and store in a data catalogue.

Manage views

Anonymous users are able to import views from other sources via a URL, provided the URL is publicly accessible. Additionally allows authorized users to save their created views, list all views they currently have saved and delete them.

2.5 Application inputs

As mentioned in section 2.4, the data schema and endpoint URLs are required for the application to work properly.

2.5.1 SPARQL endpoint

One of the required application inputs is the endpoint against which the generated SPARQL queries can be run. Such endpoint is provided as an URL to the application. Examples of SPARQL compliant endpoints can be found on w3 site[3].

2.5.2 Data schema

The application also requires a data schema to be provided on the input in a form of a public accessible URL. The data schema can be created via various means. Based on the data structure, it could also be described manually. One of the automated ways is a pipeline dedicated to this purpose, its use is briefly described in subsection 5.2.2. The data schema is provided in a specific format[4] corresponding to the following example:

[3] `https://www.w3.org/wiki/SparqlEndpoints`

[4] `https://en.wikipedia.org/wiki/Turtle_(syntax)`

Figure 2.2: Data schema example

```
# Prefixes
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix nobel: <http://data.nobelprize.org/terms/> .
@prefix dbpedia: <http://dbpedia.org/property/> .

<urn:uuid:e6702622-087a-4fb7-bc3c-d3d08464a0ec> # 1
  rdf:object rdf:Literal; # 2
  rdf:predicate foaf:name; # 3
  rdf:subject <http://data.nobelprize.org/terms/Laureate/instance> . #4

<urn:uuid:c45bb448-459c-48e2-ad3e-bec71dc16813>
  rdf:object rdf:Literal;
  rdf:predicate dbpedia:dateOfBirth;
  rdf:subject <http://data.nobelprize.org/terms/Laureate/instance> .

<urn:uuid:e999257d-89b5-473a-8e0f-c14db7822eeb>
  rdf:object <http://data.nobelprize.org/terms/Laureate/instance>;
  rdf:predicate nobel:laureate;
  rdf:subject <http://data.nobelprize.org/terms/NobelPrize/instance> .

<urn:uuid:3d70e7e0-f406-4415-91cd-b1c91e6b11bd>
  rdf:object rdf:Literal;
  rdf:predicate nobel:category;
  rdf:subject <http://data.nobelprize.org/terms/NobelPrize/instance> .

<urn:uuid:0d7d4327-1d99-42f8-8ddc-faf2e32e8a32>
  rdf:object rdf:Literal;
  rdf:predicate nobel:year;
  rdf:subject <http://data.nobelprize.org/terms/NobelPrize/instance> .

# 5
<http://data.nobelprize.org/terms/NobelPrize/instance>
  a nobel:NobelPrize .
<http://data.nobelprize.org/terms/Laureate/instance> a nobel:Laureate .
```

Figure 2.2 can be described as follows:

- #Prefixes - The definition of prefixes used throughout the document.
- #1 - Unique identifier tied to the triple entry.
- #2, 3, 4 - Definition of the triple consisting of subject, predicate, object respectively
- #5 - Type definition. Defining object and subject types

Figure 2.3 shows how the data might be displayed in the final application.

Figure 2.3: Data schema example mockup

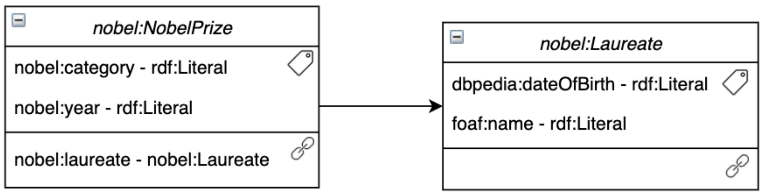

nobel:NobelPrize
nobel:category - rdf:Literal
nobel:year - rdf:Literal
nobel:laureate - nobel:Laureate
nobel:Laureate
dbpedia:dateOfBirth - rdf:Literal
foaf:name - rdf:Literal

Figure 2.3 illustrates how Figure 2.2 would look visualized. Every entity (#5 in Figure 2.2) from the data schema is represented as a node in the graph with edges representing relations between entities.

🏷 represents data properties, as such, these properties do not describe relationships between entities in the data schema and do not add any edges.

🔗 represents object properties, in this case there is a single object property for **nobel:NobelPrize**, which is **nobel:laureate**, with the target of **nobel:Laureate**. This relationship is depicted in the graph by an arrow between *nobel:NobelPrize* $\rightarrow$ *nobel:Laureate*.

A longer example with fully populated data can be found on GitHub[5].

2.6 Existing solutions

The problem area of this thesis is different from other various solutions as it aims to provide users with a tool to select data from already specified data schema.

Existing solutions providing visual tools for SPARQL query building are usually aimed at allowing users to explore the various relations and entities described by the selected vocabulary or data schema.

2.6.1 Comparison criteria

To allow for an objective comparison between existing solutions, a set of requirements has to be established. Taking section 2.3 into account, following criteria are established as a means of comparison between the works mentioned:

- C1 - Executable query

  The user is able to run the generated SPARQL query and retrieve the results in an arbitrary format.

- C2 - Configurable endpoint

  The user is able to execute the SPARQL query and choose an endpoint against which the queries are run.

- C3 - Property options

  The user is able to tweak the generated query, e.g. hiding results from the result set and marking properties as optional.

[5]`https://github.com/jaresan/simplod/blob/master/docs/example.ttl`

- C4 - Ability to query arbitrary properties

  The user is able to query any properties specified in the vocabularies provided by the application.

  For example, per the Figure 2.3 in an application that satisfies this criteria, the users would be able to query any property specified in the foaf vocabulary[6], e.g. **foaf:familyName**.

  If the application does not allow that, and for example it only allows to query for **foaf:name** as per Figure 2.3, the application would fail to meet this criteria.

- C5 - Constraints

  The user is able to impose value constraints on queried properties, resulting in a more specific SPARQL query.

- C6 - Documentation

  The user is able to read up on how to use the application and is guided in its use to understand the features implemented.

- C7 - Scenarios

  The user is able to follow examples provided by the authors to understand how to use the application.

- C8 - Demo

  The user is able to test the application without having to deploy it themselves.

2.6.2 Solution comparison

Table 2.1 describes comparison between existing solutions from this chapter based on criteria based on comparison criteria outlined in subsection 2.6.1:

Table 2.1: Comparison criteria

	RDF Explorer	VSB	This work
Executable query	x	✓	✓
Configurable endpoint	x	x	✓
Property options	x	✓	✓
Arbitrary properties	✓	x	o
Constraints	✓	✓	o
Documentation	✓	x	✓
Scenarios	✓	x	✓
Demo	✓	✓	✓

✓- Satisfies; x - Does not satisfy; o - Can be achieved using a workaround

[6] `http://xmlns.com/foaf/spec/`

RDF Explorer[7]

Solution created by authors Vargas, H., Buil-Aranda, C., Hogan, A., & López, C. in Vargas et al. [2019] might seem very similar at first sight, but the problem it solves is different from the goal of this thesis.

The application provides users with the possibility to query and explore RDF graphs via a web based visual tool. Users can access the Wikidata endpoint in a simple yet effective way and navigate through the dataset how they see fit, querying arbitrary data and imposing constraints on their properties. It does not allow for optional properties and other fine tuning of the generated queries.

Application also provides quality of life features for the user's convenience like an FAQ section, non trivial guided tour and multiple examples of usage.

Users are able to query the data set and navigate through the catalogue with ease, but there is no way to export their created queries. The hosted demo application is also statically targeted at the Wikidata endpoint, not allowing users to override this setting. The data query solution and navigating through the results does overlap with the scope of this thesis, but while appropriate for specific use cases, some of the missing features make this solution impractical for our target audience.

Figure 2.4: Lakes found in Chile

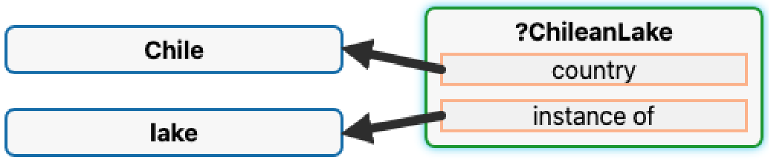
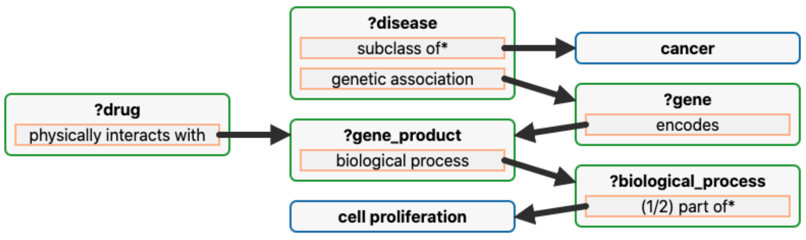


Figure 2.5: Drugs for cancer that target genes related to cell proliferation



[7]https://www.rdfexplorer.org/,http://ceur-ws.org/Vol-2456/paper60.pdf

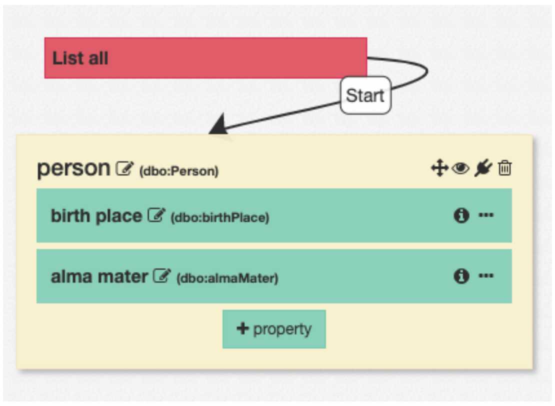
VSB: Visual SPARQL Builder[8]

Author Lukas Eipert aims to provide similar solution to the problem area as the one proposed in this work. Unfortunately the solution itself nor the page provide any information on how to properly use the application.

The application allows users to visually create SPARQL queries against the dbpedia[9] endpoint. Users are able to fine tune the queries by selecting properties as optional, hiding properties from the result set or negating their existence. The solution seems to be able to provide only properties from the domain of the chosen endpoint, limiting its use severely. There is also no option to run the application with a custom endpoint without cloning the repository and deploying the application as the one provided on the project website does not work.

Taking all its features and limitations into account, this solution seems more like a proof of concept for something that hasn't been finished entirely.

Figure 2.6: Example of querying people's alma maters and birth place



This work

As stated in chapter 1 the aim of this thesis is to provide the users with the possibility of querying data in a provided data set. Since the reader can be familiarized with the functionalities of this application and the goals of this work in chapter 2 we will describe what features our proposed solution is lacking and why, based on the comparison list specified in subsection 2.6.1.

Both of the features that are not fully implemented via the visualization tool **can be worked around** thanks to the requirement F3.3 in section 2.3, which allows the user to edit the SPARQL query directly, however they wish.

- C4 - Ability to query arbitrary properties

  Since the main use case of this application is for the users to be able to query for properties and entities in a specific data set they are provided, querying arbitrary properties from different dictionaries does not come into play.

  Users are able to create specific views to query for any of the properties found in the data set.

[8]`https://leipert.github.io/vsb/`
[9]`http://dbpedia.org/`

Querying for properties outside of the scope of the data set provided to the sure then does not make sense and does not provide any additional functionality to the user and hence is not implemented.

- C5 - Constraints

  While imposing value constraints on the properties queried is possible and might be beneficial to the user, it is outside of this scope of work. Developers familiarized with this work are welcome to submit changes to the application implementing this feature.

3. Design

This chapter describes the chosen technologies, proposed application architecture and the reasons behind these choices. It also proposes a simple view of possible solutions to the use cases described in section 2.4.

3.1 Type of application

The first decision to take is choosing the appropriate way to provide the solution to the user, in this case, the choice is between a desktop or a web application.

	Desktop application	Web application
Global distribution	x	✓
Portability	x	✓
Mobile access	x	✓
Easy update distribution	x	✓
Hassle-free user entry	x	✓
Offline capabilities	✓	x
No resource limitations	✓	x

Table 3.1: Application type comparison

Table 3.1 describes the key differences we are taking into account when choosing the appropriate solution for this work. For clarity, the comparison table displays just the few key differences which have the most impact on the final choice. There are many more differences between desktop and web application but the choice of technology is mainly dependant on the key ones chosen above.

3.1.1 Web application advantages:

- **Global distribution**

  With a web application, the solution can be run virtually anywhere in the world immediately after deployment, provided the end user has access to an internet connection.

- **Portability**

  Expanding on the previous point, the solution can be run on virtually any operating system without having the need to introduce changes to the code, provided that the users view it in a supported web browser.

- **Mobile access**

  If the web application is properly designed, the users are able to use it on their mobile devices as well without the need of installing anything. This also saves time for the developer as they don't have to develop two separate versions of the application.

- **Easy update distribution**

  Web applications benefit from being able to force the users to use the newest versions at all times. There is no need for version checks and update downloads, since the users have a single point of entry to the application provided by the deployer. This also makes bug fixes and improvements take immediate effect.

- **Hassle-free user entry**

  Since the application is not to be downloaded and installed, the users are able to start using it immediately after vising its point of entry, resulting in better user experience.

3.1.2 Desktop application advantages:

- **Offline capabilities**

  Desktop applications can usually be used without the need for internet connection.

- **No resource limitations**

  Desktop applications are not resource constrained by the web browser. They are able to use the local system's file system and all its peripherals. In addition to that, desktop applications do not suffer from reduced performance issues out of the box.

3.1.3 Conclusion

With the pros and cons described, requirements described in section 2.3 and the audience of this application in mind, web application was chosen as the best suitable option for the implementation part of this work.

3.2 Storage options

We discuss three main approaches to file handling in this section: local file system, general cloud based solutions and Solid Pods.

3.2.1 Local file system

As the type of application chosen is a web application, utilizing the user's computer file system is not convenient as direct file handling is not universally supported[1].

While all users are able to upload and download files when using a web application, implementing all file related use cases such way would be detrimental to the end user's experience. Local file system can be used as a fallback for situations where cloud solutions fail or are not accessible, but should not be considered as the main file handling approach.

[1]`https://developer.mozilla.org/en-US/docs/Web/API/File_and_Directory_Entries_API`

3.2.2 Cloud solutions

There is a plethora of cloud based storage options[2] for the developer to choose from when creating a web application. Utilizing such options provides the user with superior experience, as opposed to utilizing local file system, as there is no burden on the user to transfer and handle data by themselves. This in turn allows seamless transitions between machines, keeping the user experience consistent without their need for any additional setup.

3.2.3 Solid Pods

One such cloud based solution that can be utilized is a Solid Pod. It provides a user-driven decentralized way of storing data with the user being the owner of their data.

What is Solid?

Solid[3] (derived from "social linked data") is a set of open specifications which promote the creation of applications supporting user-ownership of data, its reusability, data/application decoupling, and interoperability.

The key points could be summarized as follows:

- Data ownership

  Users should be able to choose where to store their data and allowed to manage accessibility permissions to it.

- Vendor lock-in

  With Solid compliant applications, the users are able to avoid vendor lock-ins and switch between services without losing any data since they are their data's exclusive owners.

- Reusability & single source of truth

  Instead of repetitively creating data, users can just point third-party applications to the common resources shared amongst multiple applications.

What is a Solid Pod?

Solid Pod (personal online data store) is a place for users to store their data with whichever hosting service they prefer, using Solid.

Users can have a single Pod or multiple ones among which they might distribute different data. One Pod could be used for profile data, another for contact information, another for health information and so on. The user can then join an application by giving it permission to access the appropriate information in a specific pod. Users themselves define access to read and write permissions on the data in their associated Pods for each specific application separately, while retaining complete ownership and control of the data.

[2] `https://www.g2.com/categories/object-storage`

[3] `https://solidproject.org/faqs`

Utilizing available Solid browser libraries[4], applications are not required to have their own back-end layer implemented, and can directly access and handle the user provided data. The data is also decentralized and users can choose any type of hosting they want, provided it meets the Solid operability criteria.

Users can also opt for enterprise Solid server[5] which provides advanced security, higher performance, monitoring and other services.

3.2.4 Chosen solution

Taking the requirements of this work into account outlined in chapter 2 and considering the limitations of local file system storage, cloud storage solution is the clear pick.

While there are cloud storage solutions provided free of charge[6], Solid Pods were picked as the storage solution for this project due to their ease of use, complete permission control and data ownership for the users. Additionally utilizing Solid Pods results in zero storage fees for the application provider and makes implementing new features which operate with Solid Pod data easier thanks to the official Solid client library[7].

3.3 Language

The solution being a single page web application (SPA) restricts the language choices to JavaScript based languages and HTML with CSS used for its styling.

While vanilla JavaScript and HTML can be used, it is much easier to utilize one of the available frameworks which provide the developer with multitude of functionalities out of the box.

Such tools provide an already established and tested approach to the development process and allow the developer to utilize a higher level language for implementing more complex functionalities all the while the lower level details that might be prone to errors or more tedious to implement are taken care of by the framework. On top of that this way the project becomes easier to maintain and the development time is shortened as well.

One such example of shortening development time can be components for creating user friendly interfaces and design. There are numerous[8] libraries providing ready-to-use components with a unified design style across the board, allowing the developer to choose a single framework to use and effectively utilizing the same design principles across the whole application, making it design consistent.

3.3.1 Libraries & Frameworks

There are multiple different JavaScript frameworks to choose from when creating a single page application. Some of the examples include *Vue.js*[9], *Angular*[10],

[4]`https://github.com/solid/solid-auth-client`
[5]`https://inrupt.com/products/enterprise-solid-server/`
[6]`https://www.whizlabs.com/blog/best-free-cloud-storage/`
[7]`https://github.com/solid/solid-auth-client`
[8]`https://tinyurl.com/52jctn7d`
[9]`https://vuejs.org/`
[10]`https://angular.io/`

and *React*[11]. While all of these frameworks are different, majority of the web applications today could be implemented with either one.

In the case of this work the author decided to work with React/Redux[12] stack as React is the most popular [13] framework as of now and this stack is the one the author has the most experience with.

For creating visually pleasing user interfaces Ant Design[14] framework was selected due to its ease of use, sleek design and personal preference.

One of the crucial parts of this work is data visualization of selected queries and data sets. For this part there are multiple different JS libraries to choose from [15]. In this case the AntV[16] library was chosen. Reasons include ease of use, easily customizable data nodes, support for automatic layouts and its theme based on Ant Design which was selected for creating user interfaces in the course of this work.

React

React functions as a rendering layer for the application in which all of the HTML and dynamic functionality is implemented. With it, developers can easily create encapsulated components that manage their own state and compose them to make complex UIs. These components should be independent of other parts of the system and should only care about their inputs and how to display them and provide feedback back up the component chain.

Redux

Redux is a library that helps manage complex states of an application. It does so by having a single point of truth, so-called *store*, which should be managed only by special functions called *reducers*. This store - or its parts - is then provided to the React components in read-only mode.

To update the state with new values, the components have to dispatch special events called *actions* which get propagated further and are caught by the reducers. The reducers can then change the state based on the action's parameters.

To further separate concerns, so-called *sagas*[17] are often used as well. Sagas are usually used as a layer for handling side effects and structuring of more complex business logic, as well reacting to user interactions. Sagas can dispatch actions as well and this way change the global state via reducers.

3.4 Application architecture

The application should be divided into several layers where each layer should have its own separate responsibility. Basic separation can be a view layer with an application logic layer acting as a controller. Last layer would be concerned

[11]`https://reactjs.org/`
[12]`https://redux.js.org/`
[13]`https://tinyurl.com/nvhjwtcx`
[14]`https://ant.design/`
[15]`https://tinyurl.com/hrwknadb`
[16]`https://antv.vision/en`
[17]`https://redux-saga.js.org/`

with data storage and persistence, which would rely on Solid Pods, requiring the implementation to only use access to the layer hosted in the cloud, and not having to implement the storage layer directly.

3.4.1 View layer

Rendering of DOM elements and and all UI logic should be handled by React and it's supportive libraries (e.g. AntD, AntV) only. While the application's state updates effectively change what the user should see on their end, such changes should only be handled and wrapped by React with no direct connection between the model and UI elements. This in turn enables future developers to replace each layer separately allowing them to use different frameworks, should they so choose.

Apart from handling the rendering of components and data, React also provides user behaviour feedback upon which other layers (most importantly the saga layer) can react and decide what changes should be propagated to the application state or what additional events to initiate.

3.4.2 Application logic

When dealing with application logic, the main problems are handling user behaviour, persisting and propagating changes to the application's own state and handling side effects, such as fetching arbitrary data. To handle these problems two main layers are introduced: *reducers* for app state management and *sagas* for handling user interaction and side effects.

Reducers

Reducers are a layer responsible for changes made to the application's state. They are the only resource allowed to change the state of the application as a whole. While other layers can submit actions, specific subset of these actions gets caught in the reducers which in turn update the state. No other source of changes should be introduced for better readability and scaling.

Sagas

Sagas act as an intermediary layer between the user and application logic. Simple behaviours can be directly implemented in the UI components in the view layer, but when a user action is supposed to alter the application's state and cause data manipulation, extracting these interactions into a separate layer provides code clarity and ease of maintenance. UI elements then just fire actions with a payload and appropriate saga functions react to them and possibly fire additional actions to be handled by other sagas or reducers, resulting in state changes. This way the code can be split into logical pieces based on its functionality, separating the concerns.

Sagas are also used for handling asynchronous operations, e.g. data fetching and consequent operations.

3.4.3 Persistent data handling, Solid Pods

For user identification and authentication, Solid Pods are used. While the application should be usable by anonymous users as well, some the functionality would only be provided for authenticated users.

User's Solid Pods act as a cloud based data layer. Every user should be able to login on a different machine and navigate through their saved content with ease without the need to use any other storage methods.

3.4.4 SPARQL Editor

An integral part of the application is also the possibility of executing the generated SPARQL query against the specified endpoint. While this behaviour could be implemented directly as a part of this work, it is much easier to use an existing solution.

In this case, the author decided to choose the YASGUI[18] tool, due to its popularity and customizability. It allows for direct query handling via the code, listening on events, and handles showing the results in a meaningful format right out of the box.

3.4.5 Overview

The application should be divided into several layers where each layer should have its own separate responsibility. Such separation is readily available thanks to the utilization of above mentioned React & Redux libraries.

React framework supplies the role of the view layer, rendering necessary UI components based on provided parameters obtained from the application model, in our case the Redux store. Redux framework is then used for handling state management for the whole application, with sagas for handling side effects, data fetching and interconnecting these two layers.

Solid specific side effects, such as user authentication and subsequent data fetching from user's Pods, should be handled by third party libraries[19] dedicated specifically for such purpose.

An example of event handling and data flow through the architecture is provided in Figure 3.1.

An example of the flow could be as follows:

- *User clicks the login button*

  Corresponding react component gets notified via onClick handlers registered in the DOM and creates an action with specific payload.

- *Action catch*

  Since login will require side effects to be handled (e.g. calling the login API), this action uses the optional saga route. Corresponding saga will process the login action and utilize the Solid API to authenticate the user and wait for the result.

[18] `https://yasgui.triply.cc/`

[19] `https://github.com/solid/solid-auth-client`

- *Login result*

  After the login result is obtained (either successful or unsuccessful), the saga then creates a new action with appropriate payload that is sent to the reducer, or in case of more complex side effects can be further processed by other sagas.

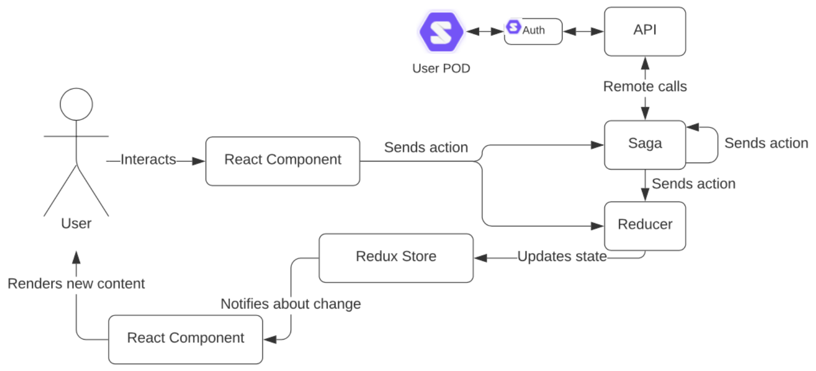
- *State change*

  The final action is then captured by the reducer which changes the actual application state values, in this case populating the application's session with the authenticated user's data. There are no additional effects created here. Reducers are the only sources of change to the state without exception.

- *View update*

  React components utilizing data from a subset of the application state get rendered with the new data where applicable.

Figure 3.1: Overview of the application life cycle

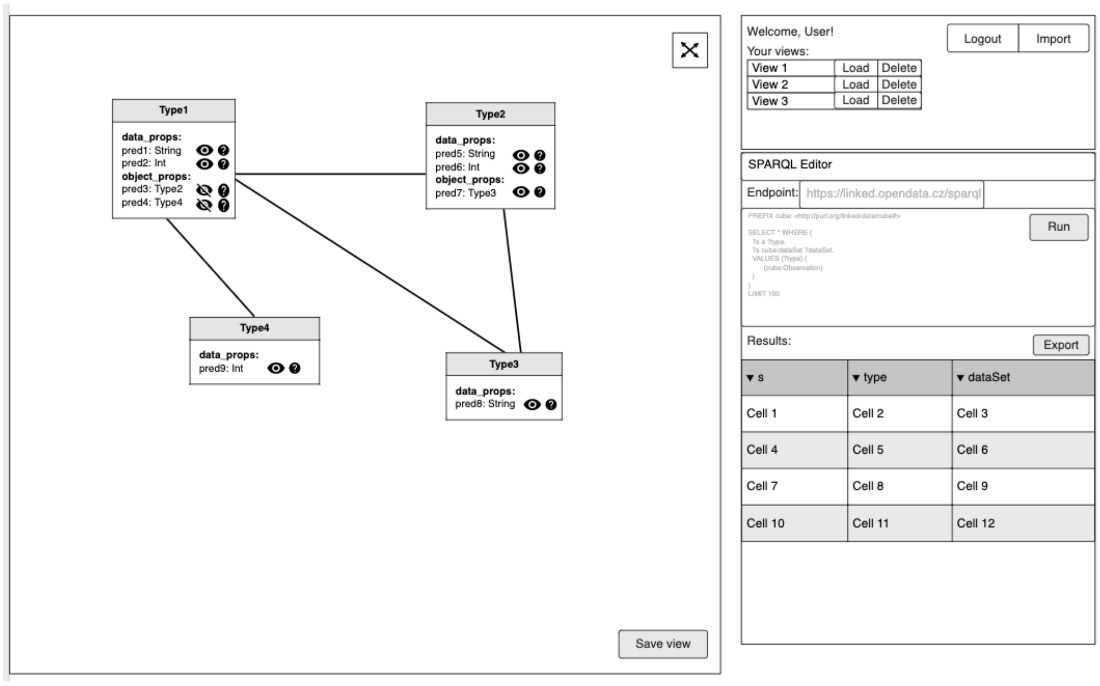


3.5 Mockups

This sections describes the approach taken to implementing use cases mentioned in section 2.4. The overview of the main screen is discussed first, followed by the description of separate parts of the application and their mockups. Each mockup is accompanied by short description of the visible elements and their purpose. Items marked with 🕒 in requirements do not have a corresponding mockup because they were introduced at a later stage of the implementation process.

3.5.1 Title page

Figure 3.2: Main screen



```
PREFIX cube: <http://purl.org/linked-data/cube#>

SELECT * WHERE {
  ?s a ?type.
  ?s cube:dataSet ?dataSet.
  VALUES (?type) {
      (cube:Observation)
  }
}
LIMIT 100
```

▾ s	▾ type	▾ dataSet
Cell 1	Cell 2	Cell 3
Cell 4	Cell 5	Cell 6
Cell 7	Cell 8	Cell 9
Cell 10	Cell 11	Cell 12

Figure 3.2 provides an overview of the main screen of the application. Considering the solution proposed is a web application without the need of transitions and additional screens, the chosen design is consisting of a single page separated into areas based on the application's business logic.

This way the user can have overview of everything that is happening on the screen at once without the need of switching between screens. The proposed design also allows displaying the main interactive area in a separate window to allow the user to navigate in the data set more easily if needed.

3.5.2 User information

Figure 3.3: User information - Anonymous

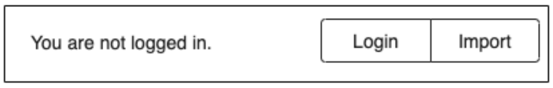


Figure 3.4: User information - Logged in

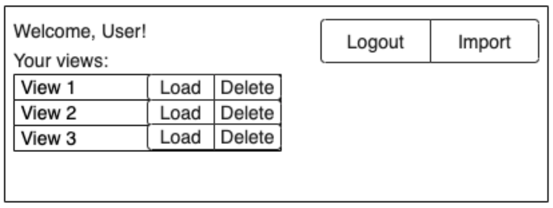


Figure 3.3 and Figure 3.4 depict the part of the application with user information and view management for anonymous and logged in users respectively.

Anonymous users are limited to log in and view import functionalities.

Logged in users are additionally able to manage the views they have saved with the application.

Screen functionalities can be described as follows:

- **Login** - Allows the user to log in to a Solid Pod
- **Logout** - Allows the user to log out of a Solid Pod
- **Import** - Allows the user to display a view created by this application by providing a URL at which the view is saved
- **”Your views” list** - List of views the user has previously saved in their Solid Pod with the application
- **Load** - Functionally the same as ”Import”, except the view is directly loaded from the user’s Solid Pod
- **Delete** - Deletes specific view from the user’s Solid Pod

This screen fulfills requirements introduced in F4 - User data storage.

3.5.3 Data area

Figure 3.5: Main area

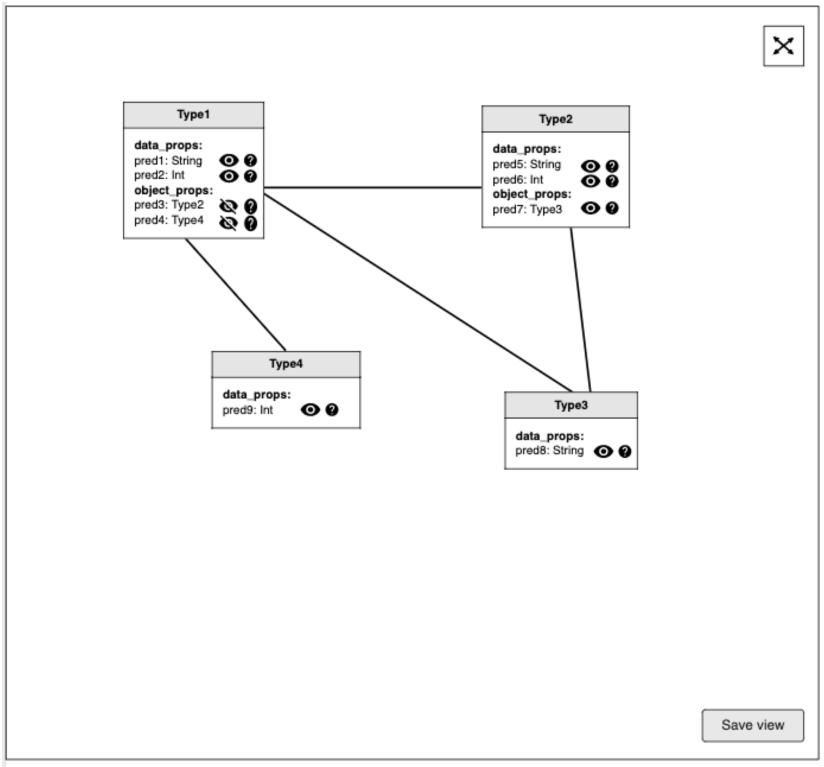


Figure 3.5 depicts the main area of the application where the user can view the data schema provided in a visual format.

Every entity from the schema is represented as a node with edges signifying relations between these entities.

Users are able to open this area in a new window, providing them with an option to view the schema on a larger scale. Additionally, logged in users are able to save their current selection as a view, effectively adding it to their catalogue shown in Figure 3.4.

Data entities

Figure 3.6: Data entity

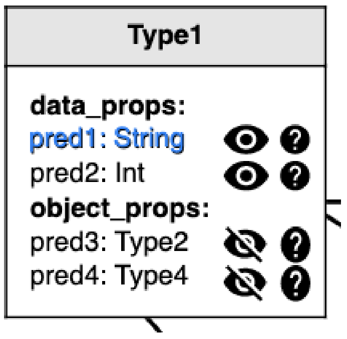


Figure 3.6 shows an element of the data area that the user can interact with. Every entity displays the list of their data and object properties. The user is able to interact with each property of the entity via the following means:

- Clicking the property

  Marks the property as selected, requesting the entity and the specific property in the result set. Selected properties are marked with a different color than not selected ones.

- 👁

  Toggles between displaying the property value in the query result or omitting it. Entities queried with properties hidden this way are still required to have such a property, the result would just omit it.

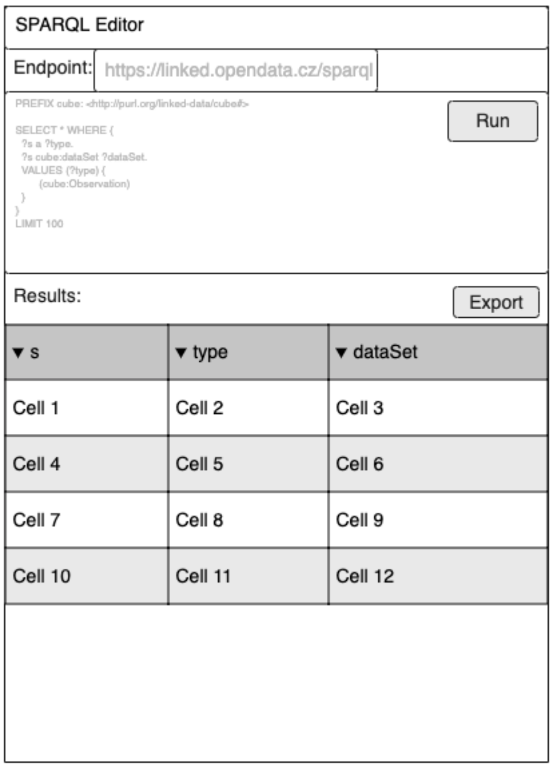
- ?

  Toggles the property as optional, changing the generated SPARQL query. Queried entities shown in the result set do not need to contain relationships marked as optional.

This screen fulfills requirements introduced in F3 - Generate SPARQL SELECT query.

3.5.4 Query editor

Figure 3.7: Query editor



```
PREFIX cube: <http://purl.org/linked-data/cube#>

SELECT * WHERE {
  ?s a ?type.
  ?s cube:dataSet ?dataSet.
  VALUES (?type) {
      (cube:Observation)
  }
}
LIMIT 100
```

▾ s	▾ type	▾ dataSet
Cell 1	Cell 2	Cell 3
Cell 4	Cell 5	Cell 6
Cell 7	Cell 8	Cell 9
Cell 10	Cell 11	Cell 12

Figure 3.7 depicts the SPARQL query editor. Here the user can see the generated query from the selection created in Figure 3.5. Interactive elements on this screen can be described as follows:

- Endpoint field

  Allows the user to change the endpoint that should be used to run the generated generated SPARQL query.

- ’Run”

  Runs the generated SPARQL query against the specified endpoint.

- ”Export”

  Exports the received result from running the generated SPARQL query against the endpoint in a CSV format to be downloaded by the user.

- Result table

  Apart from exporting the result set, the users are able to see the data directly in a table form as well.

This screen fulfills requirements introduced in F5 - Run SPARQL query.

4. Implementation

In the first part, this chapter describes the changes in implementation introduced as opposed to the proposed design in section 3.4.

The second part mentions interesting implementation tasks and the approach that was taken to solve them.

The chapter concludes with a description of improvements that could be added to the application to enhance the user's experience and allow for more granular control over the results.

4.1 Limited mobile experience

While the advantages of a web application discussed in section 3.1 mention mobile access as well, during the course of implementation of this work, this feature was not implemented in full for the following reasons:

- Out of the box support

  Chosen AntV[1] used for graph handling does not out of the box support behaviors same as on desktop. For example dragging nodes[2] is implemented on desktop by default, but for mobile, the implementation would have to be custom.

- Lack of added value

  Navigation around the canvas and other UI elements was deemed more user friendly with the desktop experience due to a larger screen and intuitive mouse controls.

- Development time

  Implementing custom canvas event handling that is intuitive for touch events is not trivial. While mobile touch events in a way simulate the behaviour of the mouse, the experiences differ.

  Another issue to tackle is styling for mobile devices which are vastly different, adding more the the development overhead.

However, the users are still able to access the application via their mobile devices and use most of the functionality in a similar fashion to the desktop experience.

For example the entity list view has no limited interactions available on mobile devices. Some issues can be found with interaction with the graph, mainly dragging of nodes as mentioned above. Selecting properties, clicking the edges and moving around the graph is not impacted on mobile devices.

[1]`https://g6.antv.vision/en`

[2]`https://g6.antv.vision/en/docs/manual/middle/states/defaultBehavior#drag-node`

4.2 State handling with Ramda

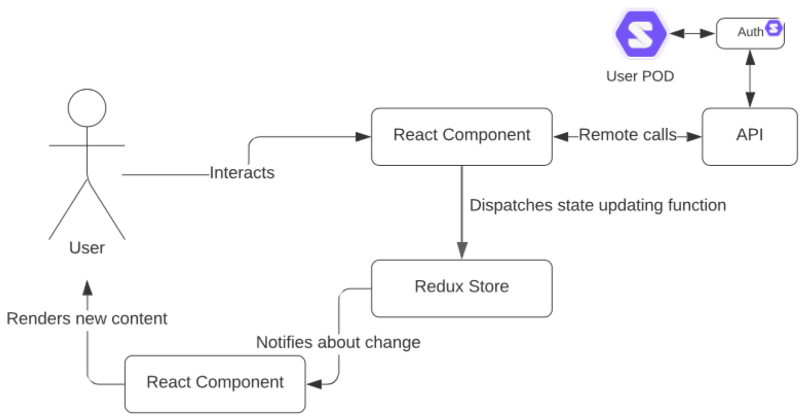
One notable change is removing Redux Sagas altogether and utilizing Ramda lenses[3] instead. One major downside of utilizing Sagas is having no direct understanding and control over the flow of the application due to the nature of the listener registrations and action propagation. Utilizing Ramda, the effects are directly observable and can be reasoned about, which in turn allows for more understandable developer experience.

Ramda lenses are also used for handling state changes in the place of reducers. Utilizing Ramda's functions has the added benefit of always returning a new copy of the object where necessary, not requiring the developer to keep track of where to use the spread operator to prevent changing the state without returning a new reference and in turn preventing the UI to update. Another such library that could be used is immutablejs[4]. The author has picked Ramda over the other solutions mainly to notable experience with it and its ease of use and maintainability.

What this ultimately means in the code is that instead of dispatching actions, state updating functions are dispatched instead. This is provided by a custom dispatch method which in turn applies the provided state updating functions and propagates the changes to the redux store.

As opposed to the picture in Figure 3.1, the newly implemented architecture looks as follows:

Figure 4.1: Overview of the implemented application architecture



4.3 SPARQL Proxy

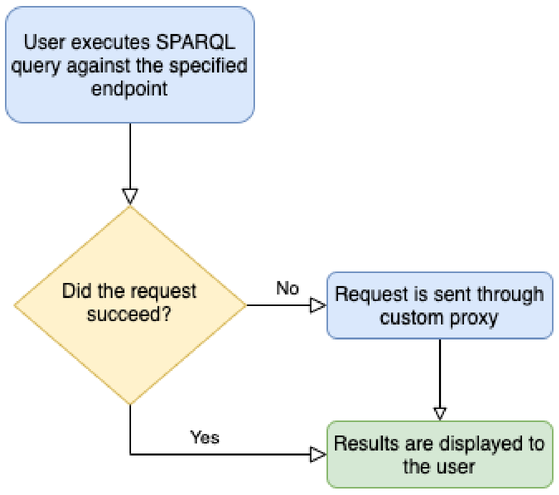
Due to the nature of this work, requests to remote sources that are not a part of this application are inevitable. This mainly includes the need for querying third-party SPARQL endpoints, which can come with obstacles, namely regarding CORS[5] and Cross-origin policy[6].

[3]https://ramdajs.com/docs/#lens
[4]https://github.com/immutable-js/immutable-js
[5]https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS
[6]https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

As mentioned in subsection 3.4.4, YASGUI[7] is used as the tool for handling SPARQL queries and their execution. This tool also allows the developer to set up a proxy which can be used as a fallback scenario, should the default requests fail. To alleviate the aforementioned problems, this work is shipped with its own implementation of such proxy, that is used to propagate the created SPARQL requests further to the endpoints specified.

The proxy repository is hosted on GitHub[8] and with the shipment of this work, a running instance can be found on heroku[9]. The flow of the SPARQL requests can be described by the following figure:

Figure 4.2: SPARQL proxy


User executes SPARQL query against the specified endpoint
Did the request succeed?
No
Request is sent through custom proxy
Yes
Results are displayed to the user

4.4 Interesting implementation tasks

This section describes interesting tasks that arose during the development process and the approach that was taken to resolve them.

4.4.1 SPARQL generation

One of the most important tasks of the development process was proper SPARQL query generation based on the selection provided by the user.

This subsection provides overview of the approach taken to this problem and some interesting caveats. The whole algorithm for generating the SPARQL query is defined in `src/@@data/parseQuery.js`.

Object properties vs data properties

For the query to be parsed properly, an important distinction has to be made between object properties and data properties. For object properties (properties whose object is a class[10]), the query also needs to contain a definition of the

[7] `https://yasgui.triply.cc/`
[8] `https://github.com/jaresan/sparql-proxy/`
[9] `https://simplod.herokuapp.com/`
[10] `https://www.w3.org/TR/rdf-schema/#ch_class`

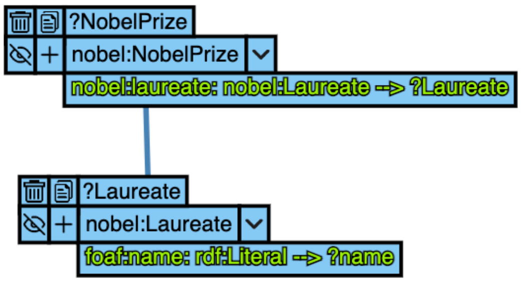
object type via the `rdf:type`[11] predicate. For data properties this is not needed, since they are not an instance of a class.

To properly and completely correctly distinguish between object and data properties, it would be necessary to query the respective vocabularies and parse the information for every object type. A shortcut approach however can be implemented by treating every entity that is not listed as a subject in the data schema as a data type. Every property with an object that is not used as a subject anywhere in the data schema is then considered a data property.

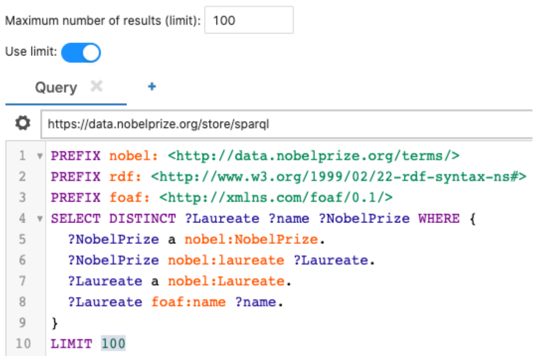
Default query creation

Users are able to influence what the generated query looks like via multiple interactions available in the graph and the list view. The default example is clicking the properties in the graph, which in turn adds them to the query as follows:

Figure 4.3: Default use case


?NobelPrize
nobel:NobelPrize
nobel:laureate: nobel:Laureate --> ?Laureate
?Laureate
nobel:Laureate
foaf:name: rdf:Literal --> ?name

(a) Graph view



Maximum number of results (limit): 100

Use limit:

Query

https://data.nobelprize.org/store/sparql

```
PREFIX nobel: <http://data.nobelprize.org/terms/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?Laureate ?name ?NobelPrize WHERE {
  ?NobelPrize a nobel:NobelPrize.
  ?NobelPrize nobel:laureate ?Laureate.
  ?Laureate a nobel:Laureate.
  ?Laureate foaf:name ?name.
}
LIMIT 100
```

(b) Corresponding query

In the example depicted in Figure 4.3, the user is simply requesting the Nobel prizes and the corresponding laureates and their names. Since all the properties represent a triple of subject, predicate, object, all of this data has to somehow be represented in the resulting query. As mentioned above, data properties do not have their object types specified in the query. The subject and object types will then be added as follows:

```
?NobelPrize a nobel:NobelPrize.
?Laureate a nobel:Laureate.
```

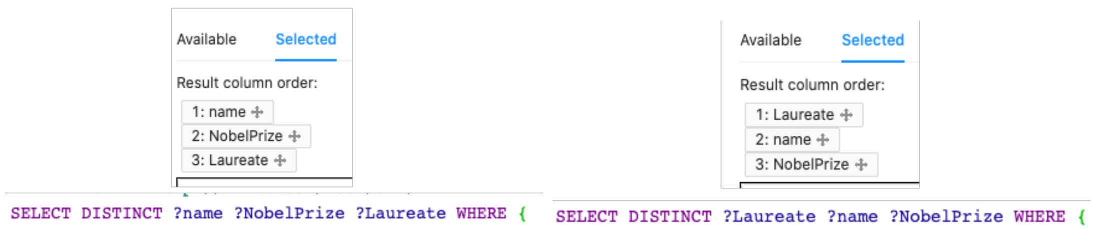
Finally the predicates linking the subjects and objects are simply represented by `?subject ?pred ?object`, in this case:

```
?NobelPrize nobel:laureate ?Laureate. # object property
?Laureate foaf:name ?name. # data property
```

The users are also able to change the selection order by reordering the items in the ”Selected” tab in list view. This interaction just changes the order of the variables in the SELECT part of the query as follows:

[11] `https://www.w3.org/TR/rdf-schema/#ch_type`

Figure 4.4: Selection order


Available Selected

Result column order:

1: name

2: NobelPrize

3: Laureate

```
SELECT DISTINCT ?name ?NobelPrize ?Laureate WHERE {
```

Available Selected

Result column order:

1: Laureate

2: name

3: NobelPrize

```
SELECT DISTINCT ?Laureate ?name ?NobelPrize WHERE {
```

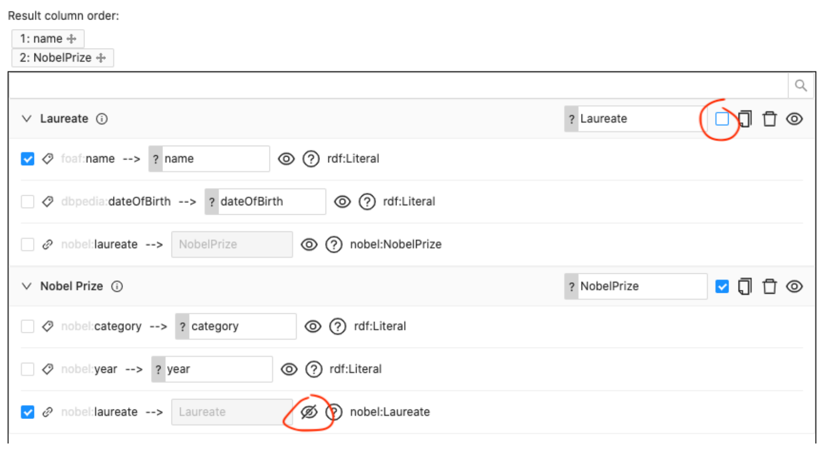
(a) Name first (b) Laureate first

Finally, one of the features provided to the user, as per section 2.3.1, is the option to hide a variable from the result set if they wished to query data satisfying certain relationships, but would not care about the details.

For data properties, this is just a simple removal of `?variable` from the SELECT part of the query, but for object properties, the behaviour is more complicated. Since object properties directly target other entities that include their own controls, the behaviour and representation has to be synchronized. Properties targeting the same entity have to be synchronized as well, because if the user removes an entity from the result set via one property, they arguably would not want other properties displaying the same information.

In the application, this behaviour is then implemented as a propagation of hide/show actions throughout all the properties targeting the same entity, and the entity itself. This means removing a property from the result set marks all properties with the same target as removed and also marks the entity itself as removed. Vice versa, removing an entity from the result set marks all properties targeting it as removed as well, as illustrated by the following picture. The fields marked in red are always synchronized:

Figure 4.5: Selection and hiding always have the same state


Result column order:

1: name

2: NobelPrize

Laureate — ? Laureate

☐ foaf:name --> ? name — rdf:Literal ✓

☐ dbpedia:dateOfBirth --> ? dateOfBirth — rdf:Literal

☐ nobel:laureate --> NobelPrize — nobel:NobelPrize

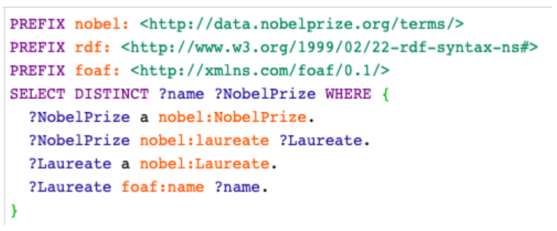
Nobel Prize — ? NobelPrize ✓

☐ nobel:category --> ? category — rdf:Literal

☐ nobel:year --> ? year — rdf:Literal

✓ nobel:laureate --> Laureate — nobel:Laureate

Figure 4.6: Laureate removed from the result set



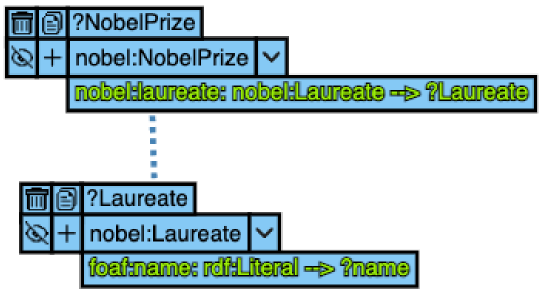
```
PREFIX nobel: <http://data.nobelprize.org/terms/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?name ?NobelPrize WHERE {
  ?NobelPrize a nobel:NobelPrize.
  ?NobelPrize nobel:laureate ?Laureate.
  ?Laureate a nobel:Laureate.
  ?Laureate foaf:name ?name.
}
```

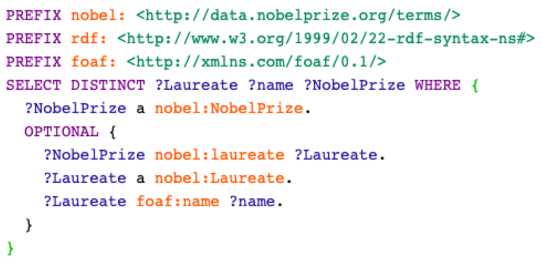
Optional clauses

Using a visualization tool naturally limits the user's influence on adjusting the query as opposed to using the SPARQL editor directly. Using the visualization tool, there can be different ways how to implement user's interactions and what role these interactions play in creating the final query. One of the interactions that can be interpreted in different ways is marking items as optional.

For the use of this application, the assumption is that the vast majority of users would like to request data that are linked. Optional data properties do not create links and are therefore of no concern. Object properties, however, create links between entities that are to be queried and could potentially introduce requesting data that the user might not be interested in. Take the following example:

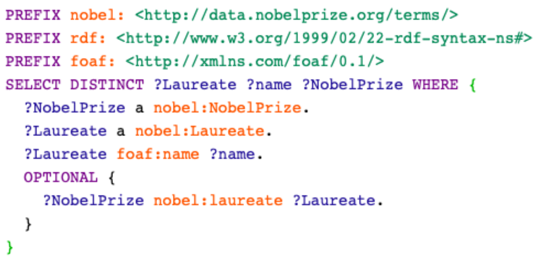
Figure 4.7: Optional link


?NobelPrize
nobel:NobelPrize
nobel:laureate: nobel:Laureate --> ?Laureate
?Laureate
nobel:Laureate
foaf:name: rdf:Literal --> ?name



```
PREFIX nobel: <http://data.nobelprize.org/terms/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?Laureate ?name ?NobelPrize WHERE {
  ?NobelPrize a nobel:NobelPrize.
  OPTIONAL {
    ?NobelPrize nobel:laureate ?Laureate.
    ?Laureate a nobel:Laureate.
    ?Laureate foaf:name ?name.
  }
}
```

(a) Query n.1



```
PREFIX nobel: <http://data.nobelprize.org/terms/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?Laureate ?name ?NobelPrize WHERE {
  ?NobelPrize a nobel:NobelPrize.
  ?Laureate a nobel:Laureate.
  ?Laureate foaf:name ?name.
  OPTIONAL {
    ?NobelPrize nobel:laureate ?Laureate.
  }
}
```

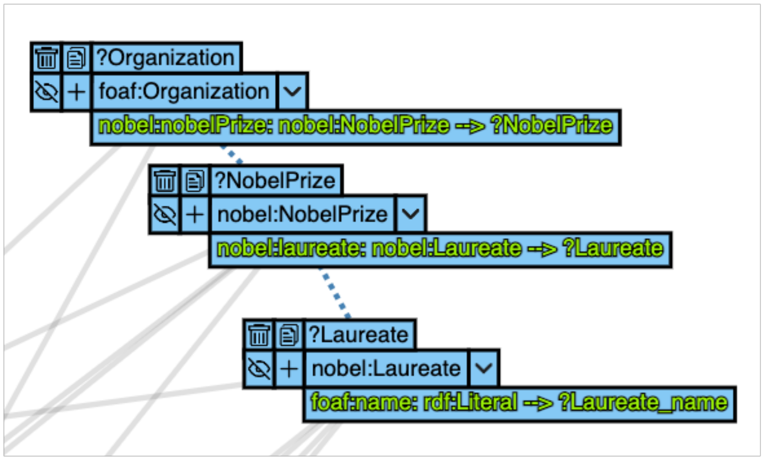
(b) Query n.2

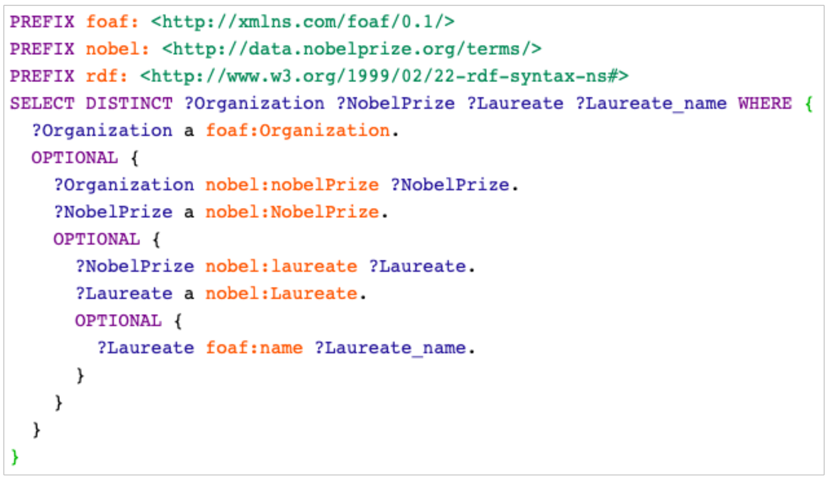
Figure 4.7 depicts an example where the user would like to query for Nobel prizes and their Laureate's names. The relation between Laureate and Nobel prizes is marked as optional here, which in turn can be interpreted two different ways, depicted by Query n.1 and Query n.2.

As mentioned before, we assume the user requests only resources that are linked, which in turn would result in Query n.1. The way this behaviour is implemented in the application is as follows: Every entity requested through an optional edge is, along with all its requested properties, defined in the same optional clause.

This works recursively, creating nested optional clauses, as depicted in the following example:

Figure 4.8: Nested optionals


?Organization
foaf:Organization
nobel:nobelPrize: nobel:NobelPrize --> ?NobelPrize
?NobelPrize
nobel:NobelPrize
nobel:laureate: nobel:Laureate --> ?Laureate
?Laureate
nobel:Laureate
foaf:name: rdf:Literal --> ?Laureate_name



```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX nobel: <http://data.nobelprize.org/terms/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT DISTINCT ?Organization ?NobelPrize ?Laureate ?Laureate_name WHERE {
  ?Organization a foaf:Organization.
  OPTIONAL {
    ?Organization nobel:nobelPrize ?NobelPrize.
    ?NobelPrize a nobel:NobelPrize.
    OPTIONAL {
      ?NobelPrize nobel:laureate ?Laureate.
      ?Laureate a nobel:Laureate.
      OPTIONAL {
        ?Laureate foaf:name ?Laureate_name.
      }
    }
  }
}
```

Every relationship edge is marked as optional, providing another level of nesting for optional clauses. This way only properly linked data is shown to the user without the concern of requesting arbitrary data that is not linked.

4.4.2 Cartesian product detection

One of the requirements mentioned in section 2.3 is to warn the user about the query potentially requesting a cartesian product.

In short, the check whether the query might result in fetching a cartesian product is based on the connectivity of the graph. If the graph can be considered a single connected component[12] there should be no concern of querying for a cartesian product, due to how the SPARQL query is generated, as pointed out in subsection 4.4.1.

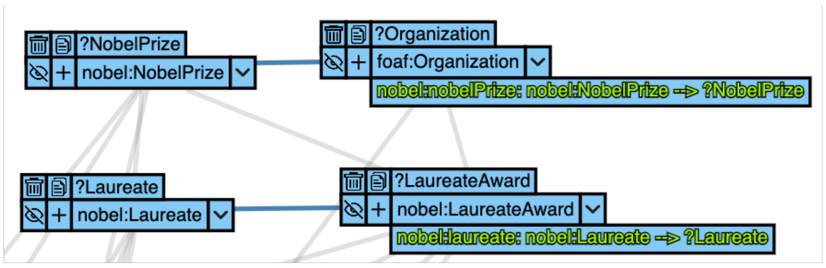
The functionality can be illustrated on the following examples:

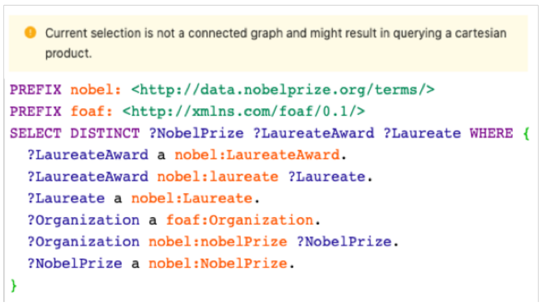
[12]`https://en.wikipedia.org/wiki/Component_(graph_theory)`

- Disconnected graph

  Creating a selection that contains multiple connected components would in turn possibly query a cartesian product:

Figure 4.9: Two connected components




Current selection is not a connected graph and might result in querying a cartesian product.

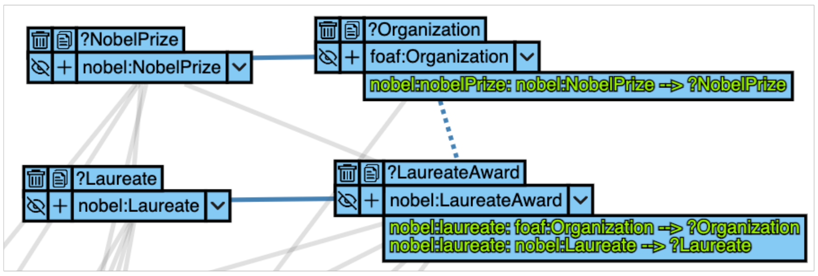
```
PREFIX nobel: <http://data.nobelprize.org/terms/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?NobelPrize ?LaureateAward ?Laureate WHERE {
  ?LaureateAward a nobel:LaureateAward.
  ?LaureateAward nobel:laureate ?Laureate.
  ?Laureate a nobel:Laureate.
  ?Organization a foaf:Organization.
  ?Organization nobel:nobelPrize ?NobelPrize.
  ?NobelPrize a nobel:NobelPrize.
}
```

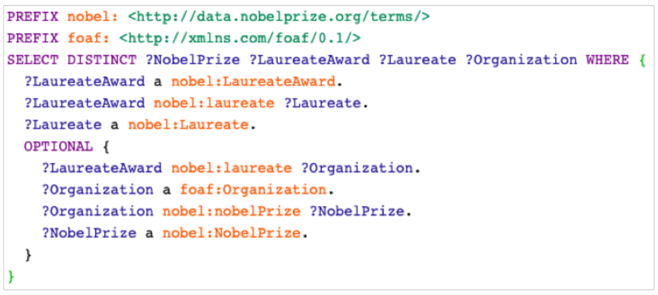
- Connected by an edge

  Adding a single edge to the previous example, even if it is optional, connects the two connected components, removing the possibility of querying for a cartesian product.

  Trivially, if the edge is not considered optional, the same holds true.

Figure 4.10: Two components connected by an optional edge


?NobelPrize
nobel:NobelPrize
?Organization
foaf:Organization
nobel:nobelPrize: nobel:NobelPrize --> ?NobelPrize
?Laureate
nobel:Laureate
?LaureateAward
nobel:LaureateAward
nobel:laureate: foaf:Organization --> ?Organization
nobel:laureate: nobel:Laureate --> ?Laureate



```
PREFIX nobel: <http://data.nobelprize.org/terms/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?NobelPrize ?LaureateAward ?Laureate ?Organization WHERE {
  ?LaureateAward a nobel:LaureateAward.
  ?LaureateAward nobel:laureate ?Laureate.
  ?Laureate a nobel:Laureate.
  OPTIONAL {
    ?LaureateAward nobel:laureate ?Organization.
    ?Organization a foaf:Organization.
    ?Organization nobel:nobelPrize ?NobelPrize.
    ?NobelPrize a nobel:NobelPrize.
  }
}
```

- Multiple optional edges

An interesting example for cartesian product detection is nested optionality. As mentioned in subsection 4.4.1, the optional clauses are generated recursively so that data that is not properly linked is not queried.

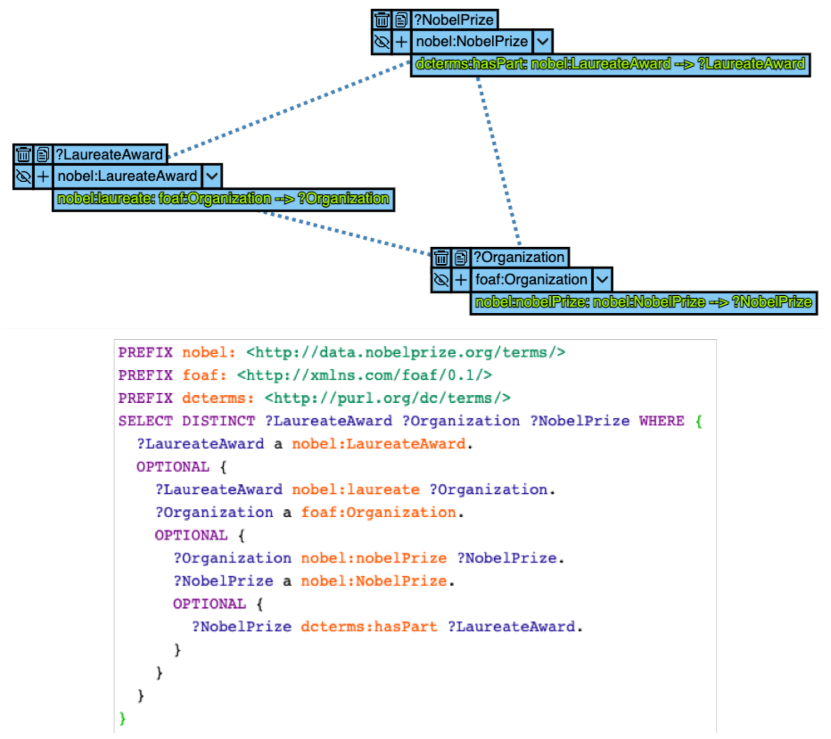
Because of this, a path of optional edges cannot introduce a cartesian product into the query, because all of the subsequent optionals are nested into one another. This way, the example depicted in Figure 4.8 cannot result in querying for a cartesian product and hence the cartesian product warning wouldn't be shown.

4.4.3 Optionality cycles

While not a common scenario, the application should be able to handle a setup where the selection graph contains a cycle. For non-optional properties this is trivial, the query does in no way reflect that and would execute just fine. For optional properties however, this becomes an issues, since they are resolved recursively as pointed out in subsection 4.4.1.

The chosen approach to solve this problem is simple, the algorithm responsible for expanding node after node does not follow edges that are pointing to already introduced nodes. This way the cycle is cut and the optional clause recursion can work properly. This can be seen in the following example:

Figure 4.11: Optionality cycle


?NobelPrize
nobel:NobelPrize
dcterms:hasPart: nobel:LaureateAward --> ?LaureateAward
?LaureateAward
nobel:LaureateAward
nobel:laureate: foaf:Organization --> ?Organization
?Organization
foaf:Organization
nobel:nobelPrize: nobel:NobelPrize --> ?NobelPrize

```
PREFIX nobel: <http://data.nobelprize.org/terms/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dcterms: <http://purl.org/dc/terms/>
SELECT DISTINCT ?LaureateAward ?Organization ?NobelPrize WHERE {
  ?LaureateAward a nobel:LaureateAward.
  OPTIONAL {
    ?LaureateAward nobel:laureate ?Organization.
    ?Organization a foaf:Organization.
    OPTIONAL {
      ?Organization nobel:nobelPrize ?NobelPrize.
      ?NobelPrize a nobel:NobelPrize.
      OPTIONAL {
        ?NobelPrize dcterms:hasPart ?LaureateAward.
      }
    }
  }
}
```

The node chosen as the first one to expand is based on the user's order of interactions with the application.

4.5 Room for improvement

This sections describes functionalities that would be considered beneficial to the end user but have not been implemented as a part of this work for various reasons. The first part discusses different features that could be added to the graph tool, following up with improvements that could be made to support additional distinction in the generated SPARQL query.

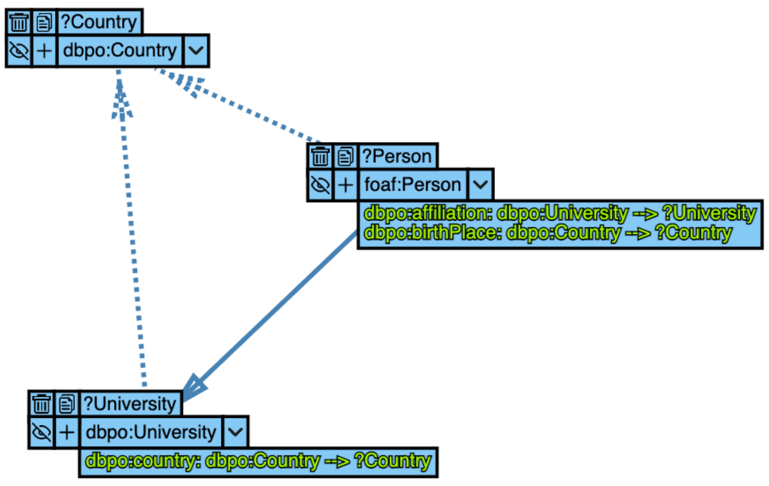
4.5.1 Graph tool

The utilized AntV graphing library comes with a variety of features out of the box, including, but not limited to, zoom, node dragging, panning and automatic node layout. Improvements that could be built on top of this library include some of the following:

Edge direction

The RDF triples[13] representing the data schemas used in this application effectively describe a graph with directed edges. This works implements edges without any directional information, not allowing the users to see the directions of the relations in the graph. While AntV supports adding directional edges, the simple example tried has been deemed unsatisfactory and further implementation has been put on hold.

Figure 4.12: AntV edge direction


?Country
dbpo:Country
?Person
foaf:Person
dbpo:affiliation: dbpo:University --> ?University
dbpo:birthPlace: dbpo:Country --> ?Country
?University
dbpo:University
dbpo:country: dbpo:Country --> ?Country

Automatic node layout

It would be ideal if the users could select different options how to distribute the nodes in the graph when the data schema is loaded. While AntV supports different layout options out of the box, due to the variety of data schemas that can be presented to the application, there is not a one-size-fits all solution. Properly

[13] https://en.wikipedia.org/wiki/Semantic_triple

distributing nodes in the graph to show the data in the best possible way is a non-trivial task and one that requires a significant amount of effort to implement. If the future developer wants to use an algorithm that would take the graph data as an input and would return the positions for each node, adding it to the application should be a simple task.

User defined node layout

Considering that the positions of graph nodes is information that is saved as a part of the project, the curators are able to edit data schemas in a way that would future users to vied the data in a more compact way.

It would be beneficial to allow these curators to align and distribute items based on certain criteria.

4.5.2 Query improvements

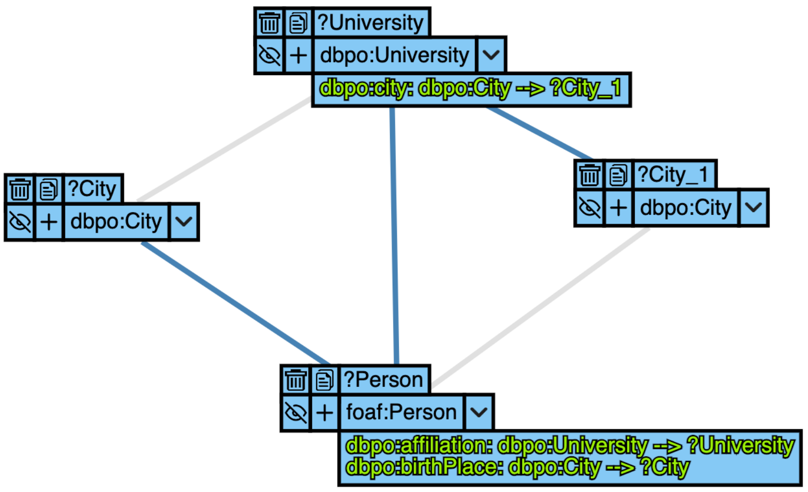
By the nature of the application being a visualization tool, the query customization will necessarily be limited as opposed to the direct manipulation of the query. Some of the following options would improve users experience with generating the query.

While not implemented directly through user's interaction with the graph or the list view, **all of the features described in this subsection can be replaced with directly editing the SPARQL query** thanks to the requirement F3.3 in Section 2.3.1.

Distinction between two entities of the same type

Copying a node in the graph via 🗐 creates another node with the same type and allows for querying for it as a new target of appropriate properties. Consider the following example:

Figure 4.13: Same type distinction


?University
dbpo:University
dbpo:city: dbpo:City --> ?City_1
?City
dbpo:City
?City_1
dbpo:City
?Person
foaf:Person
dbpo:affiliation: dbpo:University --> ?University
dbpo:birthPlace: dbpo:City --> ?City

The example pictured in Figure 4.13 queries for a person, their university and cities for both of the entities. The user might expect the cities in this case to represent strictly different values, due to them being represented by two different nodes in the graph, but that is not the case. The generated query would look as follows:

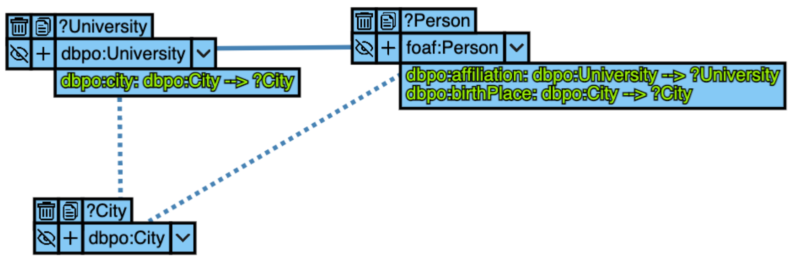
```
SELECT DISTINCT ?University ?City_1 ?Person ?City WHERE {
  ?Person a foaf:Person.
  ?Person dbpo:affiliation ?University.
  ?University a dbpo:University.
  ?University dbpo:city ?City_1.
  ?City_1 a dbpo:City.
  ?Person dbpo:birthPlace ?City.
  ?City a dbpo:City.
}
```

Should the user not copy the node via the 🗐 icon, they would query for a person and their university only if they studied in the same city they were born in.

More granular optionality handling

The way this work is implemented, every optional property has its own separate clause in the final generated SPARQL query. While such an approach is not invalid, sometimes the users could benefit from being able to group optional clauses into one. To better illustrate the feature, let's consider the following picture:

Figure 4.14: Double optional target



?University
dbpo:University
dbpo:city: dbpo:City --> ?City
?Person
foaf:Person
dbpo:affiliation: dbpo:University --> ?University
dbpo:birthPlace: dbpo:City --> ?City
?City
dbpo:City

Figure 4.14 depicts a selection of two optional relations and one required relation. From the graph, we can see, that the data queried is a person, their university, and for each of these entities, their corresponding city. The generated query would be as follows:

```
SELECT DISTINCT ?Person ?City ?University WHERE {
  ?Person a foaf:Person.
  ?Person dbpo:affiliation ?University.
  ?University a dbpo:University.
  OPTIONAL {
    ?University dbpo:city ?City.
    ?City a dbpo:City.
  }
  OPTIONAL {
    ?Person dbpo:birthPlace ?City.
    ?City a dbpo:City.
  }
}
```

In this case, since the properties are considered optional, the **?City** variable would be populated with either the city of the university, or the city of birth of the queried person. The user however has no means to distinguish between these two and might want to create a selection that would return a city only if the person was born and attended university in the same city. In that case, the desired query would look as follows:

```
SELECT DISTINCT ?Person ?City ?University WHERE {
  ?Person a foaf:Person.
  ?Person dbpo:affiliation ?University.
  ?University a dbpo:University.
  OPTIONAL {
    ?University dbpo:city ?City.
    ?Person dbpo:birthPlace ?City.
    ?City a dbpo:City.
  }
}
```

The application does not support such a way to group optional queries in this regard. The user can still directly manipulate the SPARQL query however to achieve this result.

Property value constraining

One of the comparison criteria described in subsection 2.6.1 is the ability to constrain the values of properties to be selected. Such a feature would allow the users to create more specific queries, searching for data satisfying a set of criteria imposed by the user.

5. Documentation

This chapter provides additional information about the application to users, administrators and programmers who might look to improve the application or would want to understand its inner workings better.

This chapter can also be found on GitHub[1].

5.1 User documentation

This section describes how the user can connect to the application, utilize their Solid Pod and furthermore shows basic manipulation of the application data and view creation.

The reader is encouraged to navigate through the user guide[2] should they have any problems following specific steps setting up the Solid environment.

5.1.1 Solid Pod setup

This subsection describes the requirements necessary to enable the application to persist its data by utilizing the user's Solid Pod. While the application can be used without Solid Pods as illustrated in section 2.4, no data and view management can be established, therefore severely limiting the functionality of the application. The user is encouraged to read through the documentation and follow all the steps outlined to familiarize themselves with the application and set up the environment properly.

Preliminaries

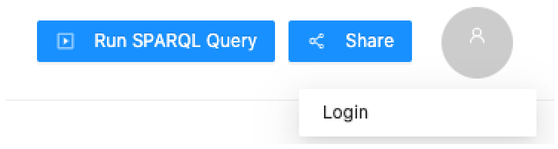
The following sections are written for Solid Pods hosted at Inrupt[3]. This application shall be usable with any Solid Pod providing service but the details regarding its use might differ.

We assume the reader has their own Solid Pod set up, if not, they are able to create a new one for free on the registration page[4].

Logging in with Solid Pod

To set up all permissions for the application, the user has to sign in to their Solid Pod first via the application's avatar button in the top right corner.

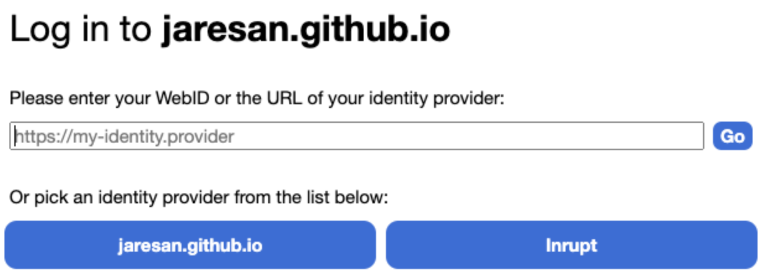
Figure 5.1: Avatar menu



[1]https://jaresan.github.io/simplod/documentation.pdf
[2]https://github.com/solid/userguide
[3]https://inrupt.net/
[4]https://inrupt.net/register

Upon clicking on the login button, the user is required to choose their Solid Pod provider either from the list provided or by specifying it themselves.

Figure 5.2: Picking Solid Pod provider



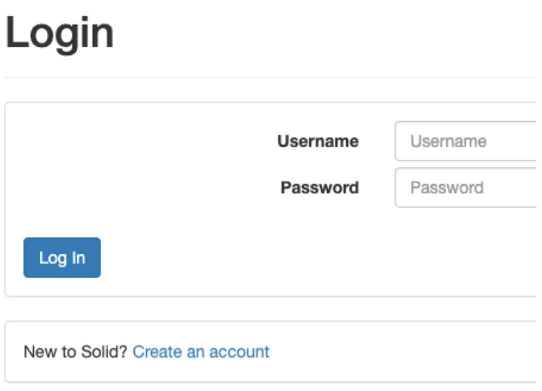
Log in to **jaresan.github.io**

Please enter your WebID or the URL of your identity provider:

https://my-identity.provider

After the provider is chosen, the user is able to authenticate via the provider's login screen.

Figure 5.3: Provider login screen



Login

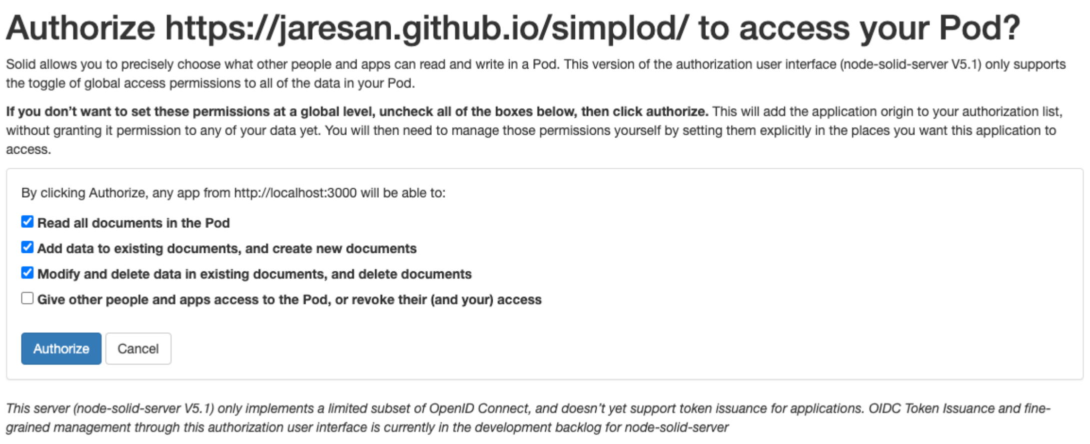
Username: Username

Password: Password

New to Solid? Create an account

When the login is successful, the user is requested to grant permission to the application which would allow it to save and handle its data across the user's Solid Pod.

Figure 5.4: App permission prompt



Authorize https://jaresan.github.io/simplod/ to access your Pod?

Solid allows you to precisely choose what other people and apps can read and write in a Pod. This version of the authorization user interface (node-solid-server V5.1) only supports the toggle of global access permissions to all of the data in your Pod.

If you don't want to set these permissions at a global level, uncheck all of the boxes below, then click authorize. This will add the application origin to your authorization list, without granting it permission to any of your data yet. You will then need to manage those permissions yourself by setting them explicitly in the places you want this application to access.

By clicking Authorize, any app from http://localhost:3000 will be able to:

- [x] **Read all documents in the Pod**
- [x] **Add data to existing documents, and create new documents**
- [x] **Modify and delete data in existing documents, and delete documents**
- [ ] **Give other people and apps access to the Pod, or revoke their (and your) access**

This server (node-solid-server V5.1) only implements a limited subset of OpenID Connect, and doesn't yet support token issuance for applications. OIDC Token Issuance and fine-grained management through this authorization user interface is currently in the development backlog for node-solid-server

If all of the above steps resolve correctly, the user is shown a positive feedback message and able to start working with the application fully.

With all of the steps outlined above complete, the application shall have all the permissions it needs to properly manage its data utilizing the user's Solid Pod.

Should the reader experience problems with the application data management, they are encouraged to resolve the problems manually directly in their Solid Pod, as outlined in Appendix A.

5.1.2 UI Elements

This subsection describes the layout of the application with a detailed explanation of each interactive element available to the user.

Layout

Figure 5.5: Layout

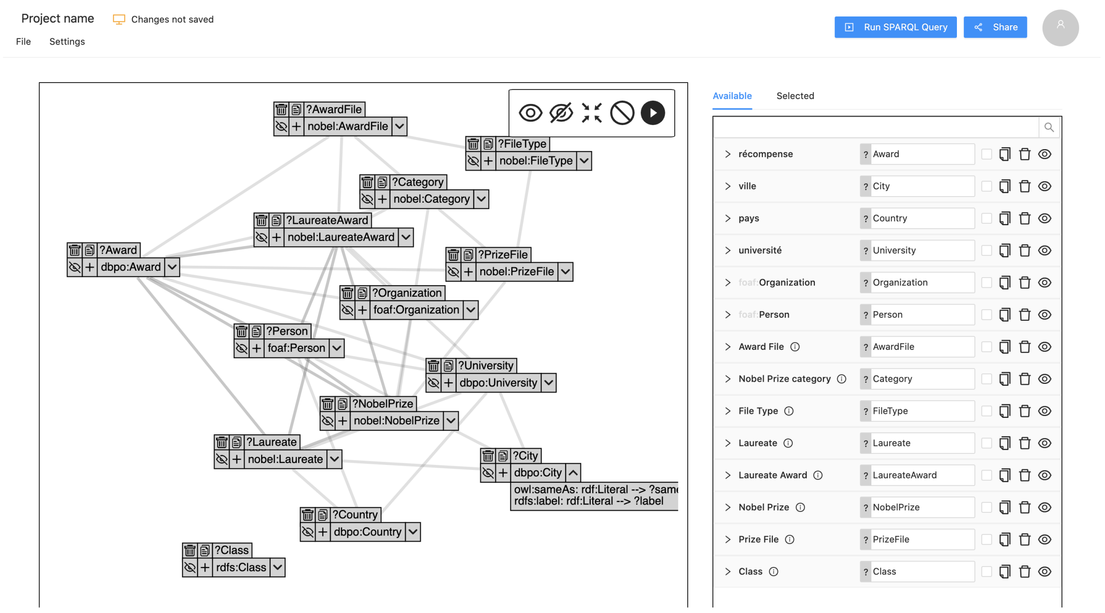


Figure 5.5 displays the overview "main screen" of the application. It is split into three main parts:

- Top bar

  Contains user information with project save status and functionality regarding application settings and file handling. Right part contains additional controls for the user, such as authentication and file sharing.

- Left part - Graph

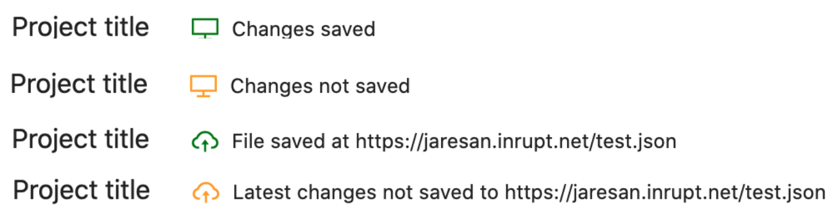
  Contains the graphical representation of the open data schema. Users are able to view different entities and their properties and the relations between them. Upper-right part of the graph also contains shortcuts to certain functionalities like showing all entities, clearing selection, and others. Described in subsection 5.1.3.

- Right part - Entity list

  A list view of the data displayed in the graph with additional controls. Contains a search bar to allow users to quickly filter out entities by their name. Described in subsection 5.1.4.

Project bar

Figure 5.6: Project status



Project title   Changes saved
Project title   Changes not saved
Project title   File saved at https://jaresan.inrupt.net/test.json
Project title   Latest changes not saved to https://jaresan.inrupt.net/test.json

Figure 5.6 contains the following:

- Project title

  Title of the project. Allows the user to edit by directly clicking in the text field.

- Change status

  Displays current project change status based on whether the newest changes are saved locally or in SOLID pod. Clicking the status text/icon saves the current state of the project to the corresponding location.

Avatar menu

Figure 5.7: Avatar menu

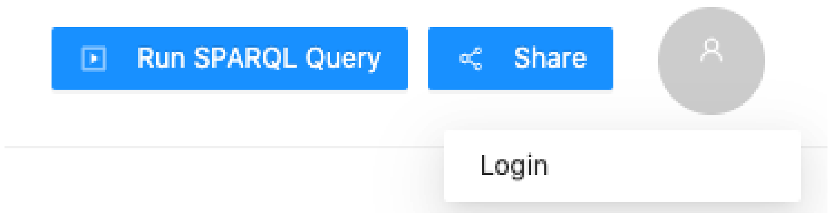


Figure 5.7 contains the following:

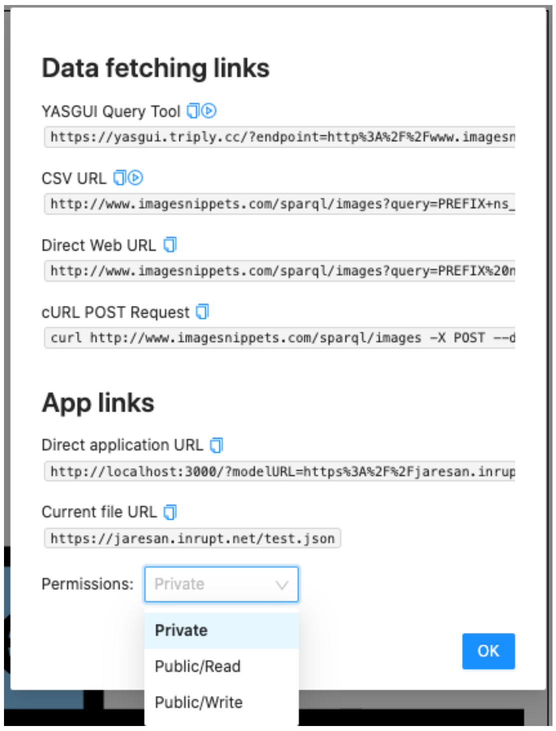
- Run SPARQL Query

  Opens a SPARQL query editor and runs the user generated query, displaying results.

- Share

  Allows the users to share the project and set view permissions for other users.

Figure 5.8: Share menu



Data fetching links

YASGUI Query Tool

`https://yasgui.triply.cc/?endpoint=http%3A%2F%2Fwww.imagesn`

CSV URL

`http://www.imagesnippets.com/sparql/images?query=PREFIX+ns_`

Direct Web URL

`http://www.imagesnippets.com/sparql/images?query=PREFIX%20n`

cURL POST Request

`curl http://www.imagesnippets.com/sparql/images -X POST --d`

App links

Direct application URL

`http://localhost:3000/?modelURL=https%3A%2F%2Fjaresan.inrup`

Current file URL

`https://jaresan.inrupt.net/test.json`

Permissions: Private

Clicking the "Share" item in Figure 5.7 opens Figure 5.8. These controls allow the user to share the project in different ways as follows:

- Data fetching links

  Links used to fetch the data represented in the project. These links could potentially be saved and used to retrieve specific data sets directly.

  - YASGUI Query Tool

    Opens YASGUI Query Tool[5] with the query representing the project loaded.
  - CSV URL

    Downloads the result set directly as a CSV if the endpoint properly supports it by adding `format=text/csv` parameter to the URL.
  - Direct Web URL

    Represents a GET request that directly returns the data set selected in the project.
  - cURL POST Request

    Since some of the endpoints might not be set up in a way that enables GET requests, the user is also provided with the option of running a cURL POST request that accepts CSV (Header "Accept: text/csv"). The endpoint has to support this functionality.

[5]`https://yasgui.triply.cc/`

- App links

  Links regarding the project and its usage in the app.

  - Direct application URL

    On access, launches the application and loads the project from the project file saved
  - Current file URL

    Displayed if the user has the project saved in a Solid pod. Remote location of the file.
  - Permissions

    Allows the user to set the file permissions directly.

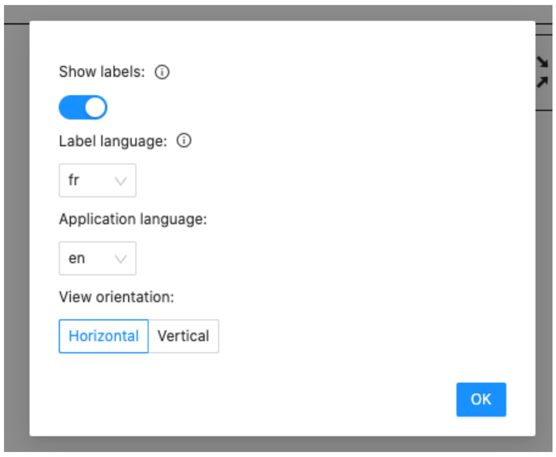
    Private - Can't be viewed by anyone else than the current user

    Public/read - Can be viewed by anyone but not edited

    Public/write - Can be edited by anyone

Settings

Figure 5.9: Settings menu



Show labels: ⓘ

Label language: ⓘ fr

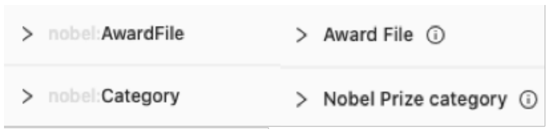
Application language: en

View orientation: [✓ Horizontal] Vertical

Clicking on "Settings" in Figure 5.5 opens the settings menu displayed in Figure 5.9 containing the following:

- Show labels

  Figure 5.10: Label

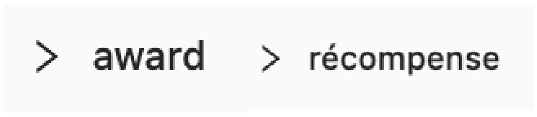
  Turning this option on/off allows the able to switch between human readable names for the entities or their IRI definitions.

  Figure 5.10 shows such an example with the labels disabled on the left and enabled on the right.

Human readable names are only displayed if they are provided by the endpoint.

- Label language

  Figure 5.11: Label language

  Allows the user to choose a language of the displayed labels if available.

  If the language selected is not available, the application defaults to displaying the English variant.

  Figure 5.11 shows an example of English labels on the left and French on the right.

- Application language

  Language of the application interface. Czech and English are provided with this work being published.

- View orientation

  Allows the user to select between horizontal/vertical view for the setup of the graph and the list screen.

File menu

Figure 5.12: File menu

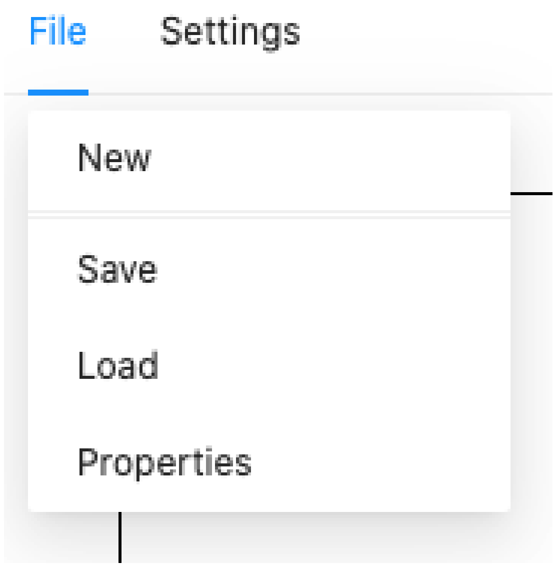
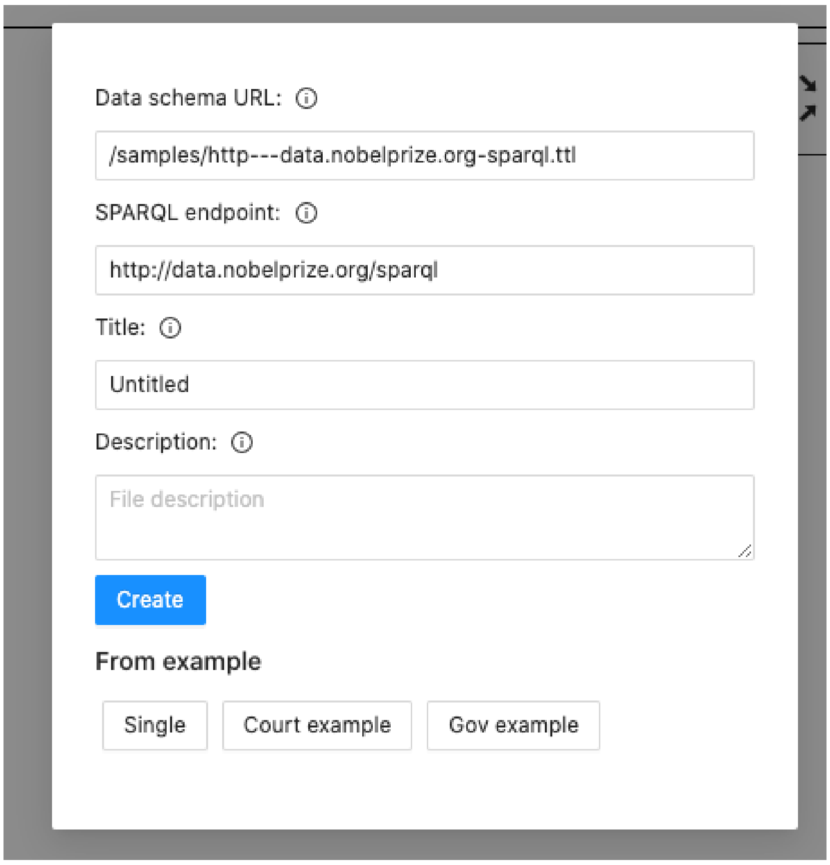


Figure 5.12 allows the user to create a new project, save/load one or change the project's properties.

Figure 5.13: New file


Data schema URL:

/samples/http---data.nobelprize.org-sparql.ttl

SPARQL endpoint:

http://data.nobelprize.org/sparql

Title:

Untitled

Description:

File description

Figure 5.13 shows new project window, allowing the user to create a new project with fields as follows:

- Data schema URL

  URL from which the data schema should be retrieved. This URL should return a file in the format described in subsection 2.5.2.

- SPARQL Endpoint

  URL of a SPARQL endpoint which will be queried for the data selected in the application.

- Title

  Title of the project

- Description

  Additional textual description of the project.

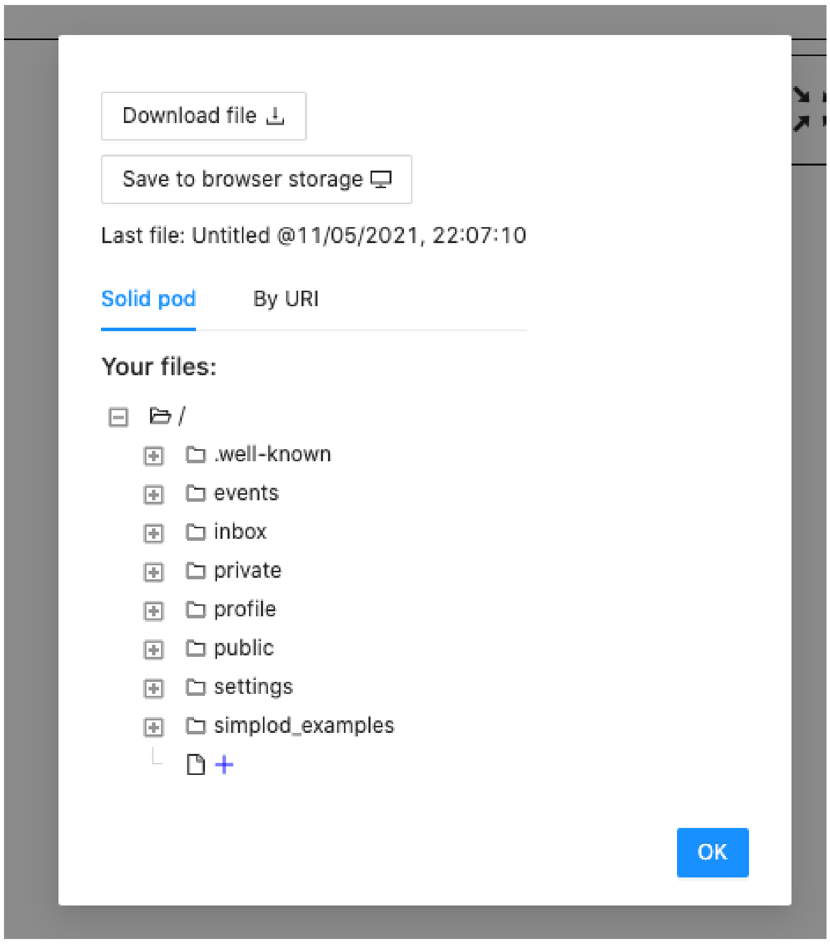
- Create

  Creates the project via the application, loading the data and populating the graph and the list.

- From example

  Users are also able to create a new project from a predefined set of examples for testing purposes or getting to know the application. This set of examples might not correspond to the examples displayed in Figure 5.13.

Figure 5.14: Save & load



Last file: Untitled @11/05/2021, 22:07:10

Your files:

/
- .well-known
- events
- inbox
- private
- profile
- public
- settings
- simplod_examples

Clicking on "save" in Figure 5.12 opens the save menu with the following items. Load menu is the same with opposite functionality:

- Download file

  Downloads a file representing the project to the user's disk. This file can be than shared and distributed to allow users to load the same project in the application.

- Save to browser storage

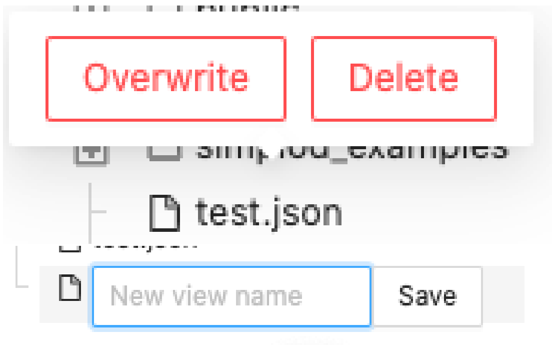
  Saves the current state of the project to the browsers storage, allowing the user to close the application and resume their work later. Due to the nature of this application, only one file can be saved to the browser storage at one time.

- Last file

  Description of the last file saved in the browser storage in the format "Project name @ DATE".

- Solid pod

Figure 5.15: Solid pod



Unauthorized users see a button **Login to Solid Pod**.

Authorized users see a list of their files in the Solid Pod they are currently logged in.

Selecting a file allows the user to delete or save to it directly.

Clicking the "+" button allows the user to create a new file in the selected folder.

- By URI

Figure 5.16: By URI

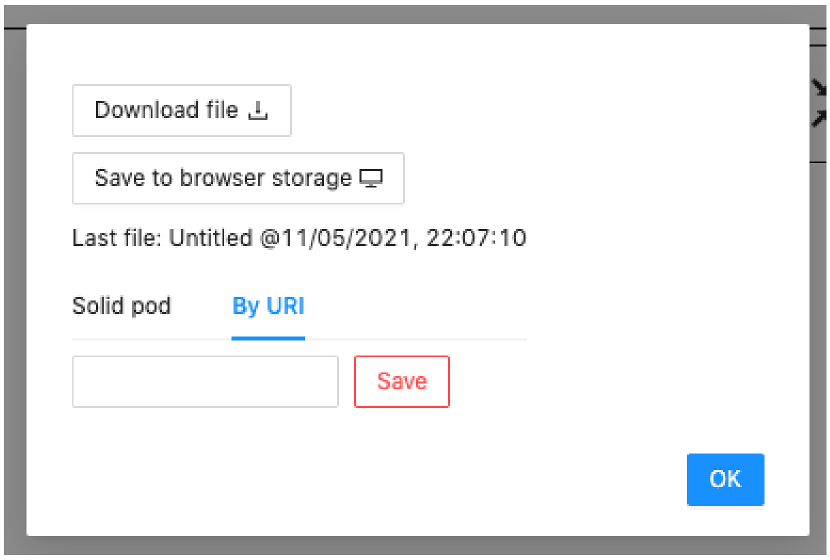
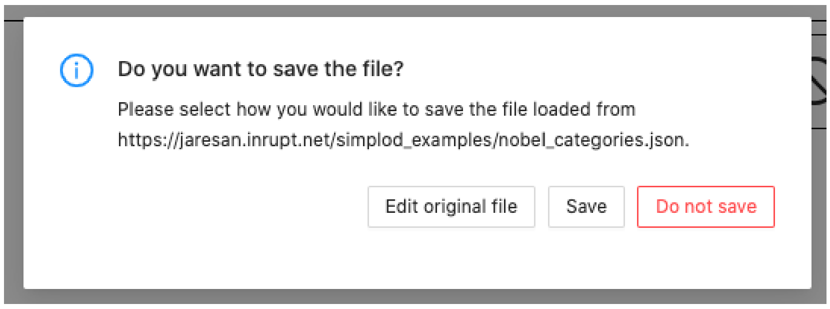


Figure 5.16 allows the user to specify the full URI path where the project file should be saved. The authenticated user has to be granted write permission to be able to save to this location.

Edit original file

Figure 5.17: Properties


Do you want to save the file?

Please select how you would like to save the file loaded from https://jaresan.inrupt.net/simplod_examples/nobel_categories.json.

If the user has write access to the project loaded from a remote location, they are asked to pick one of the following options:

- Edit original file

  Saving the changes directly modifies the original file at its location.

- Save

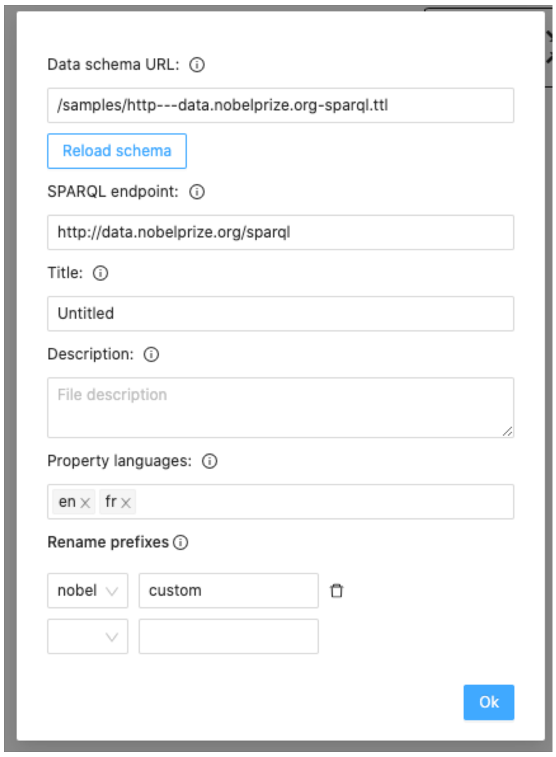
  Opens a dialog, allowing the user to save the file to a new location.

- Do not save

  Closes the prompt, letting the user pick a location of their choice later on.

Project properties

Figure 5.18: Properties


Data schema URL: /samples/http---data.nobelprize.org-sparql.ttl

SPARQL endpoint: http://data.nobelprize.org/sparql

Title: Untitled

Description: File description

Property languages: en fr

Rename prefixes: nobel custom

Figure 5.18 allows the user to change the properties of the project. Fields in this menu correspond to the same fields as in Figure 5.13 with extra items as follows:

- Property languages

  Languages that should appear in the resulting query for data properties supporting different languages. Will return every available language if nothing is specified.

- Custom prefixes

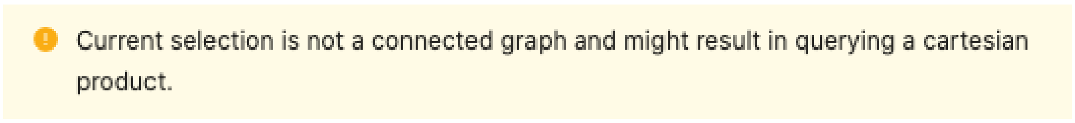
  Allows the users to rename the prefixes found in the application.

  Figure 5.18 represents an example where every ”nobel” prefix would be renamed to ”custom”, e.g. ”nobel:laureate” would become ”custom:laureate”.

  🗑 allows the users to delete their custom property entry.

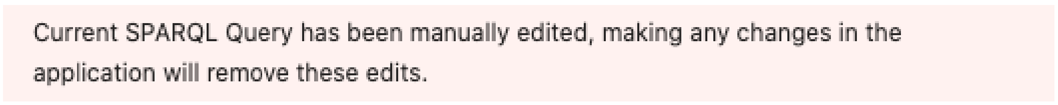
Warnings

Figure 5.19: Cartesian product warning


Current selection is not a connected graph and might result in querying a cartesian product.

This warning is displayed when the user queries for data in the graph that does not represent a strongly connected component and could therefore result in querying for a cartesian product.

Figure 5.20: Customized query warning


Current SPARQL Query has been manually edited, making any changes in the application will remove these edits.

This warning is shown if the SPARQL query has been manually edited. By changing anything regarding the selection, the user effectively removes these edits.

5.1.3 Graph interface

This subsection describes the graph part of the application and how users can interact with it. First the graph as a whole is described with its controls and controls for node separately following.

Graph component

Figure 5.21: Graph area

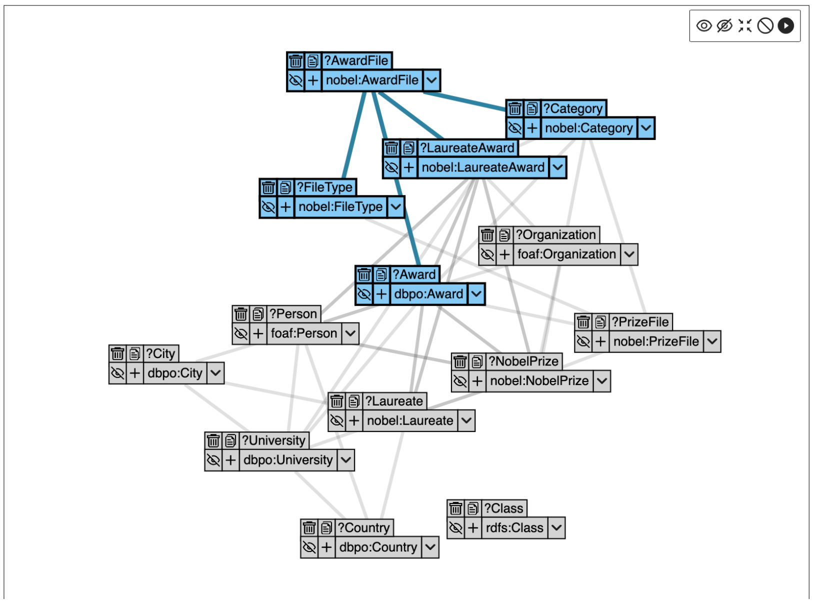


Figure 5.21 shows the data schema as a graph where nodes represent entities and edges represent the relationships between them. The users are able to interact with the graph in the following ways:

- Node drag

  Users are able to position the nodes in the graph by dragging them. This change in position is saved to the project model file and is persistent, loading the project again will result in the same positioning of the nodes.

- Empty space drag

  Users are able to navigate around the graph by dragging an empty space on it.

- Zoom

  Utilizing the mouse wheel/scroll controls, users are able to control the zoom level of the graph.

- Hover

  Hovering over an edge highlights its source and target nodes. Hovering over a node highlights all nodes connected to it with an edge and the corresponding edges.

Figure 5.22: Graph node

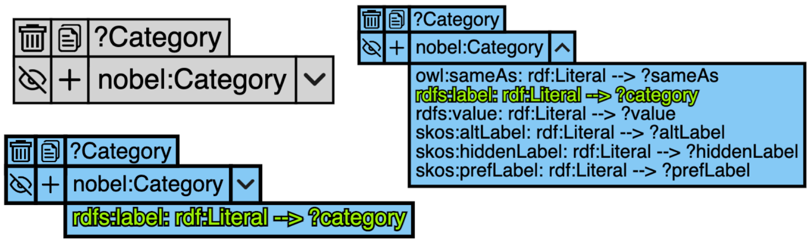

?Category
nobel:Category
?Category
nobel:Category
rdfs:label: rdf:Literal --> ?category
?Category
nobel:Category
owl:sameAs: rdf:Literal --> ?sameAs
rdfs:label: rdf:Literal --> ?category
rdfs:value: rdf:Literal --> ?value
skos:altLabel: rdf:Literal --> ?altLabel
skos:hiddenLabel: rdf:Literal --> ?hiddenLabel
skos:prefLabel: rdf:Literal --> ?prefLabel

Figure 5.22 represents a single entity from the data schema. Its controls are as follows:

- Highlight

  If some properties of the entity are selected, the entity is highlighted to easily distinguish it from other entities in the graph that are not being queried.

- Delete entity

  Remove the entity and its corresponding relationships from the schema altogether. Curators can use this feature to split up large data schemas into smaller, more specific chunks.

- Copy entity

  Creates a new instance of the same entity in the schema. This way users are able to query for the same entity types with different entity instances. In the nobel prize example provided users might want to query two different sets of countries, one for the people and one for their respective universities. Using only a single entity would not be able to achieve that in this case.

- "?Name"

  Name of the entity in the resulting data set. Can be changed in the list controls described in the next section.

- Hide

  Hides the entity from the schema.

- + Select all

  Selects all properties of the given entity.

- prefix:Name

  Entity type.

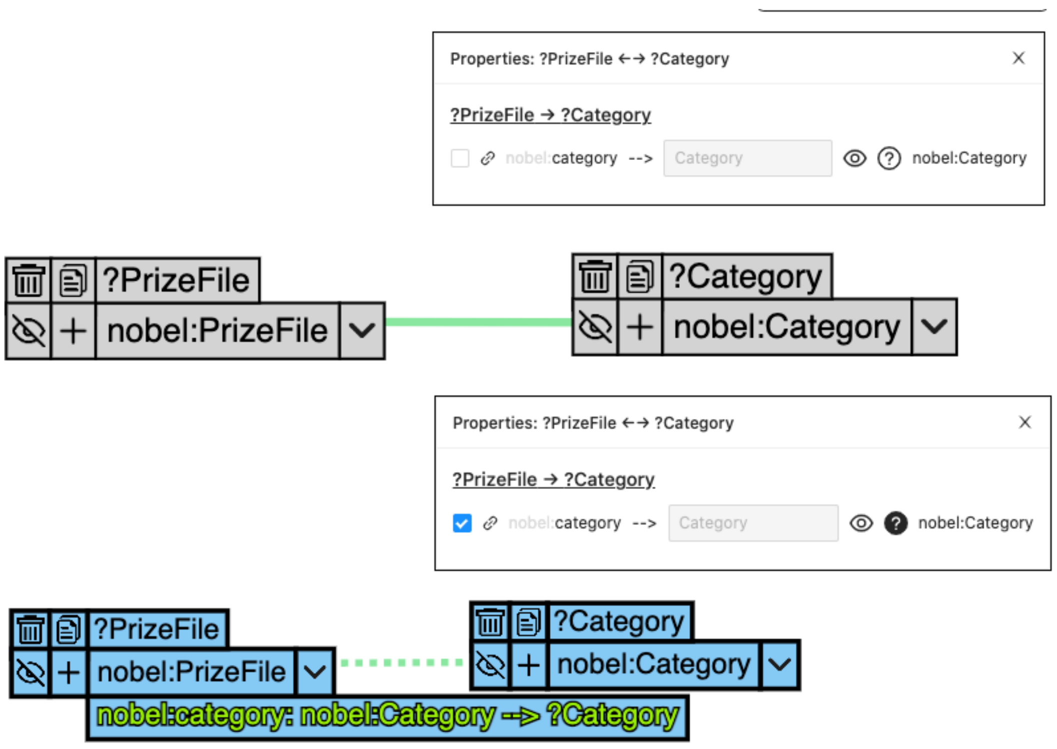
- ∨∧ Expand/Collapse

  Expands/collapses the container of the properties, displaying all data and object properties available on the entity. Collapsing the container keeps the selected properties visible.

- Property container

  List of properties for the given entity. This list contains both data properties and object properties. Selecting a property highlights both the node and the property itself.

Figure 5.23: Edge



As shown in Figure 5.23, edges in the graph represent relationships between entities in the data schema. If there exists an edge between two entities, there exists at least on property on one of the entities that has the other entity as a subject. The edge controls are as follows:

- Highlight

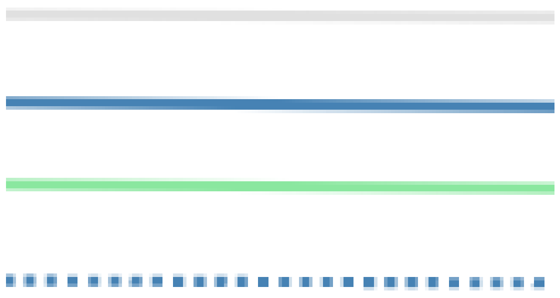
  Hovering over an edge highlights it and also its corresponding nodes. Highlighting a node from Figure 5.22 also highlights its all corresponding edges and their end nodes.

- Click

  Clicking an edge opens a menu with a list of properties the edge represents. Users are able to perform all actions on these properties the same way as they would via the list view.

Edges can also appear with different styles based on the user's interaction with them:

Figure 5.24: Edge states



- Default grey color

  None of the properties represented by the edge are selected.

- Blue color

  Some properties represented by the edge are selected.

- Green color

  The edge has been selected by clicking on it and its description menu is being shown. The user can deselect the edge by clicking anywhere else in the graph.

- Dash pattern

  All of the edge's selected properties are marked as optional.

Figure 5.25: Graph toolbar

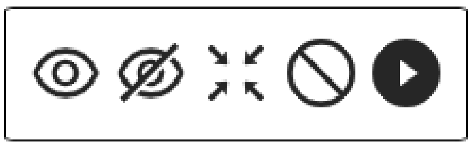


Figure 5.25 represents a toolbar with access to action shortcuts for the user's convenience as follows:

- Show all

  Toggles all entities as shown that were previously hidden via Figure 5.22 .

- Hide rest

  Toggles all entities that are not selected as hidden, functionally the same as toggling entity as hidden directly in the graph via Figure 5.22 .

- Fit into view

  Fits the whole graph in to the current graph container, allowing the users to view all entities in the window at once.

- ⊘ Deselect all

  Deselect all currently selected properties and entities.

- ⦿ Run Query

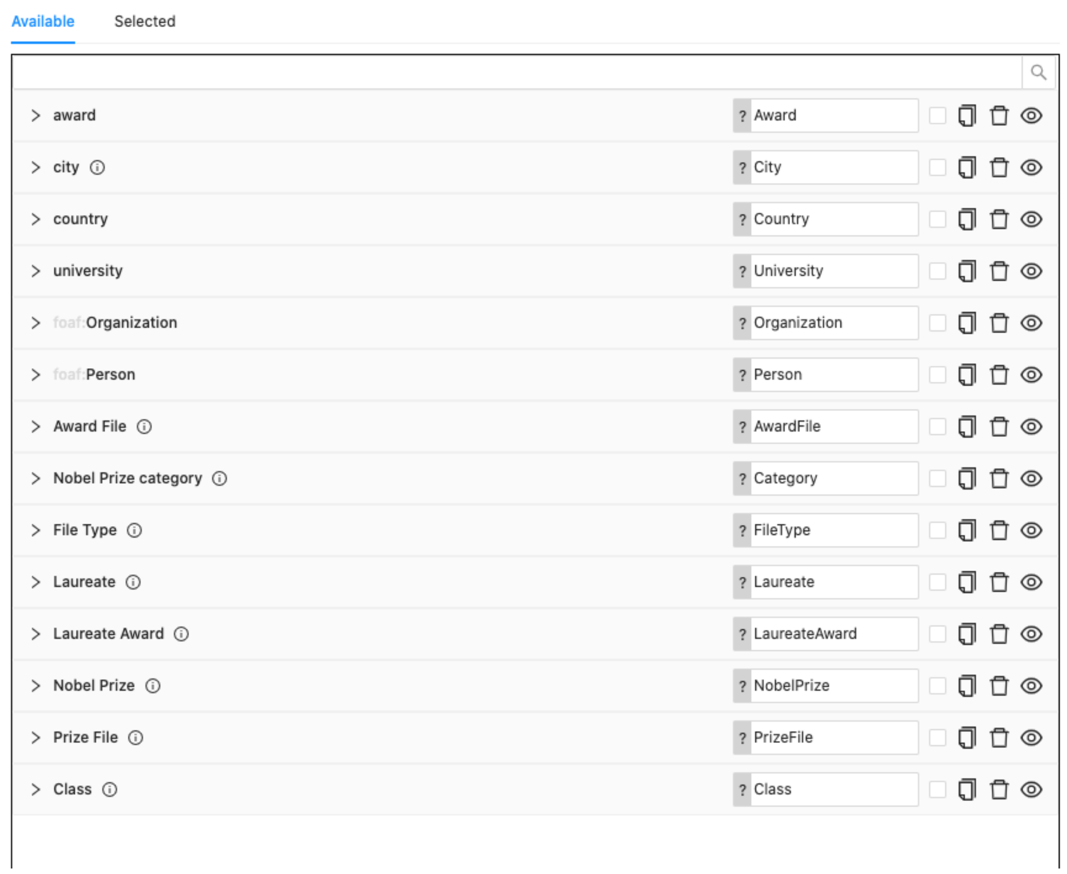
  Opens SPARQL Query editor and runs the query representing user's selection.

5.1.4 List view

Similarly to the graph interface, the user can use the list view to achieve the same results. This subsection describes the elements of the list view and how to interact with them.

List overview

Figure 5.26: List overview



award	? Award
city	? City
country	? Country
university	? University
foaf:Organization	? Organization
foaf:Person	? Person
Award File	? AwardFile
Nobel Prize category	? Category
File Type	? FileType
Laureate	? Laureate
Laureate Award	? LaureateAward
Nobel Prize	? NobelPrize
Prize File	? PrizeFile
Class	? Class

The list displays all the entities to be found in the data schema with controls that enable similar interaction to the ones described in subsection 5.1.3.

Starting from the top:

List view controls

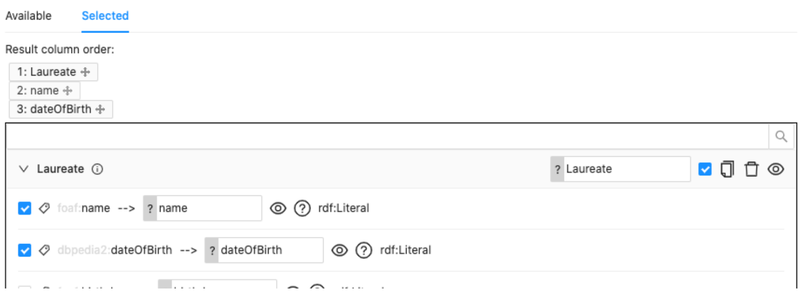
- Available tab

  This tab displays all available entities in the data schema. If the user deletes an entity from the project, this list is updated accordingly and the entity is removed from it.

- Selected Tab

  This tab displays only entities that themselves, or their properties, are requested in the result set by the user.

Figure 5.27: Selected tab



Result column order:
1: Laureate
2: name
3: dateOfBirth

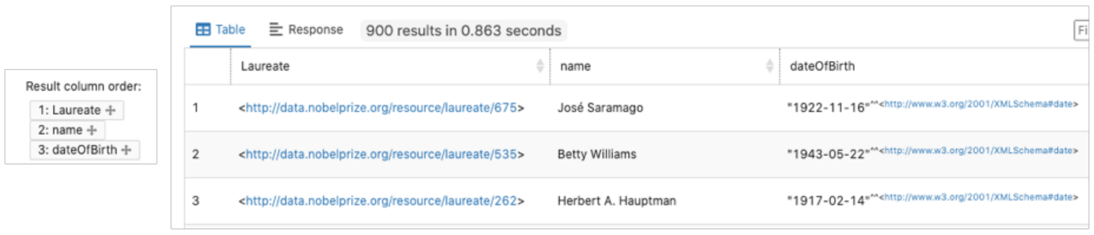
Laureate — ? Laureate

foaf:name --> ? name rdf:Literal

dbpedia2:dateOfBirth --> ? dateOfBirth rdf:Literal

The user is able to change the order of the requested resources in the top part by dragging the entries to the desired position, resulting in different order of the queried variables.

Figure 5.28: Column order example 1

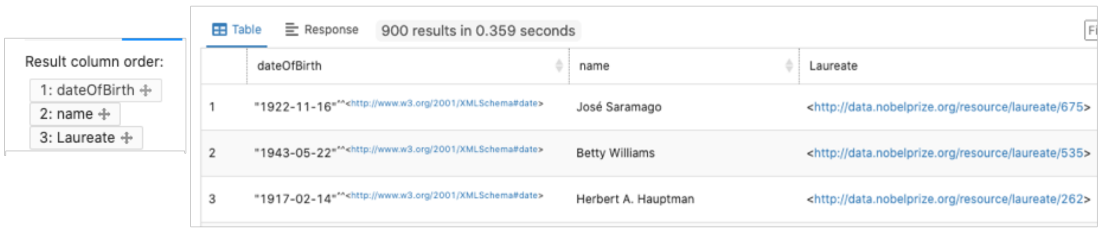


900 results in 0.863 seconds

Result column order:
1: Laureate
2: name
3: dateOfBirth

	Laureate	name	dateOfBirth
1	<http://data.nobelprize.org/resource/laureate/675>	José Saramago	"1922-11-16"^^<http://www.w3.org/2001/XMLSchema#date>
2	<http://data.nobelprize.org/resource/laureate/535>	Betty Williams	"1943-05-22"^^<http://www.w3.org/2001/XMLSchema#date>
3	<http://data.nobelprize.org/resource/laureate/262>	Herbert A. Hauptman	"1917-02-14"^^<http://www.w3.org/2001/XMLSchema#date>

Figure 5.29: Column order example 2



900 results in 0.359 seconds

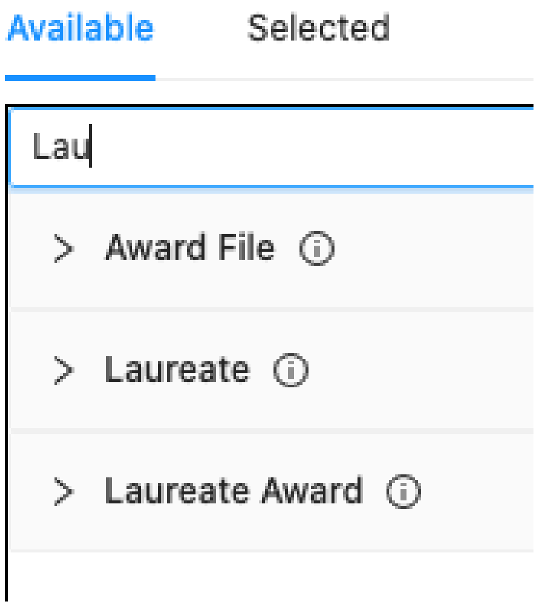
Result column order:
1: dateOfBirth
2: name
3: Laureate

	dateOfBirth	name	Laureate
1	"1922-11-16"^^<http://www.w3.org/2001/XMLSchema#date>	José Saramago	<http://data.nobelprize.org/resource/laureate/675>
2	"1943-05-22"^^<http://www.w3.org/2001/XMLSchema#date>	Betty Williams	<http://data.nobelprize.org/resource/laureate/535>
3	"1917-02-14"^^<http://www.w3.org/2001/XMLSchema#date>	Herbert A. Hauptman	<http://data.nobelprize.org/resource/laureate/262>

- Search bar

  This bar allows the user to filter out results by text search with immediate response. The search is run on the labels, descriptions and the actual IRI representation.

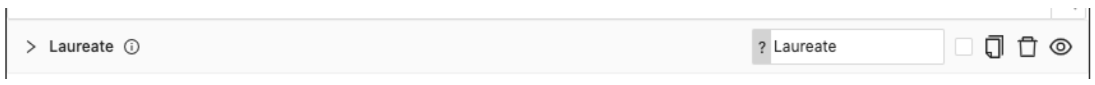
Figure 5.30: Search functionality



Entity rows

Each entity is represented by its own row entry in the list view.

Figure 5.31: Entity row

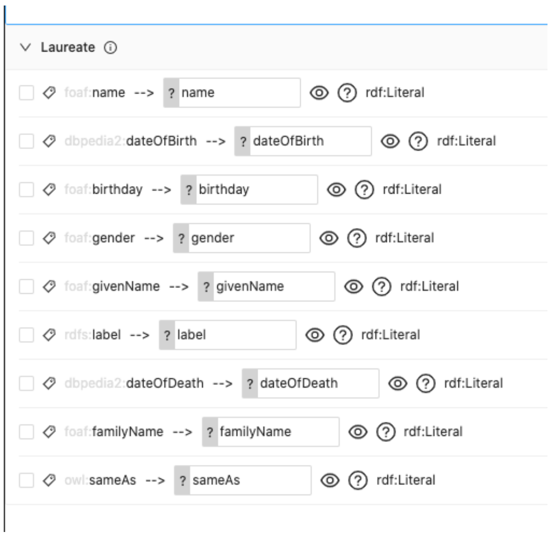


Every such row can be interacted with in the following ways:

- › Expand/Collapse icon

  Clicking this icon allows the user to expand/collapse the properties linked to this entity.

Figure 5.32: Expanded properties

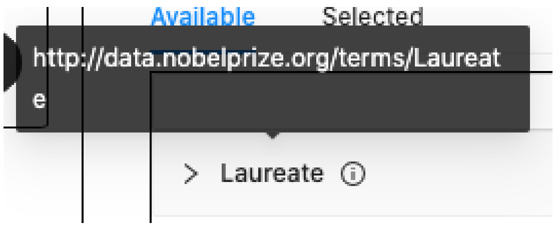

Laureate

foaf:name	-->	? name		rdf:Literal
dbpedia2:dateOfBirth	-->	? dateOfBirth		rdf:Literal
foaf:birthday	-->	? birthday		rdf:Literal
foaf:gender	-->	? gender		rdf:Literal
foaf:givenName	-->	? givenName		rdf:Literal
rdfs:label	-->	? label		rdf:Literal
dbpedia2:dateOfDeath	-->	? dateOfDeath		rdf:Literal
foaf:familyName	-->	? familyName		rdf:Literal
owl:sameAs	-->	? sameAs		rdf:Literal

- Title hover

  Hovering over an entity name displays its full IRI as a tooltip.

Figure 5.33: Entity title hover

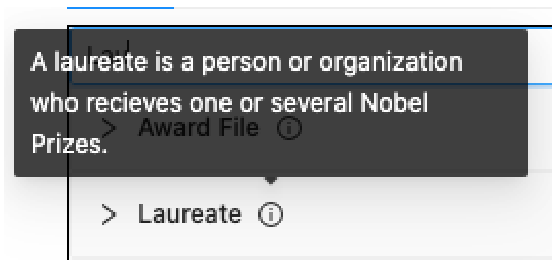


http://data.nobelprize.org/terms/Laureate

- ℹ hover

  Hovering over the ℹ icon displays human readable description of the entity if available (has to be supported by the endpoint set in the project).

Figure 5.34: Entity description

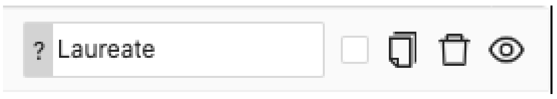


A laureate is a person or organization who recieves one or several Nobel Prizes.

Entity row controls

On the right side, every entity row includes also quick actions similar to the actions in described Figure 5.1.3

Figure 5.35: Entity description



- ? - Variable name

  Variable name to be used in the result set of the SPARQL query as per example:

Figure 5.36: Variable name field

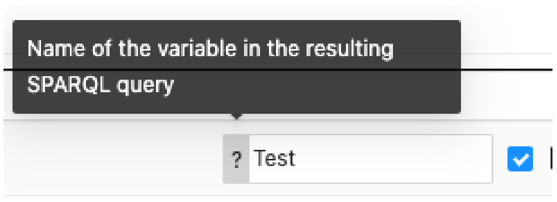
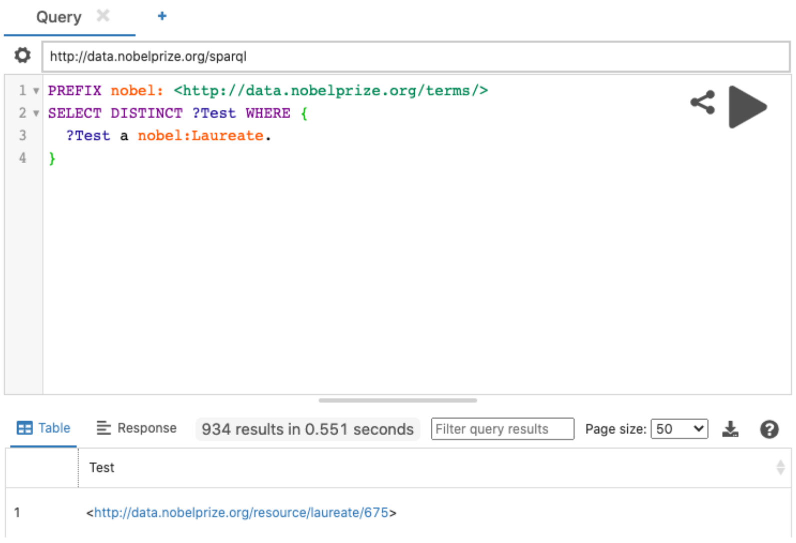


Figure 5.37: Renamed variable result

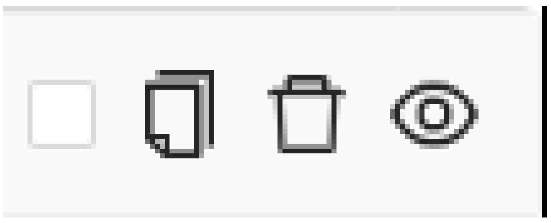

Query
http://data.nobelprize.org/sparql
Table
Response
934 results in 0.551 seconds
Filter query results
Page size: 50

```
PREFIX nobel: <http://data.nobelprize.org/terms/>
SELECT DISTINCT ?Test WHERE {
  ?Test a nobel:Laureate.
}
```

	Test
1	<http://data.nobelprize.org/resource/laureate/675>

Right side of the entity row offers quick actions as follows:

Figure 5.38: Entity row actions



- ✓ - Select entity

  Queries the entity under given variable name.

- 🗐 - Copy entity

  Creates another instance of the same entity, same behaviour as in Figure 5.1.3.

- 🗑 - Delete entity

  Deletes the entity instance from the data schema, same behaviour as in Figure 5.1.3.

- 👁 - Hide entity

  Hides the entity in the data schema, same behaviour as in Figure 5.1.3.

Property row

Property rows are divided into **data properties** and **object properties** (targeting other entities in the graph, resulting in a graph edge) with the property's target being specified at the end of the row. Every property has its own row in the list view as follows:

Figure 5.39: Data property row

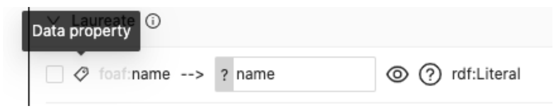
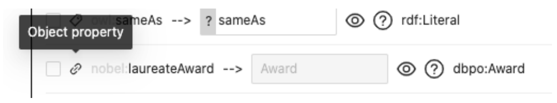


Figure 5.40: Object property row



The control elements are as follows:

- ✓ - Select property

  Selects the property under given variable name.

- 🏷 or 🔗 - Property type

  An icon representing the type of the property, data or object.

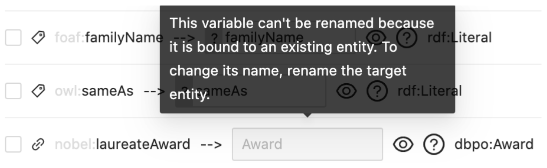
- "XXX → ? or grey field"

  XXX represents the property's predicate.

  ? denotes variable name field, same as for entity rows.

  Greyed out field is present on **object properties**, informing the user that the name of the target entity has to changed in order to change the name of this property as per the tooltip:

Figure 5.41: Object property variable field tooltip


This variable can't be renamed because it is bound to an existing entity. To change its name, rename the target entity.

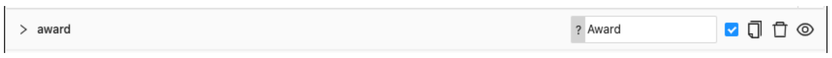
foaf:familyName --> ?familyName rdf:Literal

owl:sameAs --> ?sameAs rdf:Literal

nobel:laureateAward --> Award dbpo:Award

The user has to rename the entity directly if they wish to query the property under a different name:

Figure 5.42: Object property target



- 👁 - Hide property from the result set

  Hides the property from the result set. Useful when user wants to query only entities with an existing relation but does not care about the property value, e.g. user wants to query theses that have already been submitted (have property "submitDate") but does not care about the actual date itself.

- ? - Mark property as optional

  Marks property as optional.

5.1.5 Examples

The following subsection outlines example scenarios which can be followed to introduce the user to the features of the application.

Every example is started from the new start of the application with **default settings**. The reader is welcome to use the accompanying deployment of this work on GitHub[6] by clicking **Demo**[7].

Nobel prize categories - graph

The first example is based on a data schema of Nobel prizes. This data schema represents the information about Nobel prizes and their laureates. Let's illustrate a simple example where we would like to know what Nobel prize categories there are.

First we have to load the data schema in the application, we can do that either by accessing the demo application[8] with the data already encoded at or by following the steps below:

1. Click *File* → *New*

[6]`https://jaresan.github.io/simplod/`

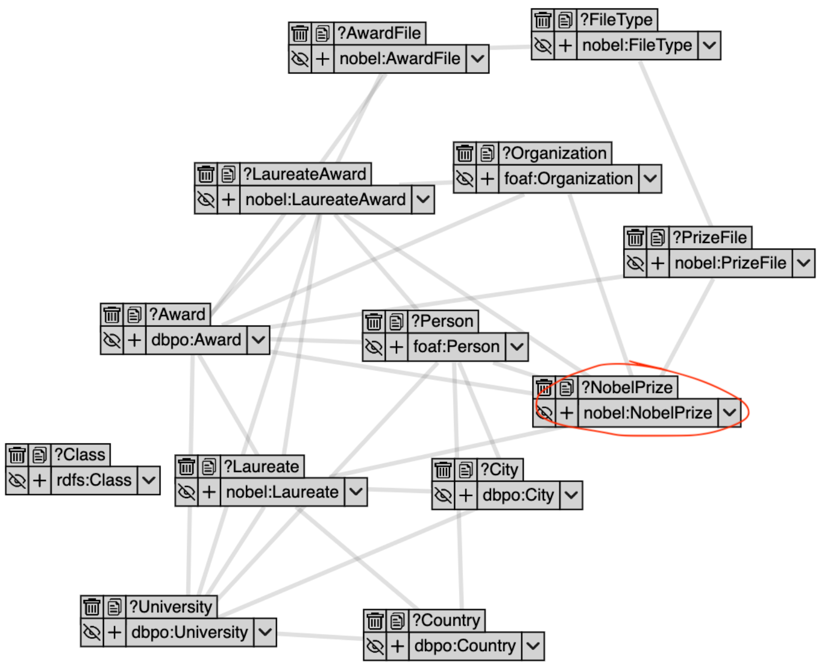
[7]`https://jaresan.github.io/simplod/build/index.html`

[8]`https://jaresan.github.io/simplod/demo.html?schemaURL=https://jaresan.github.io/simplod/example.ttl&endpointURL=https://data.nobelprize.org/store/sparql`

2. Create a New project with the following configuration:
   Data Schema:
   `https://jaresan.github.io/simplod/example.ttl`
   Endpoint: `https://data.nobelprize.org/store/sparql`

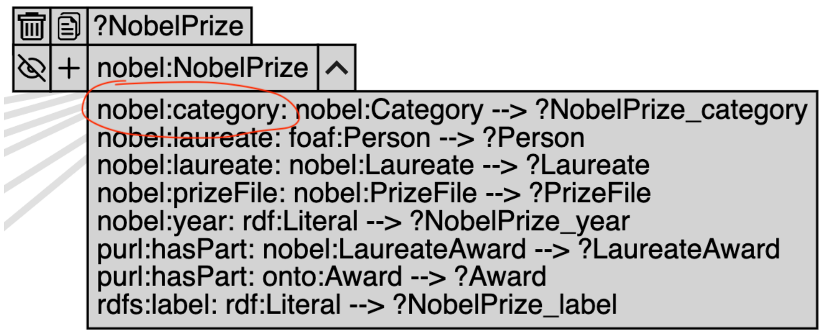
3. Click Create

Figure 5.43: Nobel prize example default view



When we open the application (depicted in Figure 5.43) we can notice the ”NobelPrize” entity in the graph.

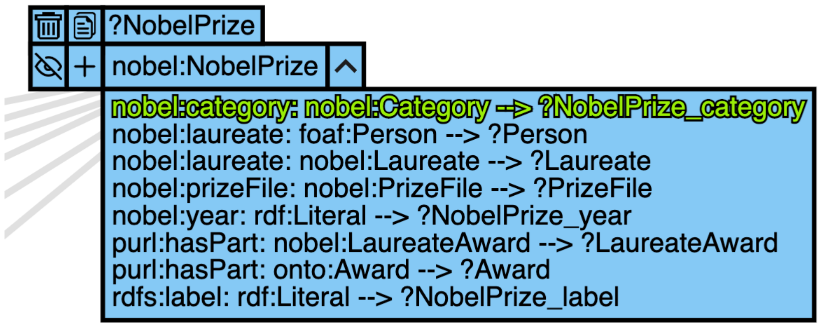
Figure 5.44: NobelPrize entity



nobel:category: nobel:Category --> ?NobelPrize_category
nobel:laureate: foaf:Person --> ?Person
nobel:laureate: nobel:Laureate --> ?Laureate
nobel:prizeFile: nobel:PrizeFile --> ?PrizeFile
nobel:year: rdf:Literal --> ?NobelPrize_year
purl:hasPart: nobel:LaureateAward --> ?LaureateAward
purl:hasPart: onto:Award --> ?Award
rdfs:label: rdf:Literal --> ?NobelPrize_label

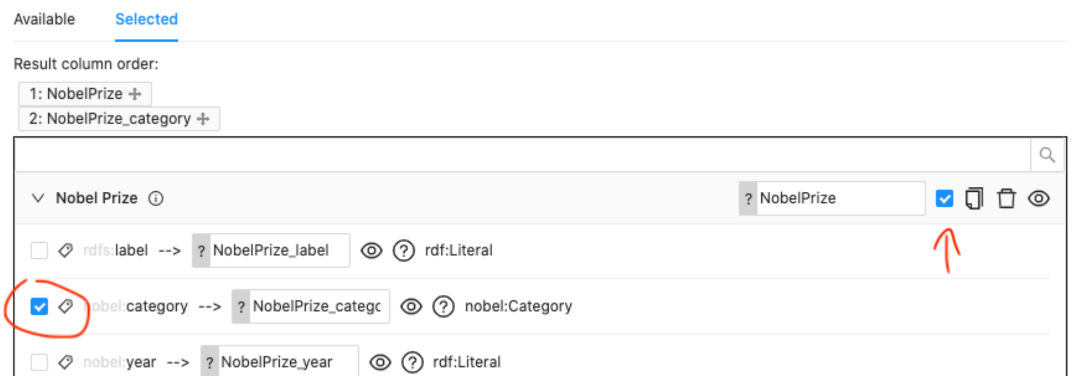
Clicking on the entity, we can expand its properties and see what it is linked to. Let’s go ahead and select the **nobel:category** property.

Figure 5.45: Selected property


?NobelPrize
nobel:NobelPrize
nobel:category: nobel:Category --> ?NobelPrize_category
nobel:laureate: foaf:Person --> ?Person
nobel:laureate: nobel:Laureate --> ?Laureate
nobel:prizeFile: nobel:PrizeFile --> ?PrizeFile
nobel:year: rdf:Literal --> ?NobelPrize_year
purl:hasPart: nobel:LaureateAward --> ?LaureateAward
purl:hasPart: onto:Award --> ?Award
rdfs:label: rdf:Literal --> ?NobelPrize_label

For a more detailed overview of what we have selected, we can take a look at the list view under the graph, or next to it, in the tab **selected**.

Figure 5.46: List view



Result column order:
1: NobelPrize
2: NobelPrize_category

Nobel Prize ? NobelPrize
rdfs:label --> ? NobelPrize_label rdf:Literal
nobel:category --> ? NobelPrize_categc nobel:Category
nobel:year --> ? NobelPrize_year rdf:Literal

Here we can see we have selected the category property, named **NobelPrize_category**. Also the entity itself is selected (checkbox next to **NobelPrize**), which will select the prizes as well. For the sake of the example, let's leave the selection as is. Clicking the "Run SPARQL Query" at the top of the screen, the editor opens and we can see the fetched results:

Figure 5.47: Run Query

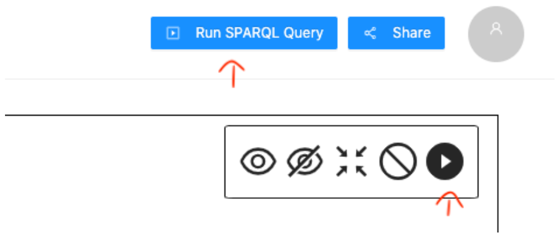
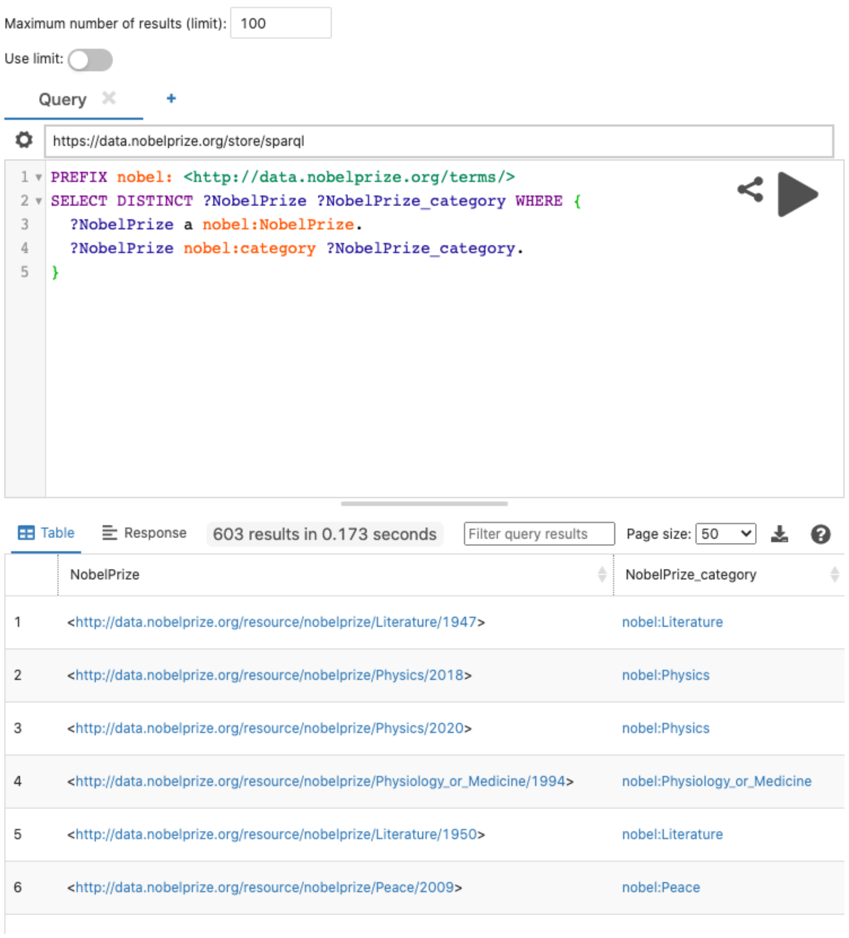


Figure 5.48: SPARQL Results


Maximum number of results (limit): 100

Use limit:

https://data.nobelprize.org/store/sparql

```
PREFIX nobel: <http://data.nobelprize.org/terms/>
SELECT DISTINCT ?NobelPrize ?NobelPrize_category WHERE {
  ?NobelPrize a nobel:NobelPrize.
  ?NobelPrize nobel:category ?NobelPrize_category.
}
```

603 results in 0.173 seconds

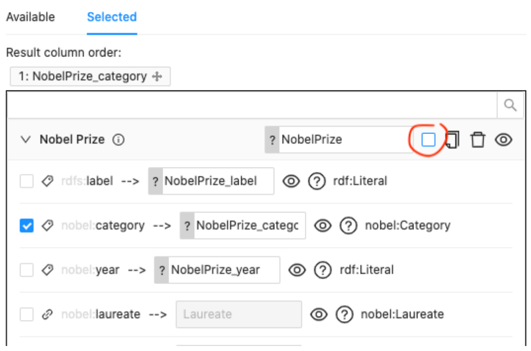
Page size: 50

	NobelPrize	NobelPrize_category
1	<http://data.nobelprize.org/resource/nobelprize/Literature/1947>	nobel:Literature
2	<http://data.nobelprize.org/resource/nobelprize/Physics/2018>	nobel:Physics
3	<http://data.nobelprize.org/resource/nobelprize/Physics/2020>	nobel:Physics
4	<http://data.nobelprize.org/resource/nobelprize/Physiology_or_Medicine/1994>	nobel:Physiology_or_Medicine
5	<http://data.nobelprize.org/resource/nobelprize/Literature/1950>	nobel:Literature
6	<http://data.nobelprize.org/resource/nobelprize/Peace/2009>	nobel:Peace

In Figure 5.48 we can see the result created by our selection. First we have the **NobelPrize** which represents the IRI of a Nobel Prize. Second we have the **NobelPrize_category** property, which is the textual representation of the category for the given prize.

Considering we wanted to find only what categories Nobel prizes are awarded in, this result is superfluous. To get rid of the prizes, we can go back to the list view to deselect them:

Figure 5.49: List view

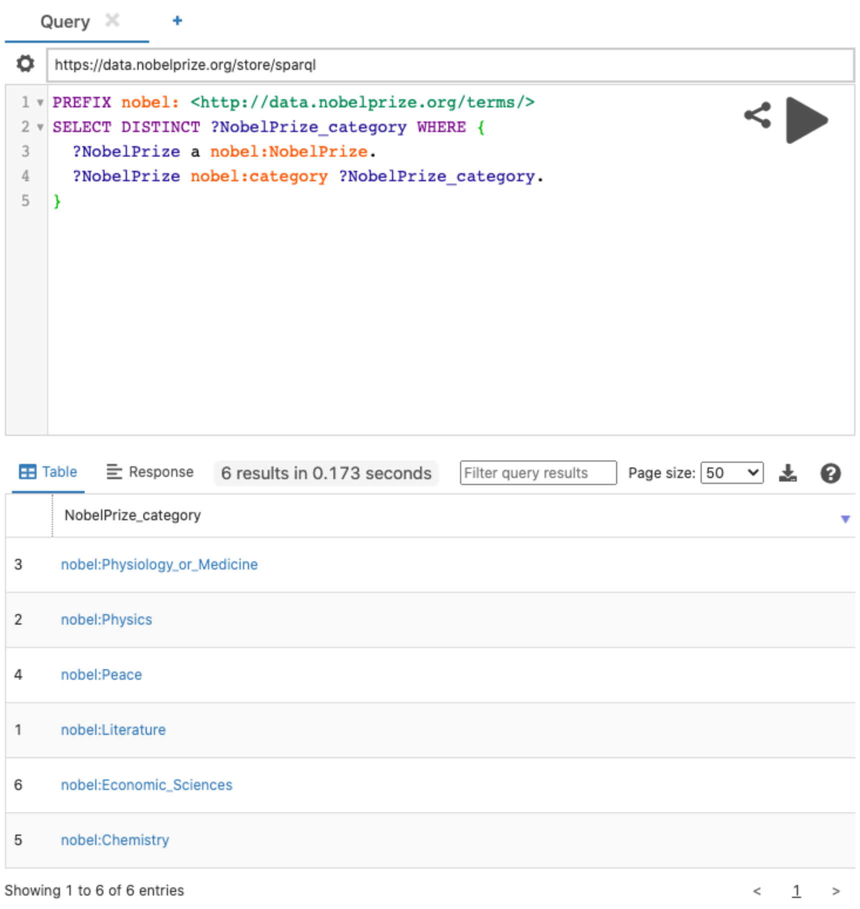

Result column order:

1: NobelPrize_category

Nobel Prize		? NobelPrize	
☐	rdfs:label -->	? NobelPrize_label	rdf:Literal
☑	nobel:category -->	? NobelPrize_categc	nobel:Category
☐	nobel:year -->	? NobelPrize_year	rdf:Literal
☐	nobel:laureate -->	Laureate	nobel:Laureate

Deselecting the entity will remove its IRI from the results set. Executing the query again, we get the following:

Figure 5.50: Cleaner results



Query
https://data.nobelprize.org/store/sparql

```
PREFIX nobel: <http://data.nobelprize.org/terms/>
SELECT DISTINCT ?NobelPrize_category WHERE {
  ?NobelPrize a nobel:NobelPrize.
  ?NobelPrize nobel:category ?NobelPrize_category.
}
```

Table
Response
6 results in 0.173 seconds
Filter query results
Page size: 50

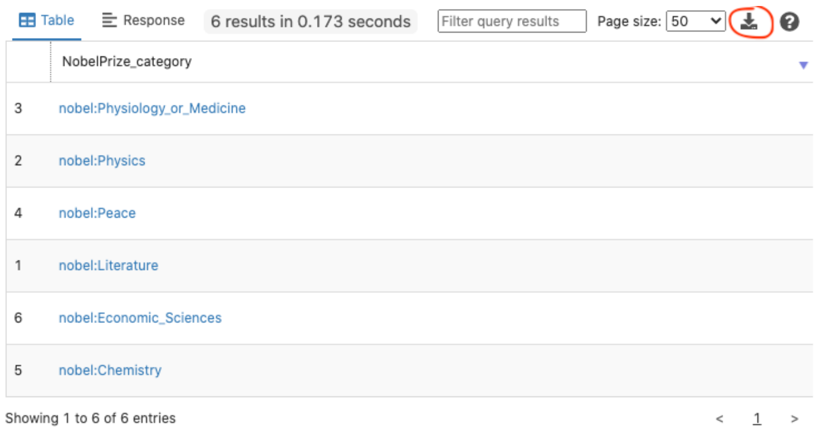
	NobelPrize_category
3	nobel:Physiology_or_Medicine
2	nobel:Physics
4	nobel:Peace
1	nobel:Literature
6	nobel:Economic_Sciences
5	nobel:Chemistry

Showing 1 to 6 of 6 entries

In Figure 5.50 we can see that we have now only fetched the categories. Anybody shown these results can immediately understand what they represent.

Now that we've fetched the data, we might want to share them. We can do so by downloading the result directly in the CSV format and sharing that file:

Figure 5.51: Download CSV

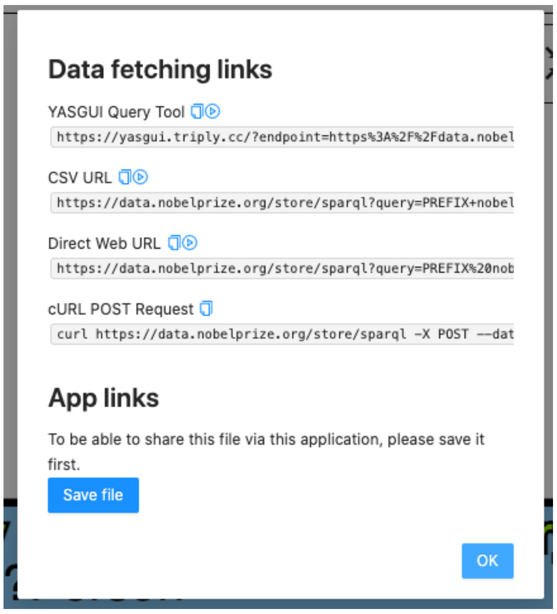


	NobelPrize_category
3	nobel:Physiology_or_Medicine
2	nobel:Physics
4	nobel:Peace
1	nobel:Literature
6	nobel:Economic_Sciences
5	nobel:Chemistry

Showing 1 to 6 of 6 entries

We can also share the data by sharing a link to a third party tool populated with our query. To do that, we can open the share menu via the "Share" button at the top of the screen. By clicking on the 🗐 icon for **YASGUI Query Tool**, we copy the URL with the query encoded into our clipboard and can then just paste it in the browser and view the result. We can also click the ⦿ icon to launch the tool directly.

Figure 5.52: Copy yasgui query URL



Data fetching links

YASGUI Query Tool

```
https://yasgui.triply.cc/?endpoint=https%3A%2F%2Fdata.nobel
```

CSV URL

```
https://data.nobelprize.org/store/sparql?query=PREFIX+nobel
```

Direct Web URL

```
https://data.nobelprize.org/store/sparql?query=PREFIX%20nob
```

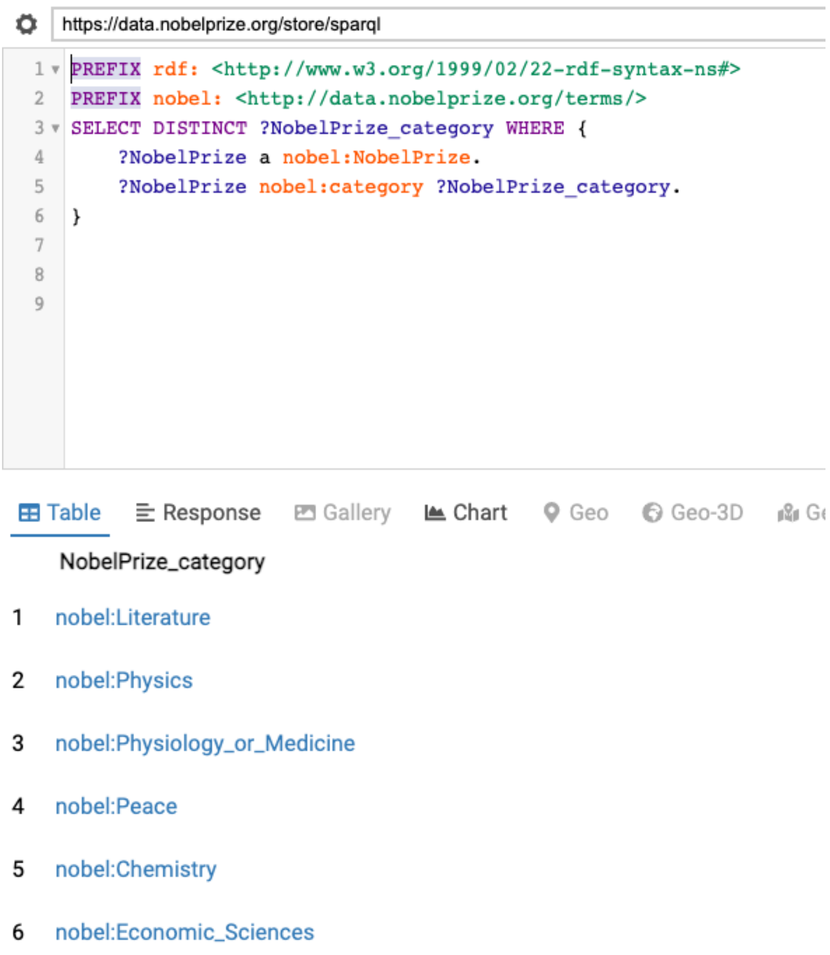
cURL POST Request

```
curl https://data.nobelprize.org/store/sparql -X POST --dat
```

App links

To be able to share this file via this application, please save it first.

Figure 5.53: Yasgui query results


https://data.nobelprize.org/store/sparql

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX nobel: <http://data.nobelprize.org/terms/>
SELECT DISTINCT ?NobelPrize_category WHERE {
    ?NobelPrize a nobel:NobelPrize.
    ?NobelPrize nobel:category ?NobelPrize_category.
}
```

	NobelPrize_category
1	nobel:Literature
2	nobel:Physics
3	nobel:Physiology_or_Medicine
4	nobel:Peace
5	nobel:Chemistry
6	nobel:Economic_Sciences

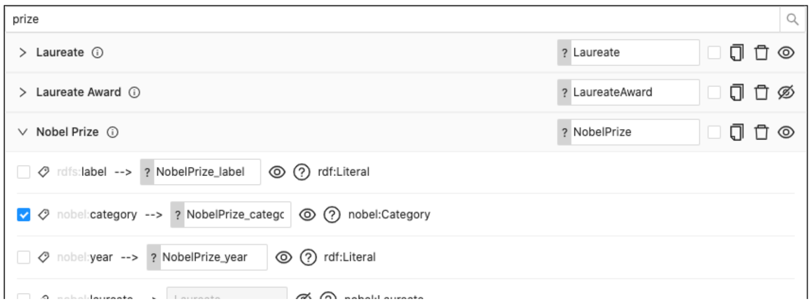
We can also use ”Direct access URL” which returns the results directly, or get the cURL POST request to use in the terminal. All of the share options are described in Figure 5.1.2.

Nobel prize categories - list view

What if we don’t want to navigate through the graph because it might seem too clunky?

We can use the search functionality in the list view. We are looking for Nobel prizes. By typing ”prize” in the search field, we can see entity rows being filtered out based on their matching text.

Figure 5.54: List view


prize

- Laureate — ? Laureate
- Laureate Award — ? LaureateAward
- Nobel Prize — ? NobelPrize
  - [ ] rdfs:label --> ? NobelPrize_label — rdf:Literal
  - [✓] nobel:category --> ? NobelPrize_categc — nobel:Category
  - [ ] nobel:year --> ? NobelPrize_year — rdf:Literal

In this case, we can select the property directly by checking the box on its left side. The rest of the steps is the same as the end for the graph variant. The graph and list are connected and new changes are reflected in both of these components.

Nobel prize laureates

Continuing with the example of nobel prizes, let's try an example where we'd like to get nobel laureates with some additional info about them. Let's begin with the default view by following the same steps as previously, either accessing the example directly[9] or by following the steps below:

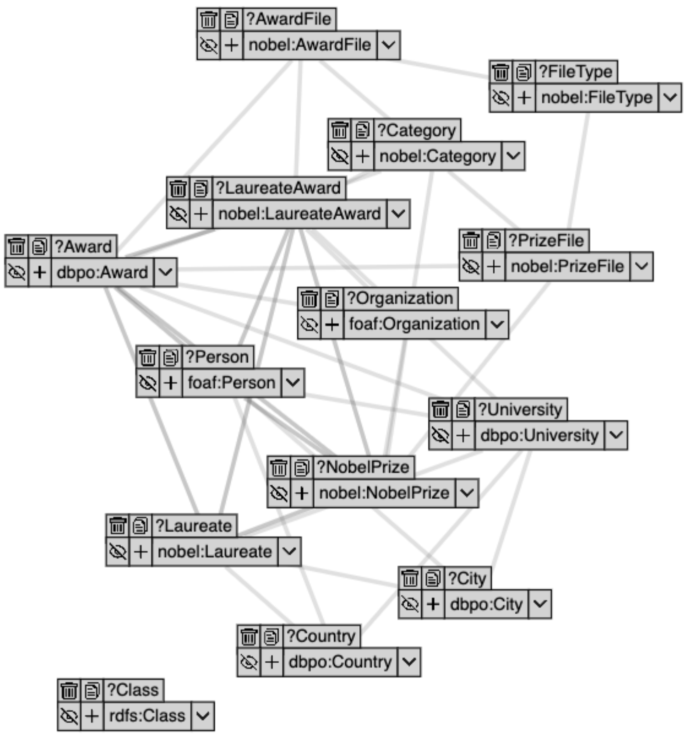
1. Click *File* → *New*
2. Create a New project with the following configuration:

   Data Schema:

   `https://jaresan.github.io/simplod/example.ttl`

   Endpoint: `https://data.nobelprize.org/store/sparql`
3. Click Create

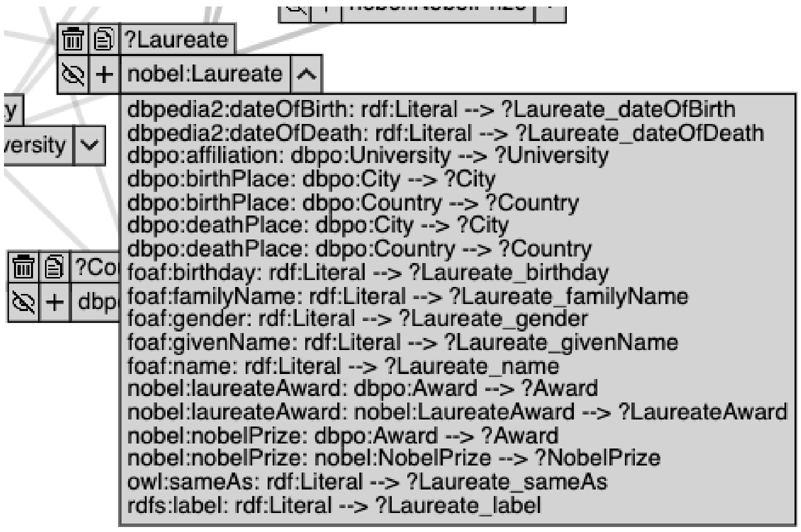
Figure 5.55: Nobel prize example default view



[9]`https://jaresan.github.io/simplod/demo.html?schemaURL=https://jaresan.github.io/simplod/example.ttl&endpointURL=https://data.nobelprize.org/store/sparql`

Since we are interested in nobel prize laureates, we can click the **Laureate** entity to see what relationships there are:

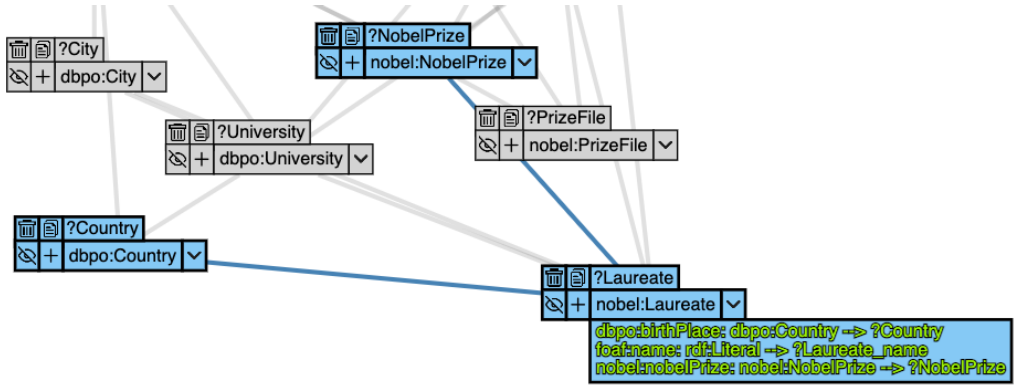
Figure 5.56: Laureate entity properties



```
dbpedia2:dateOfBirth: rdf:Literal --> ?Laureate_dateOfBirth
dbpedia2:dateOfDeath: rdf:Literal --> ?Laureate_dateOfDeath
dbpo:affiliation: dbpo:University --> ?University
dbpo:birthPlace: dbpo:City --> ?City
dbpo:birthPlace: dbpo:Country --> ?Country
dbpo:deathPlace: dbpo:City --> ?City
dbpo:deathPlace: dbpo:Country --> ?Country
foaf:birthday: rdf:Literal --> ?Laureate_birthday
foaf:familyName: rdf:Literal --> ?Laureate_familyName
foaf:gender: rdf:Literal --> ?Laureate_gender
foaf:givenName: rdf:Literal --> ?Laureate_givenName
foaf:name: rdf:Literal --> ?Laureate_name
nobel:laureateAward: dbpo:Award --> ?Award
nobel:laureateAward: nobel:LaureateAward --> ?LaureateAward
nobel:nobelPrize: dbpo:Award --> ?Award
nobel:nobelPrize: nobel:NobelPrize --> ?NobelPrize
owl:sameAs: rdf:Literal --> ?Laureate_sameAs
rdfs:label: rdf:Literal --> ?Laureate_label
```

Let's say we are interested in the laureate's birth country, their name and additional information about the prize they received. If we take a look at the properties, the corresponding ones would be **dbpo:birthPlace (dbpo:Country)**, **foaf:name** and **nobel:nobelPrize (nobel:NobelPrize)** respectively. The selection would look as follows:

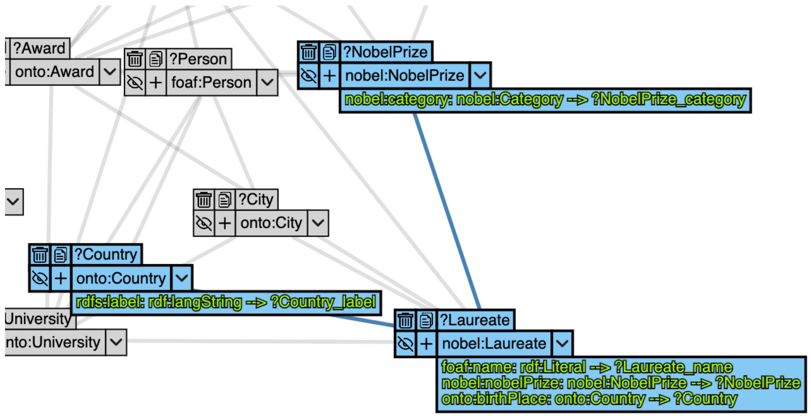
Figure 5.57: Laureate properties of interest



```
dbpo:birthPlace: dbpo:Country --> ?Country
foaf:name: rdf:Literal --> ?Laureate_name
nobel:nobelPrize: nobel:NobelPrize --> ?NobelPrize
```

We can notice that the entities for **nobel:NobelPrize**, **dbpo:Country** are highlighted as well, this is because we have selected the properties targeting them. Let's say for the **dbpo:Country** we are interested in its label **rdfs:label**. For **nobel:NobelPrize**, we would like to know for which **nobel:category** it was awarded. After selecting all of this information, the resulting selection would look as follows in the graph:

Figure 5.58: All properties of interest

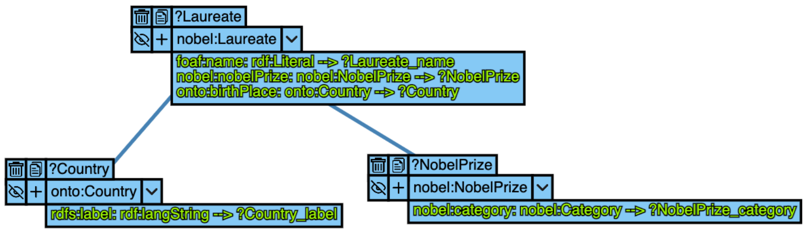

?Award
onto:Award
?Person
foaf:Person
?NobelPrize
nobel:NobelPrize
nobel:category: nobel:Category --> ?NobelPrize_category
?City
onto:City
?Country
onto:Country
rdfs:label: rdf:langString --> ?Country_label
University
nto:University
?Laureate
nobel:Laureate
foaf:name: rdf:Literal --> ?Laureate_name
nobel:nobelPrize: nobel:NobelPrize --> ?NobelPrize
onto:birthPlace: onto:Country --> ?Country

We can clean up the graph by removing data we aren't interested in. We can do this by pressing the icon in the top right of the graph, which will in turn hide all the data that is not requested.

We can also manually delete every single entity via the 🗑 icon, this in turn would make it impossible to get the entities back for this project (as opposed to which just hides the entities, they can be made visible afterwards).

After hiding the data, let's arrange the nodes better by moving them around. After cleaning the graph up a bit, we get something that could look as follows:

Figure 5.59: Cleaned up selection


?Laureate
nobel:Laureate
foaf:name: rdf:Literal --> ?Laureate_name
nobel:nobelPrize: nobel:NobelPrize --> ?NobelPrize
onto:birthPlace: onto:Country --> ?Country
?Country
onto:Country
rdfs:label: rdf:langString --> ?Country_label
?NobelPrize
nobel:NobelPrize
nobel:category: nobel:Category --> ?NobelPrize_category

This is already a valid selection. Using the graph quick action toolbar in the top right, we can hit the ▶ button and execute the query for the following results (the order might differ based on the specific order of selecting the properties):

Figure 5.60: Execute query button

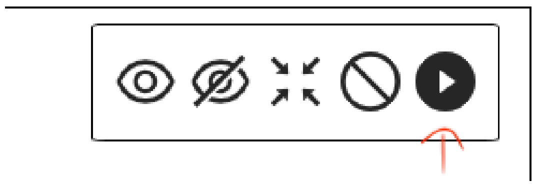


Figure 5.61: Results

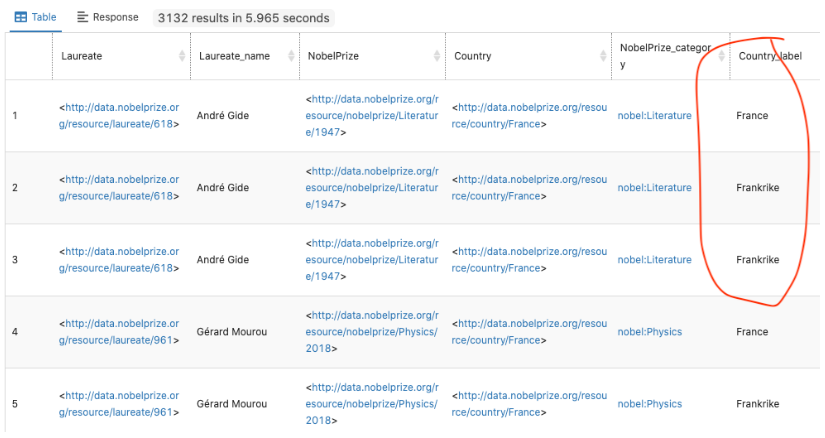


Table Response 3132 results in 5.965 seconds

	Laureate	Laureate_name	NobelPrize	Country	NobelPrize_category	Country_label
1	<http://data.nobelprize.org/resource/laureate/618>	André Gide	<http://data.nobelprize.org/resource/nobelprize/Literature/1947>	<http://data.nobelprize.org/resource/country/France>	nobel:Literature	France
2	<http://data.nobelprize.org/resource/laureate/618>	André Gide	<http://data.nobelprize.org/resource/nobelprize/Literature/1947>	<http://data.nobelprize.org/resource/country/France>	nobel:Literature	Frankrike
3	<http://data.nobelprize.org/resource/laureate/618>	André Gide	<http://data.nobelprize.org/resource/nobelprize/Literature/1947>	<http://data.nobelprize.org/resource/country/France>	nobel:Literature	Frankrike
4	<http://data.nobelprize.org/resource/laureate/961>	Gérard Mourou	<http://data.nobelprize.org/resource/nobelprize/Physics/2018>	<http://data.nobelprize.org/resource/country/France>	nobel:Physics	France
5	<http://data.nobelprize.org/resource/laureate/961>	Gérard Mourou	<http://data.nobelprize.org/resource/nobelprize/Physics/2018>	<http://data.nobelprize.org/resource/country/France>	nobel:Physics	Frankrike

You might notice that some of the rows repeat themselves. This is due to the **Country_label** having entries in multiple languages. If the data is set up properly, properties utilizing multiple languages are of type **rdfs:langString**. To query only for English variants of the country labels, we can change the project properties:

Figure 5.62: Results

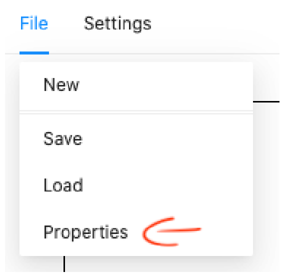
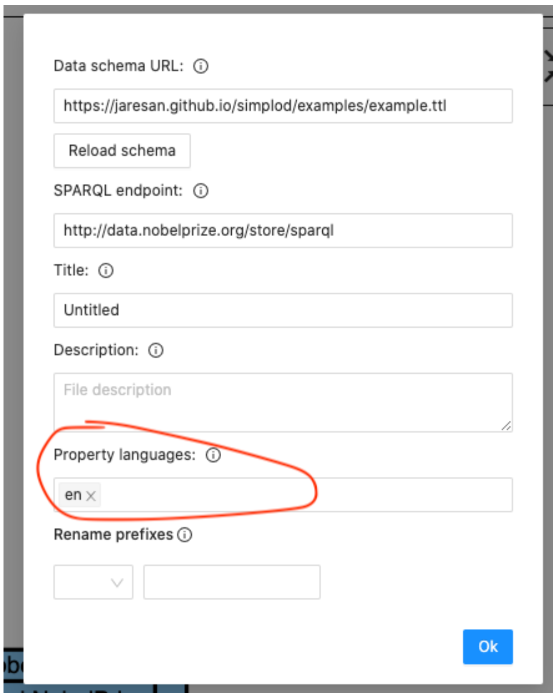


Figure 5.63: Results



Data schema URL:

https://jaresan.github.io/simplod/examples/example.ttl

Reload schema

SPARQL endpoint:

http://data.nobelprize.org/store/sparql

Title:

Untitled

Description:

File description

Property languages:

en ×

Rename prefixes

Ok

Doing this ensures that all of the selected properties will be queried in their English variant if their type is **rdfs:langString**. You can set multiple languages this way, the result set will then contain all of them. With the properties set to query for English only, the results look as follows:

Figure 5.64: Results

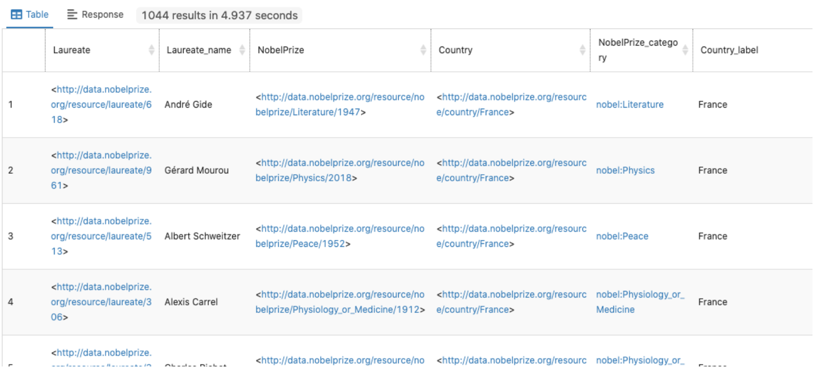
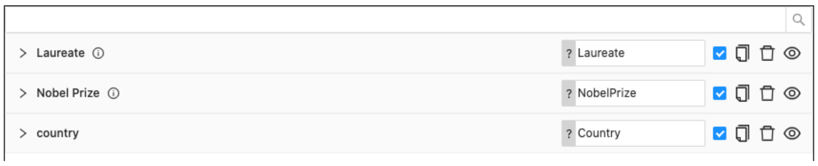


Table | Response | 1044 results in 4.937 seconds

	Laureate	Laureate_name	NobelPrize	Country	NobelPrize_category	Country_label
1	<http://data.nobelprize.org/resource/laureate/618>	André Gide	<http://data.nobelprize.org/resource/nobelprize/Literature/1947>	<http://data.nobelprize.org/resource/country/France>	nobel:Literature	France
2	<http://data.nobelprize.org/resource/laureate/961>	Gérard Mourou	<http://data.nobelprize.org/resource/nobelprize/Physics/2018>	<http://data.nobelprize.org/resource/country/France>	nobel:Physics	France
3	<http://data.nobelprize.org/resource/laureate/513>	Albert Schweitzer	<http://data.nobelprize.org/resource/nobelprize/Peace/1952>	<http://data.nobelprize.org/resource/country/France>	nobel:Peace	France
4	<http://data.nobelprize.org/resource/laureate/306>	Alexis Carrel	<http://data.nobelprize.org/resource/nobelprize/Physiology_or_Medicine/1912>	<http://data.nobelprize.org/resource/country/France>	nobel:Physiology_or_Medicine	France
	<http://data.nobelprize.		<http://data.nobelprize.org/resource/no	<http://data.nobelprize.org/resourc	nobel:Physiology_or_	

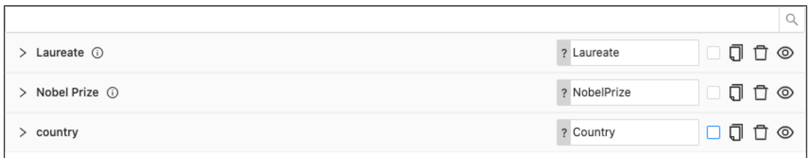
While this query is valid, a person interested in this information might not want the IRIs to be present in the result set, the textual representation provided via labels might be sufficient. To get rid of the IRIs in the result set, we have to deselect the entities via the list view as follows:

Figure 5.65: Selected entities



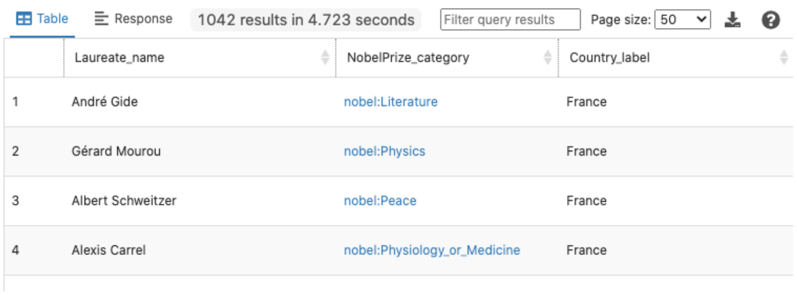
> Laureate	? Laureate	✓
> Nobel Prize	? NobelPrize	✓
> country	? Country	✓

Figure 5.66: Deselected entities



This effectively removes the IRIs from the query and displays only the queried properties (in this case the labels we selected). The result is more concise and shorter:

Figure 5.67: Result set without IRIs



	Laureate_name	NobelPrize_category	Country_label
1	André Gide	nobel:Literature	France
2	Gérard Mourou	nobel:Physics	France
3	Albert Schweitzer	nobel:Peace	France
4	Alexis Carrel	nobel:Physiology_or_Medicine	France

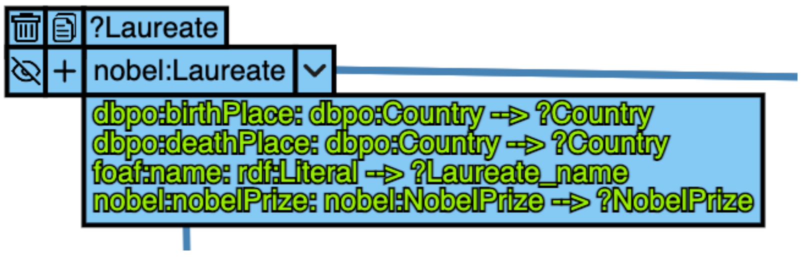
Nobel prize laureates - part 2

We have retrieved information about Nobel prize laureates and about the awards they received. Let's extend the search by querying for the places where the laureates passed away.

Checking Figure 5.56, we can see there are two properties of interest, namely **deathPlace (dbpo:Country)** and **dateOfDeath**. Let's query for **deathPlace (dbpo:Country)** then. Understandably this will result in a data set of **only deceased laureates**, since laureates with no such property will be omitted from the result set, as the property is marked as required, not optional. Marking the property as optional would allow to search for both living and deceased laureates with place of death filled in where applicable.

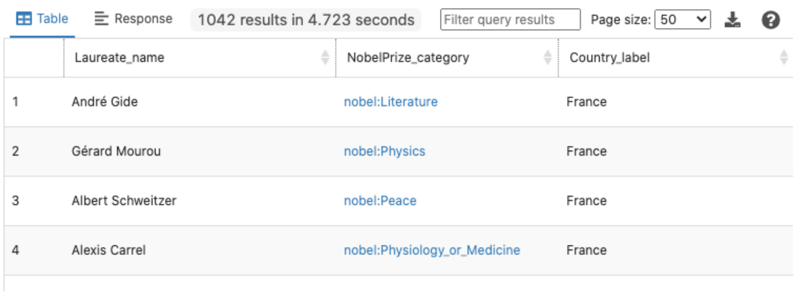
Updated selection and the results with death place under **Country_label** would look as follows:

Figure 5.68: Selection with place of death



?Laureate
nobel:Laureate
dbpo:birthPlace: dbpo:Country --> ?Country
dbpo:deathPlace: dbpo:Country --> ?Country
foaf:name: rdf:Literal --> ?Laureate_name
nobel:nobelPrize: nobel:NobelPrize --> ?NobelPrize

Figure 5.69: Results with place of death



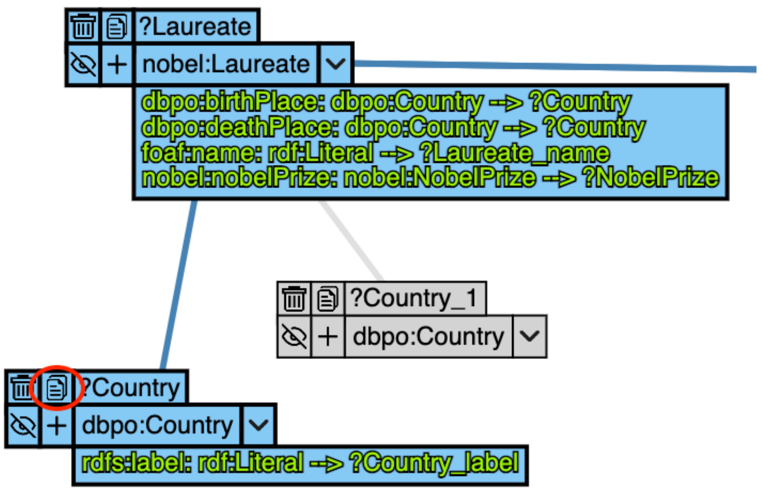
1042 results in 4.723 seconds

	Laureate_name	NobelPrize_category	Country_label
1	André Gide	nobel:Literature	France
2	Gérard Mourou	nobel:Physics	France
3	Albert Schweitzer	nobel:Peace	France
4	Alexis Carrel	nobel:Physiology_or_Medicine	France

You might notice the new result didn't change from Figure 5.67. This is because we are querying for a **single country**. Our query actually translates to **find laureates who were born and died in the same country** due to the links/edges being pointed to the same **Country** entity. While this is not an invalid query and can have its uses, it is not what we are looking for. This is where we need to use the 🗐 icon on the **Country** node in the graph (or use the same on in the list view) and create a separate entity instance for **Country** to introduce a distinction between the death place and birth place.

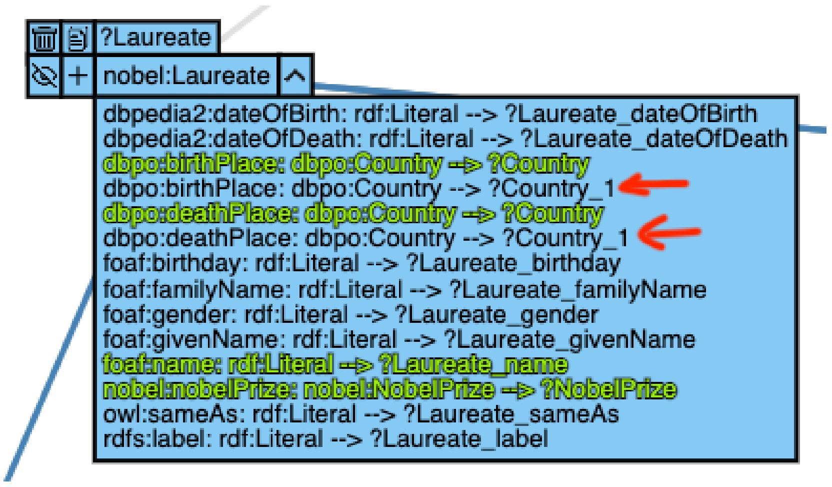
Clicking the 🗐 icon on the **Country** entity, we get the following:

Figure 5.70: Selection with cloned Country entity


?Laureate
nobel:Laureate
dbpo:birthPlace: dbpo:Country --> ?Country
dbpo:deathPlace: dbpo:Country --> ?Country
foaf:name: rdf:Literal --> ?Laureate_name
nobel:nobelPrize: nobel:NobelPrize --> ?NobelPrize
?Country_1
dbpo:Country
?Country
dbpo:Country
rdfs:label: rdf:Literal --> ?Country_label

By copying the node, the **Laureate** entity has new properties added that target the new **Country** entity:

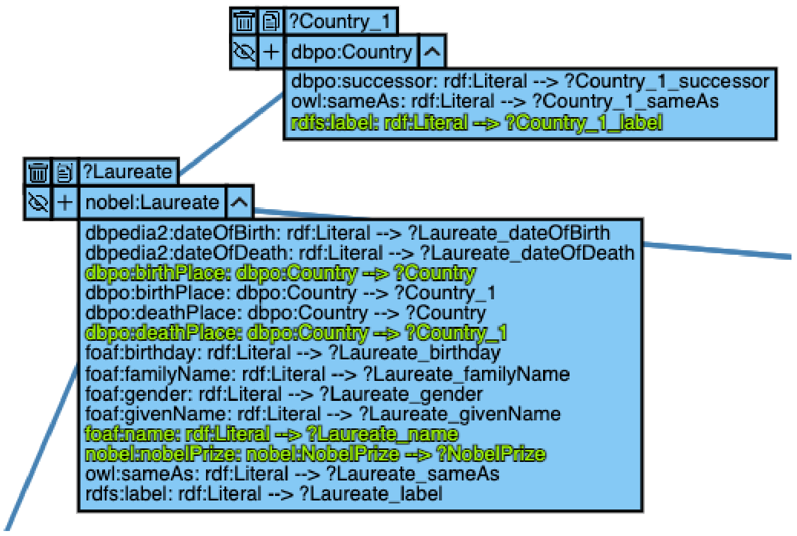
Figure 5.71: Newly listed properties


?Laureate
nobel:Laureate

```
dbpedia2:dateOfBirth: rdf:Literal --> ?Laureate_dateOfBirth
dbpedia2:dateOfDeath: rdf:Literal --> ?Laureate_dateOfDeath
dbpo:birthPlace: dbpo:Country --> ?Country
dbpo:birthPlace: dbpo:Country --> ?Country_1
dbpo:deathPlace: dbpo:Country --> ?Country
dbpo:deathPlace: dbpo:Country --> ?Country_1
foaf:birthday: rdf:Literal --> ?Laureate_birthday
foaf:familyName: rdf:Literal --> ?Laureate_familyName
foaf:gender: rdf:Literal --> ?Laureate_gender
foaf:givenName: rdf:Literal --> ?Laureate_givenName
foaf:name: rdf:Literal --> ?Laureate_name
nobel:nobelPrize: nobel:NobelPrize --> ?NobelPrize
owl:sameAs: rdf:Literal --> ?Laureate_sameAs
rdfs:label: rdf:Literal --> ?Laureate_label
```

Next we just have to remove the old **deathPlace** and use the new one:

Figure 5.72: Proper selection


?Country_1
dbpo:Country

```
dbpo:successor: rdf:Literal --> ?Country_1_successor
owl:sameAs: rdf:Literal --> ?Country_1_sameAs
rdfs:label: rdf:Literal --> ?Country_1_label
```

?Laureate
nobel:Laureate

```
dbpedia2:dateOfBirth: rdf:Literal --> ?Laureate_dateOfBirth
dbpedia2:dateOfDeath: rdf:Literal --> ?Laureate_dateOfDeath
dbpo:birthPlace: dbpo:Country --> ?Country
dbpo:birthPlace: dbpo:Country --> ?Country_1
dbpo:deathPlace: dbpo:Country --> ?Country
dbpo:deathPlace: dbpo:Country --> ?Country_1
foaf:birthday: rdf:Literal --> ?Laureate_birthday
foaf:familyName: rdf:Literal --> ?Laureate_familyName
foaf:gender: rdf:Literal --> ?Laureate_gender
foaf:givenName: rdf:Literal --> ?Laureate_givenName
foaf:name: rdf:Literal --> ?Laureate_name
nobel:nobelPrize: nobel:NobelPrize --> ?NobelPrize
owl:sameAs: rdf:Literal --> ?Laureate_sameAs
rdfs:label: rdf:Literal --> ?Laureate_label
```

We again pick **label** on the newly copied **Country_1** and remove the IRI specification by deselecting the checkbox in the list view for **Country_1**. Executing the query via ⦿ now yields results even if the laureate wasn't born and died in the same country:

Figure 5.73: Proper death place results

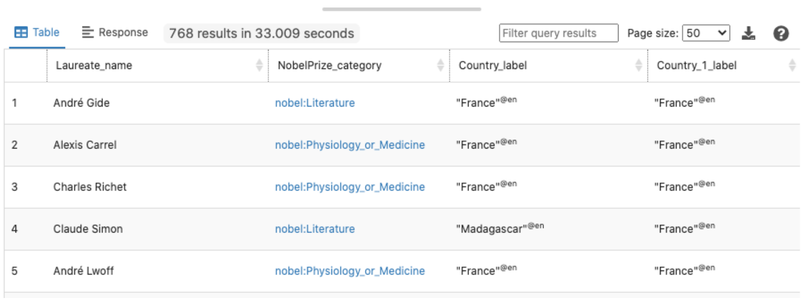
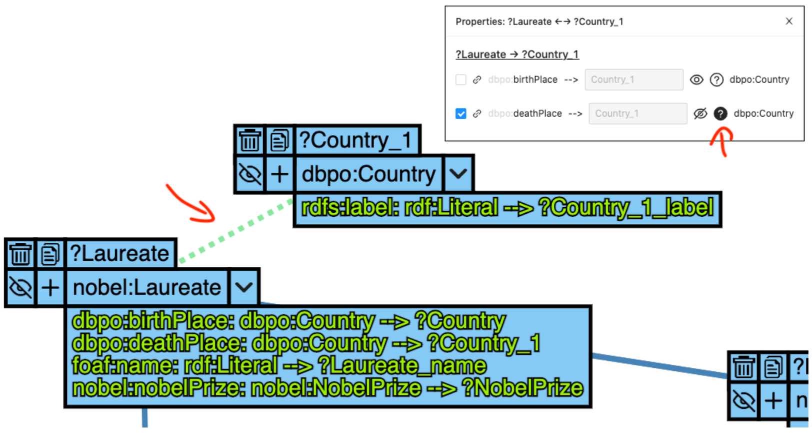


Table | Response | 768 results in 33.009 seconds | Filter query results | Page size: 50

	Laureate_name	NobelPrize_category	Country_label	Country_1_label
1	André Gide	nobel:Literature	"France"@en	"France"@en
2	Alexis Carrel	nobel:Physiology_or_Medicine	"France"@en	"France"@en
3	Charles Richet	nobel:Physiology_or_Medicine	"France"@en	"France"@en
4	Claude Simon	nobel:Literature	"Madagascar"@en	"France"@en
5	André Lwoff	nobel:Physiology_or_Medicine	"France"@en	"France"@en

If we were looking for all laureates (living and dead) and just adding the information of their death place if it exists, we could mark the **deathPlace** property as optional via the list view or clicking the edge:

Figure 5.74: Death place optional



Properties: ?Laureate ←→ ?Country_1
?Laureate → ?Country_1
dbpo:birthPlace -->
Country_1
dbpo:Country
dbpo:deathPlace -->
Country_1
dbpo:Country
?Country_1
dbpo:Country
rdfs:label: rdf:Literal --> ?Country_1_label
?Laureate
nobel:Laureate
dbpo:birthPlace: dbpo:Country --> ?Country
dbpo:deathPlace: dbpo:Country --> ?Country_1
foaf:name: rdf:Literal --> ?Laureate_name
nobel:nobelPrize: nobel:NobelPrize --> ?NobelPrize

Which in turn returns all laureates, living and dead, with the country of their death if it's specified:

Figure 5.75: Results with death place optional

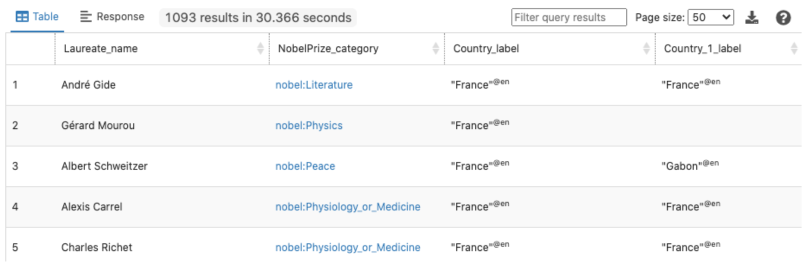


Table | Response | 1093 results in 30.366 seconds | Filter query results | Page size: 50

	Laureate_name	NobelPrize_category	Country_label	Country_1_label
1	André Gide	nobel:Literature	"France"@en	"France"@en
2	Gérard Mourou	nobel:Physics	"France"@en	
3	Albert Schweitzer	nobel:Peace	"France"@en	"Gabon"@en
4	Alexis Carrel	nobel:Physiology_or_Medicine	"France"@en	"France"@en
5	Charles Richet	nobel:Physiology_or_Medicine	"France"@en	"France"@en

Another nice example would be to query only for **living laureates**. However, such an example would require the ability to constrain the values of properties, which is not a feature implemented in the graph tool. However, users can also directly edit the SPARQL query, should they want to tweak it.

German books - Save & Load example

Let us finish with an example that will show the loading and saving capabilities of the application. This example is based on B3Kat cataloguing platform[10] which we will use to fetch data about books and their relevant information.

Unlike in the previous examples, here we will be starting from an already curated example project file. Files like these can be created by appointed users to separate bigger data sets into smaller, better manageable chunks. The data schema used in this example has been curated directly via the application from a *ttl* file available at GitHub[11].

[10] `https://www.kobv.de/services/katalog/b3kat/`

[11] `https://raw.githubusercontent.com/jaresan/simplod/master/public/german_books.ttl`

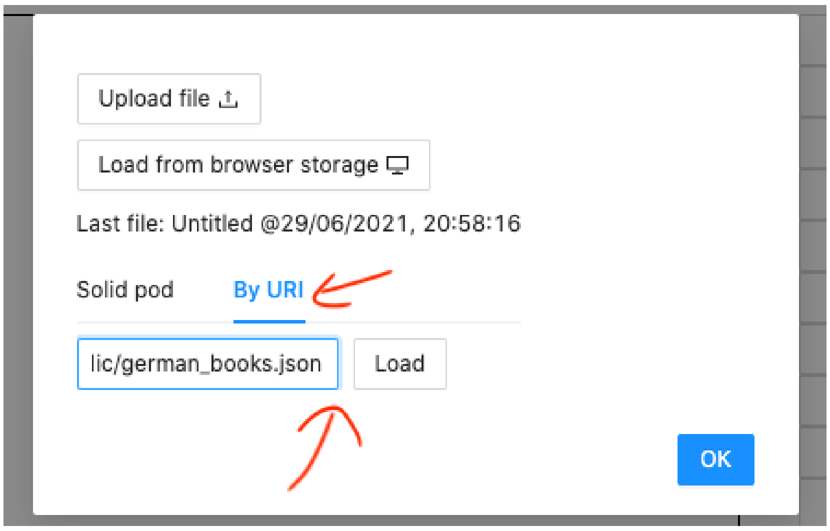
We will also be utilizing a Solid pod for data persistence. Before proceeding further, it is important we follow the steps outlined in subsection 5.1.1 and have the Solid pod set up with the necessary application permissions up correctly.

Let's open the application Demo - `https://jaresan.github.io/simplod/build/index.html`.

To start with the example, we will first load the appropriate project file by navigating to *File* → *Load* → *By URI*.

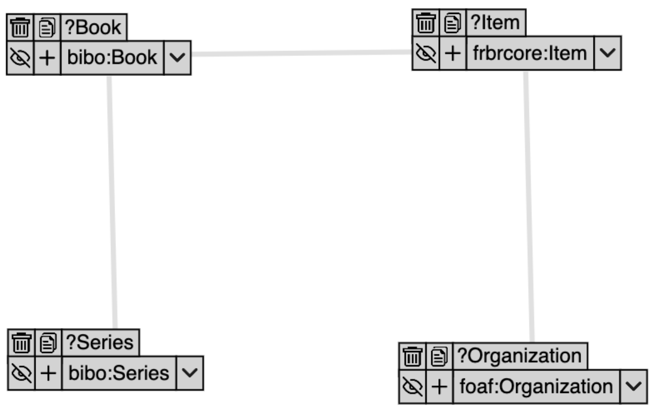
In the input field we put `https://jaresan.github.io/simplod/examples/german_books.json` and press **Load**.

Figure 5.76: Load by URI


Upload file
Load from browser storage
Last file: Untitled @29/06/2021, 20:58:16
Solid pod
By URI
lic/german_books.json
Load
OK

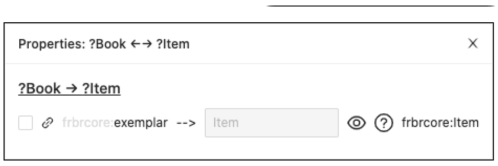
Upon loading the project file, we can see the default view for the curated book data set:

Figure 5.77: Graph loaded



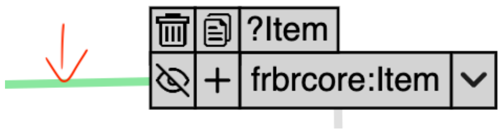
By clicking on the edges between the nodes, we can inspect the relationships they represent:

Figure 5.78: Edge descriptions


Properties: ?Book ←→ ?Item

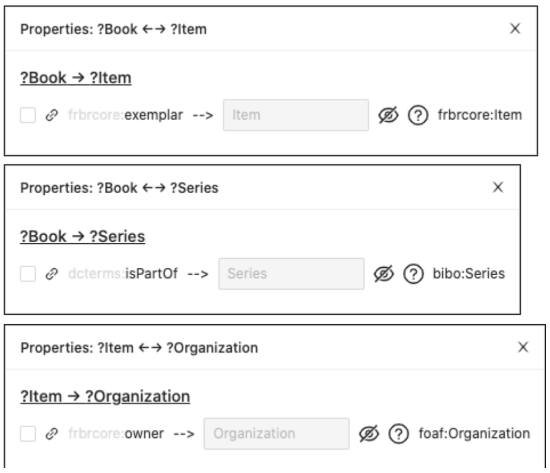
?Book → ?Item

frbrcore:exemplar --> Item frbrcore:Item



By inspecting all three edges presented, we will get the following information for the existing relations in the data schema:

Figure 5.79: Edge descriptions


Properties: ?Book ←→ ?Item

?Book → ?Item

frbrcore:exemplar --> Item frbrcore:Item

Properties: ?Book ←→ ?Series

?Book → ?Series

dcterms:isPartOf --> Series bibo:Series

Properties: ?Item ←→ ?Organization

?Item → ?Organization

frbrcore:owner --> Organization foaf:Organization

From this, we can gather that a book can have an exemplar item which in turn is owned by an organization. A book can also be a part of a series.

Let's find books with their respective series, should they be a part of one. If we are looking for books, we might also be interested where we could borrow them from. We are then going to query their respective exemplar **Items**, and for the items, we will select the **Organizations** that represent the owner.

As usual, we want to get the labels or titles for each item. Organizations in this case also contain the property **homepage** which could be useful as a reference to the owner as well. With all of this data in mind, the selection would look as follows:

- Book

  bibo:isbn - ISBN of the book

  dce:title - Title of the book

  dcterms:partOf, Optional - Book series

  frbrcore:exemplar - Exemplar of the book

- Series

  bibo:shortTitle - Title of the series

- Item

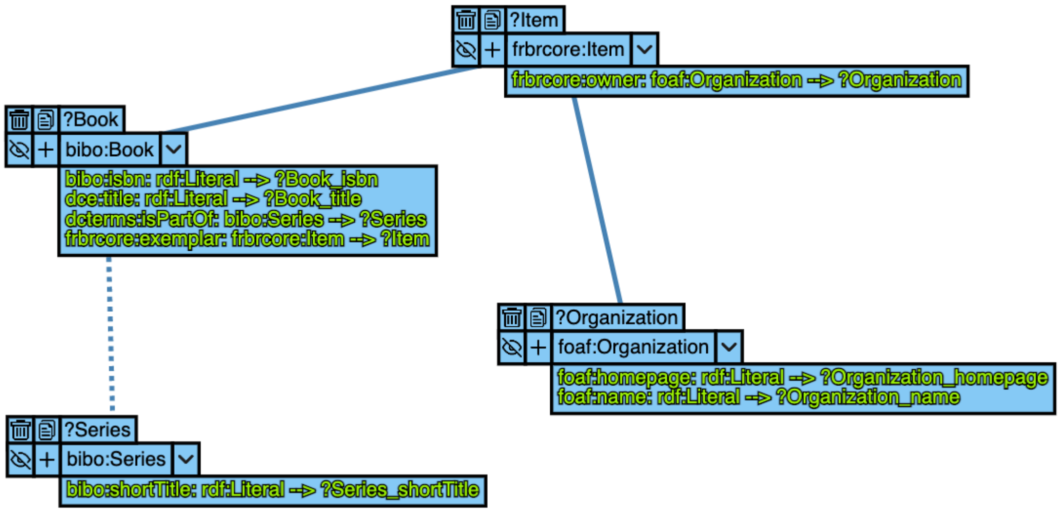
  frbrcore:owner - Organizational owner of the exemplar

- Organization

  foaf:homepage - Homepage of the organization

  foaf:name - Name of the organization

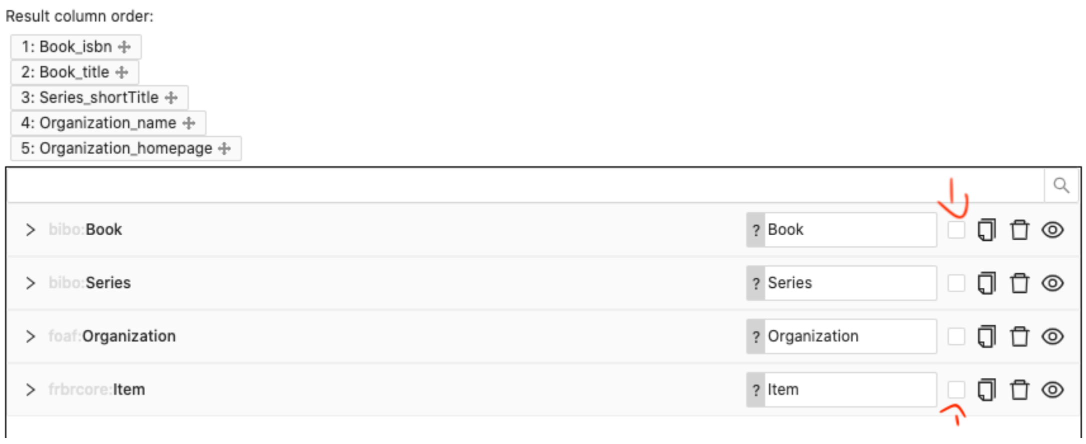
Figure 5.80: Graph selection



?Item
frbrcore:Item
frbrcore:owner: foaf:Organization --> ?Organization
?Book
bibo:Book
bibo:isbn: rdf:Literal --> ?Book_isbn
dce:title: rdf:Literal --> ?Book_title
dcterms:isPartOf: bibo:Series --> ?Series
frbrcore:exemplar: frbrcore:Item --> ?Item
?Organization
foaf:Organization
foaf:homepage: rdf:Literal --> ?Organization_homepage
foaf:name: rdf:Literal --> ?Organization_name
?Series
bibo:Series
bibo:shortTitle: rdf:Literal --> ?Series_shortTitle

Notice the dashed edge, this means the relationship between book and its series is optional, meaning we will query for all books and return their respective series, if existing. Not marking this edge as optional, we would only query for books that exist in a series. In the list selection, we can deselect the entities themselves to omit the IRIs from displaying in the result set:

Figure 5.81: List selection



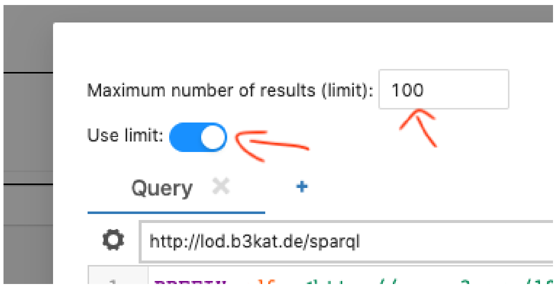
Result column order:

1: Book_isbn
2: Book_title
3: Series_shortTitle
4: Organization_name
5: Organization_homepage

bibo:Book	? Book
bibo:Series	? Series
foaf:Organization	? Organization
frbrcore:Item	? Item

We can run the query via the ▶ icon. For the demonstration purposes of this example, it would be a good idea to limit the maximum number of results we can retrieve to 100. The data set provided by B3Kat spans over 25 million titles and querying across them all might take a significant amount of time.

Figure 5.82: Query limit

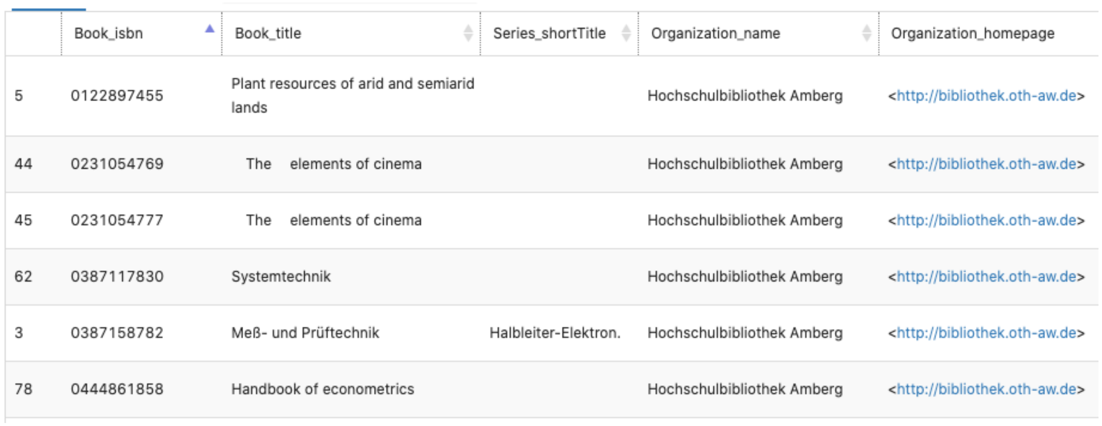

Maximum number of results (limit): 100

Use limit:

http://lod.b3kat.de/sparql

After changing the limit and running the query again by pressing the ▶, we can get the following results:

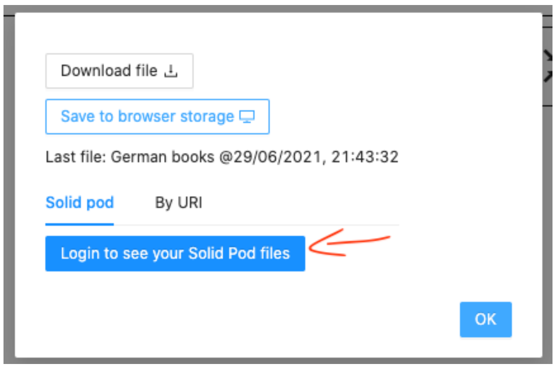
Figure 5.83: Results



	Book_isbn	Book_title	Series_shortTitle	Organization_name	Organization_homepage
5	0122897455	Plant resources of arid and semiarid lands		Hochschulbibliothek Amberg	<http://bibliothek.oth-aw.de>
44	0231054769	The elements of cinema		Hochschulbibliothek Amberg	<http://bibliothek.oth-aw.de>
45	0231054777	The elements of cinema		Hochschulbibliothek Amberg	<http://bibliothek.oth-aw.de>
62	0387117830	Systemtechnik		Hochschulbibliothek Amberg	<http://bibliothek.oth-aw.de>
3	0387158782	Meß- und Prüftechnik	Halbleiter-Elektron.	Hochschulbibliothek Amberg	<http://bibliothek.oth-aw.de>
78	0444861858	Handbook of econometrics		Hochschulbibliothek Amberg	<http://bibliothek.oth-aw.de>

What if we want to save this result to our Solid pod? We just go to *File* → *Save*. If we are logged in, we already see our Solid pod files. If we are not logged in, we can log in either directly through the button in this menu, or through the top right avatar menu.

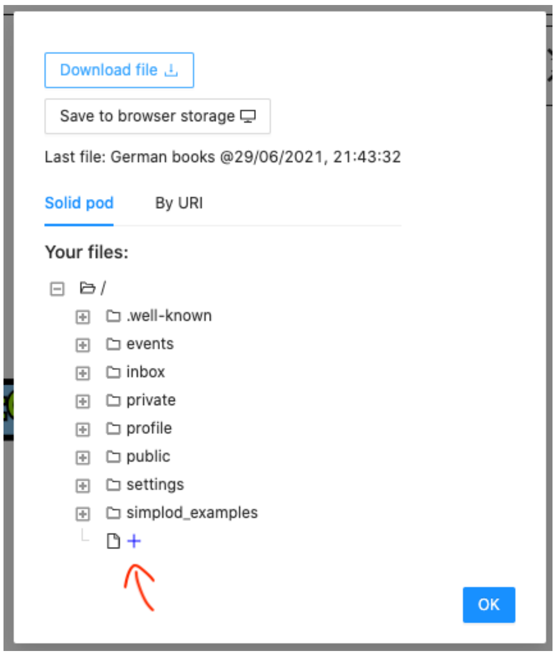
Figure 5.84: Save menu



Last file: German books @29/06/2021, 21:43:32

After successfully logging in, we can see our the list of our Solid pod files. We can list through the folders and in each one click the ✚ icon to save the file in that location. Let's go with the root folder and click the ✚ icon.

Figure 5.85: Solid pod files



Download file

Save to browser storage

Last file: German books @29/06/2021, 21:43:32

Solid pod   By URI

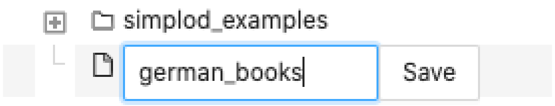
Your files:

```
⊟ /
  ⊞ .well-known
  ⊞ events
  ⊞ inbox
  ⊞ private
  ⊞ profile
  ⊞ public
  ⊞ settings
  ⊞ simplod_examples
  └ +
```

OK

After clicking the ✚ icon, we just have to pick a file name. Let's type in *german_books* and click on save or hit enter.

Figure 5.86: New filename

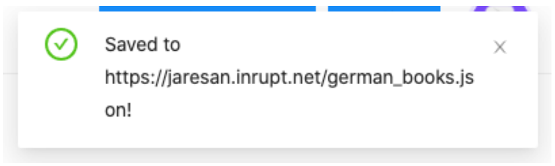


⊞ simplod_examples

german_books   Save

If the file got saved correctly, we are greeted with a notification confirming the new save location:

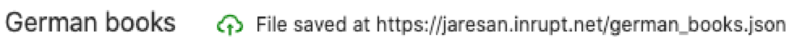
Figure 5.87: File saved notification



Saved to https://jaresan.inrupt.net/german_books.json!

We can also see the new file location in the status bar. Hitting **ctrl+s** now saves the changes to the new remote location.

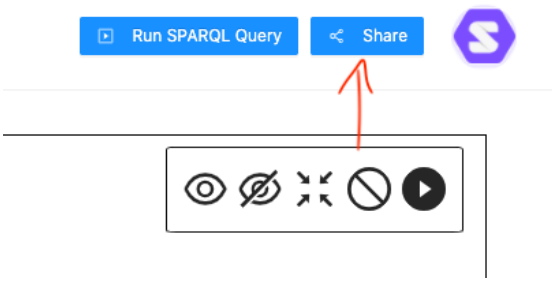
Figure 5.88: Status bar after remote save



German books   File saved at https://jaresan.inrupt.net/german_books.json

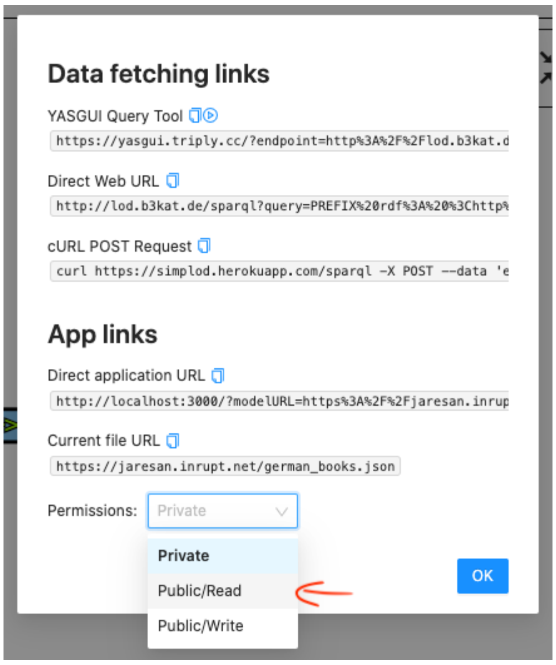
Finally, we might want to share this project file among other users. To do that, first we need to set its permission appropriately from the **Share** menu in the top-right.

Figure 5.89: Permission drop-down



From the drop-down list, we can select **Public/read**, so that every user will be able to read this project file, but only we, as the owner, will be able to edit it.

Figure 5.90: Permission drop-down



Data fetching links

YASGUI Query Tool

```
https://yasgui.triply.cc/?endpoint=http%3A%2F%2Flod.b3kat.d
```

Direct Web URL

```
http://lod.b3kat.de/sparql?query=PREFIX%20rdf%3A%20%3Chttp%
```

cURL POST Request

```
curl https://simplod.herokuapp.com/sparql -X POST --data 'e
```

App links

Direct application URL

```
http://localhost:3000/?modelURL=https%3A%2F%2Fjaresan.inrup
```

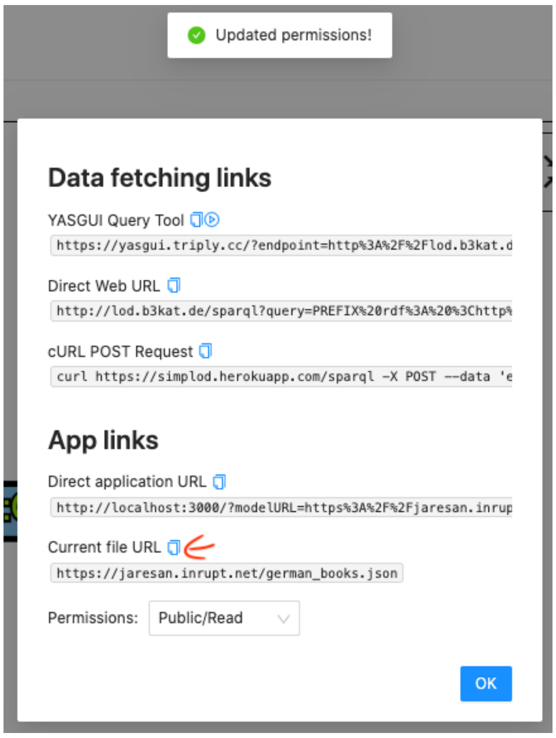
Current file URL

```
https://jaresan.inrupt.net/german_books.json
```

Permissions: Private

If everything goes correctly, the action is confirmed by a notification and we can proceed to copy the file URL via the 🗐 icon and share it among other users.

Figure 5.91: Permissions changed successfully


Updated permissions!

Data fetching links

YASGUI Query Tool
https://yasgui.triply.cc/?endpoint=http%3A%2F%2Flod.b3kat.d

Direct Web URL
http://lod.b3kat.de/sparql?query=PREFIX%20rdf%3A%20%3Chttp%

cURL POST Request
curl https://simplod.herokuapp.com/sparql -X POST --data 'e

App links

Direct application URL
http://localhost:3000/?modelURL=https%3A%2F%2Fjaresan.inrup

Current file URL
https://jaresan.inrupt.net/german_books.json

Permissions: Public/Read

OK

The copied URL (in this case `https://jaresan.inrupt.net/german_books.json`) leads directly to the model file in our Solid pod. Same way as in the first step of this example this URL can be directly loaded in the application by *File* → *Load* → *By URI*.

5.2 Administrator documentation

In the first part this section describes the steps to take to deploy the application. The second part focuses on providing the application with initialization inputs to change what data the application displays when the users enter from a specific source.

5.2.1 Prerequisites

The minimum required configuration to be able to follow the deployment and development steps are:

1. NodeJS[12]

   Version $\geq$ 10.0.

2. Npm[13].

   This documentation is written for npm, but other package managers, e.g. yarn[14] can be used as well.

[12]`https://nodejs.org/en/`
[13]`https://www.npmjs.com/`
[14]`https://yarnpkg.com/`

5.2.2 Data schema creation

As mentioned in section 2.5, one of the required inputs for the application is a data schema. One of the ways to create such a schema is by providing an endpoint containing the data to a LinkedPipes ETL Klímek and Škoda [2017] pipeline created by the supervisor of this thesis.

This subsection describes how to create a data schema from an endpoint utilizing the demo instance[15] and the pipeline specification[16].

Preparing the endpoint specification

Before running the pipeline, we have to create an input file specifying the endpoints we would like the pipeline to use. For this purpose we can edit the template file on GitHub[17] with our own endpoint by replacing `http://vocabularies.unesco.org/sparql` with the URL of the SPARQL endpoint of our choosing.

We can also provide more endpoints at once, as shown in a template file on GitHub[18]. However, in such a case case, the pipeline would aggregate all the results into one .ttl file, which might not be desirable. To prevent this, we can always update the single endpoint file and run the pipeline again.

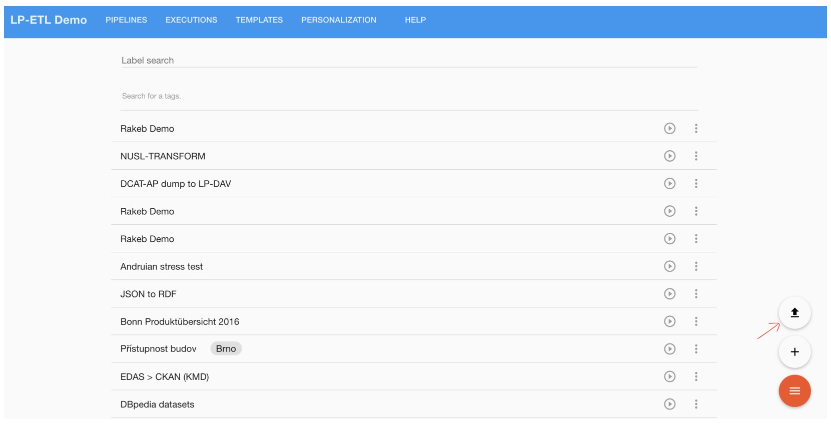
We then make the file remotely accessible and in the steps outlined below can provide the URL to the pipeline. For the sake of this example, let's consider the file hosted on GitHub[19].

Running the pipeline

To run the pipeline with our endpoint specification, we can follow these steps:

1. Access `https://demo.etl.linkedpipes.com/#/pipelines` and click on the upload button:

Figure 5.92: Upload button



Rakeb Demo
NUSL-TRANSFORM
DCAT-AP dump to LP-DAV
Rakeb Demo
Rakeb Demo
Andruian stress test
JSON to RDF
Bonn Produktübersicht 2016
Přístupnost budov Brno
EDAS > CKAN (KMD)
DBpedia datasets

[15] `https://demo.etl.linkedpipes.com/#/pipelines`

[16] `https://raw.githubusercontent.com/jaresan/lod-cloud/master/simplod_data_schema_pipeline.jsonld`

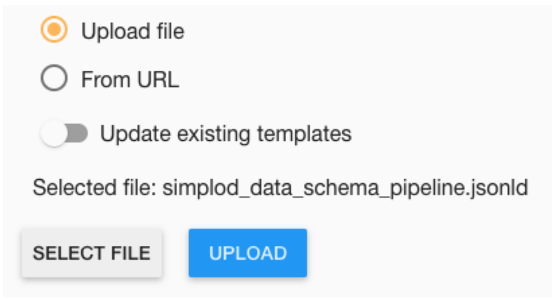
[17] `https://github.com/jaresan/lod-cloud/blob/master/endpoint.ttl`

[18] `https://github.com/jaresan/lod-cloud/blob/master/LODCloud_SPARQL_Endpoints.ttl`

[19] `https://raw.githubusercontent.com/jaresan/lod-cloud/master/endpoint.ttl`

2. Provide the .jsonld file specification[20] and click "Upload":

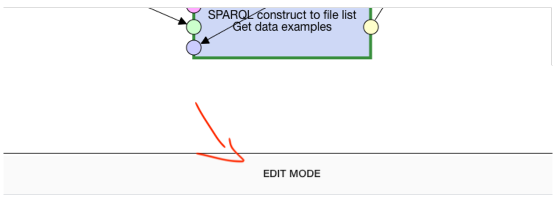
Figure 5.93: Upload detail



Selected file: simplod_data_schema_pipeline.jsonld

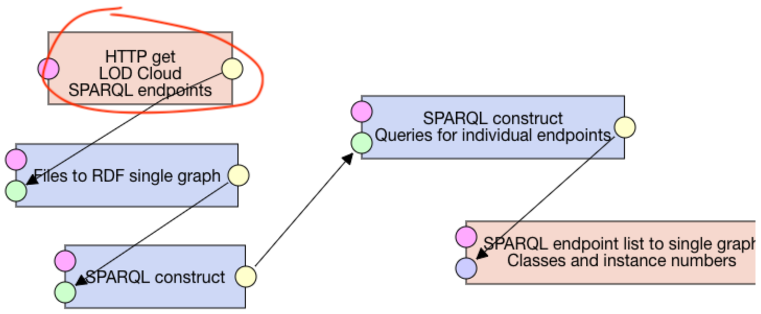
3. The graphical representation of the pipeline is shown on successful upload. If the graph is not in edit mode, click on the "edit mode" button in the bottom right:

Figure 5.94: Pipeline graph



4. Double click on the first node in the graph marked as "HTTP get" to enter its edit mode:

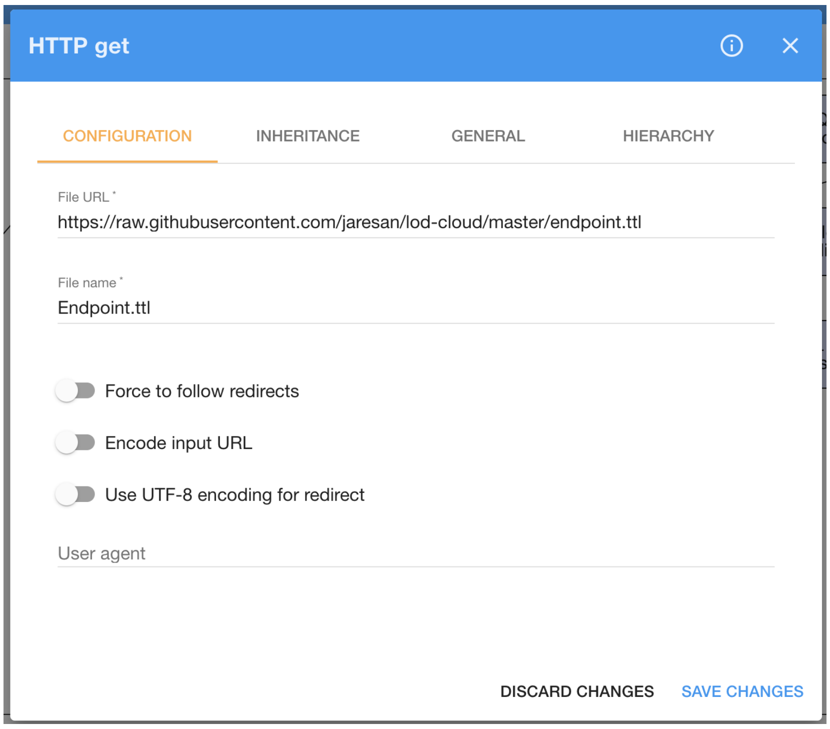
Figure 5.95: Pipeline start node


HTTP get
LOD Cloud
SPARQL endpoints
Files to RDF single graph
SPARQL construct
SPARQL construct
Queries for individual endpoints
SPARQL endpoint list to single graph
Classes and instance numbers

5. Change the File URL to point to your .ttl file with the endpoint specification. The filename is not important for our use case, but make sure to specify .ttl as suffix:

[20]`https://raw.githubusercontent.com/jaresan/lod-cloud/master/simplod_data_schema_pipeline.jsonld`

Figure 5.96: Start node options


HTTP get

CONFIGURATION | INHERITANCE | GENERAL | HIERARCHY

File URL *
https://raw.githubusercontent.com/jaresan/lod-cloud/master/endpoint.ttl

File name *
Endpoint.ttl

Force to follow redirects

Encode input URL

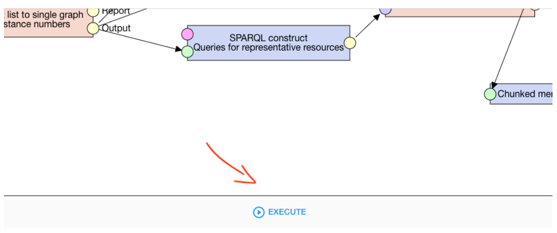
Use UTF-8 encoding for redirect

User agent

DISCARD CHANGES SAVE CHANGES

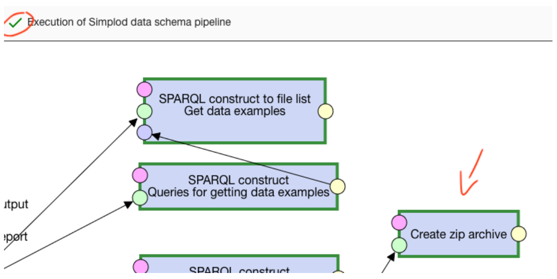
6. After the data is properly changed, click on the ”execute” button at the bottom of the graph:

Figure 5.97: Execute button



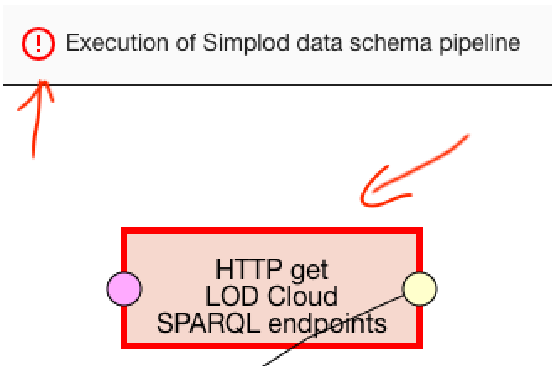
7. Wait for the execution of the pipeline to finish. The success of the run is symbolized by the status bar or by the final node’s edge being green:

Figure 5.98: Execute button


Execution of Simplod data schema pipeline
SPARQL construct to file list
Get data examples
SPARQL construct
Queries for getting data examples
utput
port
Create zip archive
SPARQL construct

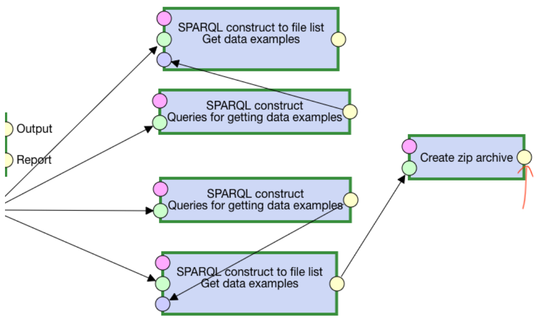
8. The process can also fail for various reasons, this fact is similarly by an icon and by the edge of the failing node being red. This work does not cover troubleshooting for failing pipeline executions.

Figure 5.99: Execute button



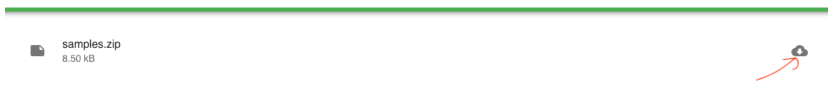
9. Click on the output (yellow) socket on the last node of the graph:

Figure 5.100: Results location


SPARQL construct to file list
Get data examples
SPARQL construct
Queries for getting data examples
Output
Report
Create zip archive
SPARQL construct
Queries for getting data examples
SPARQL construct to file list
Get data examples

10. Click on the download icon to get an archive with the results:

Figure 5.101: Download results



The resulting .ttl can then be directly loaded in the application. The .ttl file generated by this example can be found on GitHub[21].

5.2.3 SPARQL proxy

As described in section 4.3, the application might also require a proxy to be used in the case of failing requests due to various reasons, mainly due to CORS and Same-Origin policy issues. To prevent this, this work is also submitted with an Express[22] application, that can be hosted and used as the proxy.

- Clone the repository

  Clone the repository at `https://github.com/jaresan/sparql-proxy/`.

[21]`https://github.com/jaresan/lod-cloud/blob/master/unesco.ttl`
[22]`https://expressjs.com/`

- Deploy the proxy Express application

  Follow basic deployment steps.

  The steps to deploy an Express application are not a part of this work since there are various hosting services with their specific guides each.

- Change the application proxy path

  To use the proxy, you have to change `src/@@constants/api.js`:

```
const root = 'YOUR_PROXY_PATH_HERE';
const useProxy = true;
```

This application is submitted using a proxy running at `https://simplod.herokuapp.com/`.

Not using a proxy can result in failing to fetch human readable labels for the entities in the list view and potentially failing query execution due to the endpoint not being set up correctly.

5.2.4 Deployment

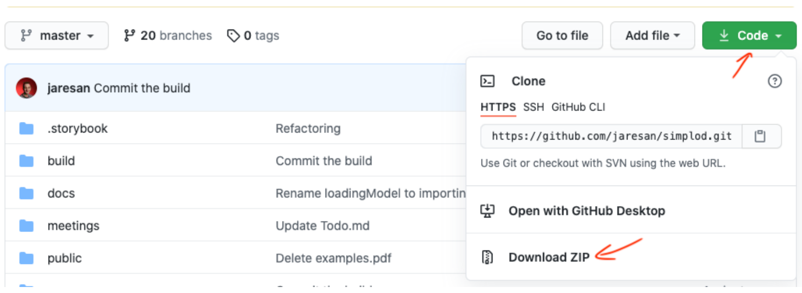
There are only few steps the administrator has to take to deploy the application. The steps are written utilizing npm[23], but the same pattern can be followed when using other package managers:

Direct build download

For convenience, the build files[24] are committed as well in the repository. The deployment steps in that case are as follows:

1. Download the repository[25] as zip

Figure 5.102: Download as zip



2. Upload the build file to your hosting and specify `index.html` as the entry-point

[23]`https://www.npmjs.com/`
[24]`https://github.com/jaresan/simplod/tree/master/build`
[25]`https://github.com/jaresan/simplod/`

Project deployment

Since this application has been developed with Creact React App[26], the administrators can also follow the steps outlined on the React deployment page[27] directly, hosting via `npm start`.

1. Clone the repository

   First, clone the repository at `https://github.com/jaresan/simplod`.

2. Install the dependencies

   From inside the repository, run `npm install`.

3. Build the repository

   Build the production version of the repository by running `npm run build` inside the repository.

4. Publish build and expose `build/index.html`

   Upload the build directory to a hosting service and make index.html accessible.

This work also includes a heroku[28] postbuild hook[29], meaning all new pushed changes to heroku are automatically built.

In conclusion, as terminal commands, the steps could be summed up as follows:

```
# ... cd to the location you want to save this project

git clone https://github.com/jaresan/simplod your_project_name
cd your_project_name
npm install
npm run build

# upload ./build to a hosting site and make index.html accessible
```

5.2.5 Application parameters

When deployed, the administrator is able to change the data with which the application opens by utilizing one of the three available parameters in the application URL.

- `schemaURL=`

  URL of the data schema to load. For example:

  `https://jaresan.github.io/simplod/example.ttl`

[26]https://github.com/facebook/create-react-app

[27]https://create-react-app.dev/docs/deployment/

[28]https://www.heroku.com/

[29]https://devcenter.heroku.com/articles/nodejs-support#customizing-the-build-process

- `endpointURL=`

  SPARQL endpoint to be set in the application.

  For example `https://data.nobelprize.org/store/sparql`.

- `modelURL=`

  URL of a project file to load, acting in the same way as *File → Load → By URI*. This option overrides both `schemaURL=` and `endpointURL=`

  For example: `https://jaresan.inrupt.net/german_books.json`

Example usage:

- index.html path

  `https://jaresan.github.io/simplod/build/index.html`

- Data schema

  `https://jaresan.github.io/simplod/example.ttl`

- Endpoint

  `https://data.nobelprize.org/store/sparql`

- Project

  `https://jaresan.inrupt.net/german_books.json`

The link pointing to the application would have these two added as URL params[30] named **schemaURL** and **endpointURL**.

With the parameters set up, we get a link[31] pointing to the instantiated application with the data schema and endpoint, or a link[32] to the application with project file to be loaded.

5.3 Programmer documentation

This section gives a basic overview of the application for the developers. The first part describes how to set up the development environment locally, following with a brief description of the code structure and examples of new features and how to implement them. Lastly, automatically generated documentation from the code is mentioned.

[30] `https://developer.mozilla.org/en-US/docs/Web/API/URLSearchParams`

[31] `https://jaresan.github.io/simplod/build/index.html?schemaURL=https://jaresan.github.io/simplod/example.ttl&endpointURL=https://data.nobelprize.org/store/sparql`

[32] `https://jaresan.github.io/simplod/build/index.html?modelURL=https://jaresan.inrupt.net/german_books.json`

5.3.1 Prerequisites

The minimum required configuration for developing the application:

1. NodeJS[33]

   Version $\geq$ 10.0.

2. Npm[34].

   This documentation is written for npm, but other package managers, e.g. yarn[35] can be used as well.

5.3.2 Local development setup

- Clone the repository

  First, clone the repository at `https://github.com/jaresan/simplod`.

- Install the dependencies

  From inside the repository, run `npm install`.

- Run local environment

  From inside the repository, run `npm run start:dev`. After running the command, the application will be available on `http://localhost:3000`. If you want to run a version with SSL on, you can use

  `npm run start:dev:https`.

  Run this way, the server listens to changes to the codebase, and reloads the application if any are detected.

- OPTIONAL - Add localhost to your trusted apps

  If you wish to use a Solid Pod while running the application locally, you have to add the hosting address to your Solid Pod. This can be done automatically through signing to the Solid Pod in the upper right corner, or by following the steps mentions in Appendix A

- OPTIONAL - Link SPARQL Proxy

  Add SPARQL Proxy based on subsection 5.2.3.

  This work is submitted with a proxy avaiable on Heroku[36].

5.3.3 Overview

This subsection provides an overview of the application state and code structure.

[33]`https://nodejs.org/en/`
[34]`https://www.npmjs.com/`
[35]`https://yarnpkg.com/`
[36]`https://simplod.herokuapp.com`

Redux state

As mentioned in section 3.4, Redux is used for handling the application state. The whole state is split up into 5 separate sub-states as follows:

```
{
  "solid": {},
  "model": {},
  "settings": {},
  "controls": {},
  "yasgui": {}
}
```

- solid

  Represents the authentication state. Contains user's information and their session.

- model

  Represents all of the data in the application that can be exported. When a user saves/downloads the project file, the file they create is a **direct copy of this sub-state**. Importing a file directly replaces this sub-state.

- settings

  User's specific settings, e.g. language, view layout.

- controls

  Contains arbitrary information used to help render components interactively to the user. For example contains the currently selected edge, which is used to highlight the edge and display the properties it represents.

- yasgui

  YASGUI specific information. Contains the currently parsed query and the YASGUI instance.

Detailed description of the state can be found on GitHub[37].

Folder structure

The source files of the application are split into folders as follows:

- `@@actions`

  More complex actions that can be triggered throughout the application and can trigger state changes.

- `@@app-state`

  State handling functionality for every sub-state of the application and the definition of the main reducer and how changes propagate to the store.

[37] `https://github.com/jaresan/simplod/tree/master/src/%40%40app-state`

- `@@components`

  All React components in the application split further into:

  - `controls`

    Helper interaction components, for example modals, confirm dialogs.
  - `entityList`

    List view, allowing the user to interact with the data set via a list component instead of through the graph.
  - `menu`

    All menu components, e.g. the menu bar, save/load menu, share menu.

- `@@constants`

  Declaration of the constants used throghout the application.

- `@@data`

  Data handling, be it graph calculations, parsint `.ttl` files or the SPARQL query itself.

- `@@graph`

  All of the graph layer, as described in subsection 3.3.1, the graph library used is AntV. The graph folder is split further into two more folders:

  - `wrappers`

    Classes wrapping the interaction with the actual graph elements and reacting to it.
  - `handlers`

    Classes handling and triggering state changes, as opposed to wrappers, these classes don't react to user interactions directly.

- `@@selectors`

  State selectors, used by the components to subscribe to specific subsets of the application state.

More thorough description of all files with generated code documentation by JsDoc[38] can be found on GitHub[39].

5.3.4 Implementation examples

This subsections describes an example of how to implement some features that are not present in the application.

[38] `https://jsdoc.app/`

[39] `https://jaresan.github.io/simplod/documentation`

Hide rest → Delete rest

As mentioned in subsection 5.1.3, the ”Hide rest” hides all entities that are not selected via any means. For this exmaple let’s change this to delete all such entities instead of hiding them.

Checking the source code, we can find the button in the graph component in `@@components/GraphContainer.js`. Following the functionality, we can see that the main logic taking place is implemented in `@@app-state/model/state`’s hideUnselected. To build it in a similar way, we could do the following:

```
export const deleteUnselected = s => {
  const properties = pipe(
    getProperties,
    filter(prop('selected')),
    values
  )(s);
  const toKeep = properties
    .reduce((acc, p) => Object.assign(acc, {
      [p.target]: true,
      [p.source]: true
    }), {});

  const toDelete = keys(
    filter(
      c => !c.selected,
      omit(keys(toKeep), view(classes, s))
  ));

  return toDelete.reduce((acc, id) => deleteClass(id, acc), s);
}
```

If you test this out, you will see that while the entities disappeared from the list view, they remained intact in the graph. This is because the graph items have to be handled separately and deleted directly in the graph. Luckily there’s already a static method `onDeleteEntity` in `@@graph/Graph.js`, which handles both. This means we just have to find ids we would like to delete and pass them to `onDeleteEntity`.

The simplest way would be to add a selector to `@@selectors` akin to the following:

```
export const getIgnoredEntityIds = s => {
  const properties = pipe(
    getProperties,
    filter(prop('selected')),
    values
  )(s);
  const toKeep = properties
    .reduce((acc, p) => Object.assign(acc, {
      [p.target]: true,
      [p.source]: true
    }), {});

  return keys(
    filter(
      c => !c.selected,
      omit(keys(toKeep), getClasses(s))
    )
  );
};
```

This way, we get a function that return ids of entities that could potentially be completely deleted. Now we just have to extract this information from the state somewhere and add a control element to handle the graph method invocation. This can be simply added to `@@components/GraphContainer.js`. The full implementation of this feature can be found in a pull request[40] on GitHub.

Custom schema node layout

Another example that might be interesting to implement is implementing a new node layout. Based on the AntV documentation[41] we can see that nodes expose a `updatePosition` method. To implement this then, we just need to add a method to the graph handler and lay out the nodes based on predefined criteria. For the sake of simplicity let's go with a simple grid layout.

All graph functionality related to the canvas and its data is contained in `@@graph/Graph.js`. We just need to implement a simple static method that accesses the graph instance, gets its nodes and updates their positions. It could look something like the following:

```
static gridLayout() {
  const rowCount = 5;
  const columnGap = 250;
  const rowGap = 200;
  const nodes = this.instance.getNodes();

  let rowIndex = 0;
  let columnIndex = 0;
  nodes.forEach(n => {
    n.updatePosition({
      x: columnIndex * columnGap,
      y: rowIndex * rowGap
    })

    n.getEdges().forEach(e => e.refresh());
    // Edges need to be refreshed to properly link to the nodes,
    // otherwise they stay in empty space
```

[40] https://github.com/jaresan/simplod/pull/28
[41] https://g6.antv.vision/en/docs/api/Items/itemMethods

```
    columnIndex++;
    if (columnIndex === rowCount) {
      columnIndex = 0;
      rowIndex++;
    }
  });
}
```

The only thing left to do is to tie a call to this method through a control element in the graph. Complete implementation of this feature can be found in a pull request[42] on GitHub.

5.3.5 Automatically generated documentation

This work also includes generated documentation from the JavaScript annotations inside the codebase using JsDoc[43].

The documentation is accessible directly on GitHub Pages[44].

[42]https://github.com/jaresan/simplod/pull/29
[43]https://github.com/jsdoc/jsdoc
[44]https://jaresan.github.io/simplod/documentation

6. Tests

This chapter describes the approach to tests and their implementation split into two sections.

The first describes unit tests, used to make sure the functionality present in the code is kept intact when the developer is adding implementing new features. This way, the developer can be confident that the changes they are making will not negatively impact other functionalities and break the application.

The second section presents manual test scenarios that act as an acceptance criteria for the requirements laid out in subsection 2.3.1. The reader is welcome to go through these scenarios one by one to confirm that all the requirements, as presented, are present in the final application and functional. These scenarios might as well be used as a necessary manual checklist to ensure the flow of the application hasn't broken before releasing a new version of the application. In a more rigorous version, this approach is also know as end to end testing[1].

6.1 Unit testing

An integral part of every production ready application is also a suite of tests which covers critical functionality. This section describes what tools were used to test the implemented code, the choice of what to cover, the reasons behind these choices, and what the resulting code coverage was.

6.1.1 Libraries

There are numerous available JavaScript testing frameworks[2]. We have used Mocha[3] for testing purposes of this application for its ease of use, popularity and familiarity. Nyc[4] package is used to measure code covered by the unit tests and provide an easy to read HTML output of the code coverage.

Every test file is specified with a `*.spec.js` suffix. The application also contains auxiliary test scripts declared in the `package.json` file as follows:

- `test`

  Runs all the test suites interactively, waiting and rerunning tests when changes are detected.

- `test:nowatch`

  Runs all tests once, does not wait.

- `test:coverage`

  Runs all tests and creates a code coverage report in HTML format.

- `test:coverage:show`

  Opens the HTML code coverage file.

[1]https://www.browserstack.com/guide/end-to-end-testing
[2]https://www.browserstack.com/guide/top-javascript-testing-frameworks
[3]https://mochajs.org/
[4]https://github.com/istanbuljs/nyc

6.1.2 Covered code

Majority of the codebase of this work consists of React components and wrappers around the graphing library itself, which are not ideal targets for unit testing. The design of the React components is an ever-changing process during the development phase and would impose additional time burden on the development while providing very little value. The graphing library is subject to the same, and moreover has to be worked around due to the utilisation of the canvas element, resulting in much higher time demands with little value added.

Based on the above mentioned reasons, it was decided to cover only the necessary and non-trivial functions with unit tests. These functions represent data handling and data transformation regarding RDF and the application model. All of the functionality regarding this area of the code is located in `@@data` and `@@app-state/model` respectively.

All of the logic in the `@@data` folder is 100% covered by corresponding unit tests. Non-trivial logic in `@@app-state/model` is covered by unit tests, which amounts to 81.9% coverage of the file.

Altogether, the code coverage provided by the unit tests is 26.54% for the whole project.

6.2 Manual test scenarios

This section describes step by step scenarios that can be used to make sure every functional requirement laid out in subsection 2.3.1 is present in the application. The section goes through all of the requirements, which are grouped based on similarities and the ease of being tested at once.

✓ icons represent a step where the user should check that the current application state corresponds to the result described in the scenario.

The requirements are covered by respective test cases according to the following table:

Table 6.1: Requirements tested by scenarios

	T1	T2	T3	T4	T5	T6
F1	✓	x	x	x	x	x
F2.1	✓	x	x	x	x	x
F2.2	✓	x	x	x	x	x
F2.3	x	✓	x	x	x	x
F2.4	x	✓	x	x	x	x
F2.5	x	✓	x	x	x	x
F2.6	x	✓	x	x	x	x
F2.7	x	x	✓	x	x	x
F3.*	x	x	x	x	✓	x
F4.*	x	x	✓	x	x	x
F5.*	x	x	x	x	✓	x
F6.*	x	x	x	x	x	✓
F6.6	x	x	✓	x	x	x
F6.7	x	x	x	x	✓	x

6.2.1 Preliminaries

For the manual test scenarios to be properly usable, the reader is at some steps required to input their Solid pod credentials. It is required the reader followed the steps in subsection 5.1.1 and set up their Solid pod accordingly before proceeding with the manual scenarios.

Every test scenario is started from the new start of the application with **default settings**, the reader is welcome to use the accompanying deployment of this work on GitHub[5] by clicking **Demo**.

The reader is encouraged to read through every scenario completely before going by their steps.

Default project

Some manual test scenarios require the tester to first load a project in the application. For the sake of removing repetition, the scenarios will refer to the following steps below as **Load the default project**:

1. Click *File* → *New*
2. Create a New project with the following configuration:

   Data Schema: `https://jaresan.github.io/simplod/example.ttl`

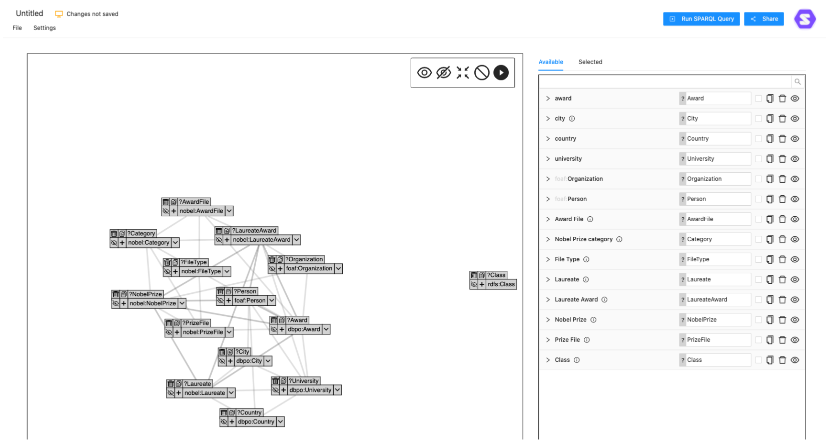
   Endpoint: `https://data.nobelprize.org/store/sparql`
3. Click Create

6.2.2 T1 - Visualization and simple configuration

This scenario tests the requirements F1, F2.1, F2.2:

1. **Load the default project**
2. ✓ The user is greeted with the default view of the nobel prizes data schema example as the following:

Figure 6.1: Default data schema view

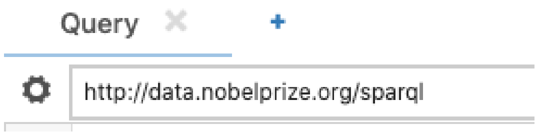


[5] `https://jaresan.github.io/simplod/`

3. Click ⊙ in the top right corner of the graph to execute a query. This step is to check the SPARQL endpoint, the query result is irrelevant at this point.

4. ✓ The SPARQL endpoint shows `https://data.nobelprize.org/store/sparql`

Figure 6.2: Sparql endpoint



http://data.nobelprize.org/sparql

5. Click *File* → *Properties*

6. Under SPARQL Endpoint, change
   `https://data.nobelprize.org/store/sparql`
   to `http://example.com/sparql`

7. Click Ok

8. Click ⊙ in the top right corner of the graph to execute a query

9. ✓ The SPARQL endpoint shows `http://example.com/sparql`

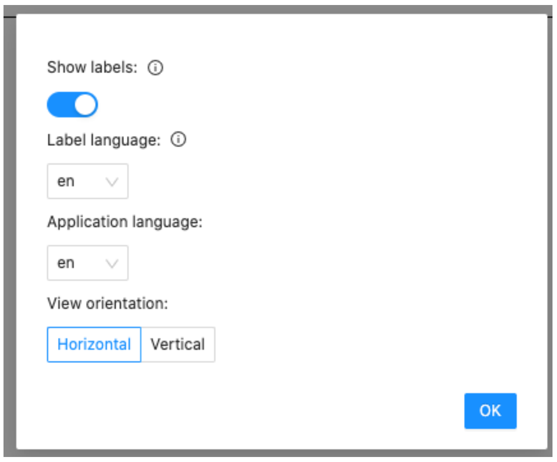
6.2.3 T2 - Language selection

This scenario tests the requirements F2.3, F2.4, F2.5, F2.6:

1. Open the application

2. Click "Settings", the menu looks as follows":

Figure 6.3: Settings in English



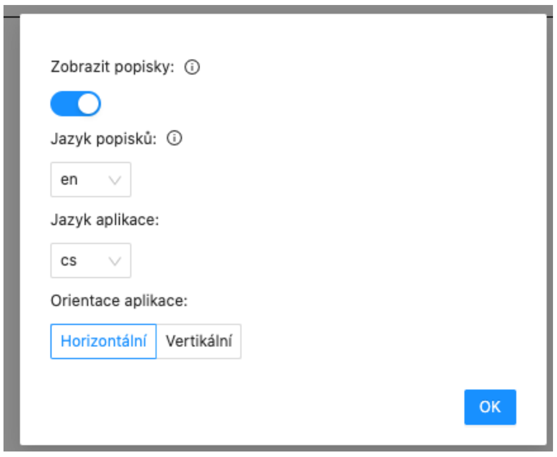
Show labels: ⓘ
Label language: ⓘ en
Application language: en
View orientation: Horizontal | Vertical

3. Under "Application language", change the language to "cs"

4. Refresh the application

5. Open Settings
6. ✓ The Settings menu now has its language updated and looks as follows:

Figure 6.4: Settings in Czech


Zobrazit popisky: ⓘ

Jazyk popisků: ⓘ en

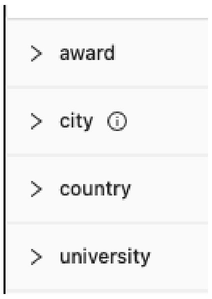
Jazyk aplikace: cs

Orientace aplikace: [✓ Horizontální] Vertikální

OK

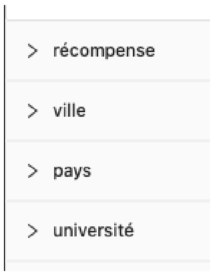
7. Change the application language back to English for the sake of the test and refresh the application
8. **Load the default project**
9. Make sure "Show labels" in Settings is turned on and "Label language" is set to **en**
10. ✓ The list view contains the following items:

Figure 6.5: English labels



11. Change the "Label language" in Settings to **fr**
12. ✓ Items from the previous step changed to the following:

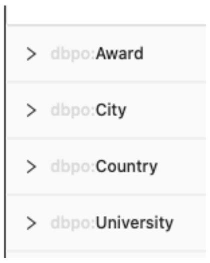
Figure 6.6: French labels



13. Turn "Show labels" in Settings off

14. ✓ Items from the previous step changed to the following:

Figure 6.7: Labels turned off



15. Create a New project with the following configuration:
    Data Schema:
    `https://jaresan.github.io/simplod/examples/opendata.ttl`
    Endpoint:
    `http://linked.opendata.cz/sparql`

16. For entity **skos:Concept** select the property **skos:prefLabel**

17. Execute the query by pressing the ⏵ button

18. ✓ Check there are results for both English (@en) and Czech (@cs) language

19. Under *File* → *Properties* → *Property languages*, add **en**

20. Execute the query by pressing the ⏵ button

21. ✓ Check there are results only for English (@en) language

6.2.4 T3 - Offline capabilities

This scenario tests the requirements F2.7 in that the user is able to make changes if the internet goes offline after the user has opened the application, it shows that the user can make a selection, save it locally and load it afterwards:

1. **Load the default project**

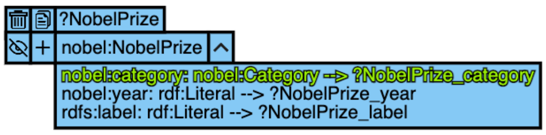
2. Turn the internet offline for the application (e.g. turn the wifi off or use throttling from developer tools in the browser)

3. Delete all entities except for **NobelPrize** from the list by pressing the 🗑 icon

4. Select **nobel:category** on **NobelPrize**

5. Save the changes made to browser (press ctrl+s or go to *File* → *Save* → *Save to browser storage* The graph should look like this:

Figure 6.8: Saved state



nobel:category: nobel:Category --> ?NobelPrize_category
nobel:year: rdf:Literal --> ?NobelPrize_year
rdfs:label: rdf:Literal --> ?NobelPrize_label

6. Close the application

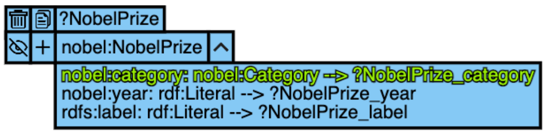
7. Turn the internet on and reload the application

8. Turn the internet off for the application

9. Load the data from browser storage *File* → *Load* → *Load from browser storage*

10. ✓ Loaded graph reflects the changes saved in previous steps:

Figure 6.9: Loaded state



nobel:category: nobel:Category --> ?NobelPrize_category
nobel:year: rdf:Literal --> ?NobelPrize_year
rdfs:label: rdf:Literal --> ?NobelPrize_label

6.2.5 T4 - Data handling

This scenario tests the user's capability to save their data to their Solid pod, view the files they have saved, load and delete them and update the project's permissions, testing all of the requirements under F4 and an interaction requirement F6.6:

1. Click *File* → *New*

2. Create a New project with the following configuration:

   Data Schema:

   `https://jaresan.github.io/simplod/examples/images.ttl`

   Endpoint:

   `http://www.imagesnippets.com/sparql/images`

3. Click Create

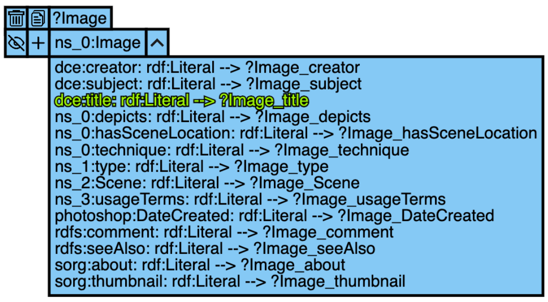
4. Hover over the top right avatar → *Login*

5. Proceed to login with the Solid pod provider of your choice

6. ✓ Hovering over the top right avatar now shows "Logout"

7. Select **dce:title** on the **Image** entity. The resulting graph looks like this:

Figure 6.10: Selection to save



```
?Image
ns_0:Image
dce:creator: rdf:Literal --> ?Image_creator
dce:subject: rdf:Literal --> ?Image_subject
dce:title: rdf:Literal --> ?Image_title
ns_0:depicts: rdf:Literal --> ?Image_depicts
ns_0:hasSceneLocation: rdf:Literal --> ?Image_hasSceneLocation
ns_0:technique: rdf:Literal --> ?Image_technique
ns_1:type: rdf:Literal --> ?Image_type
ns_2:Scene: rdf:Literal --> ?Image_Scene
ns_3:usageTerms: rdf:Literal --> ?Image_usageTerms
photoshop:DateCreated: rdf:Literal --> ?Image_DateCreated
rdfs:comment: rdf:Literal --> ?Image_comment
rdfs:seeAlso: rdf:Literal --> ?Image_seeAlso
sorg:about: rdf:Literal --> ?Image_about
sorg:thumbnail: rdf:Literal --> ?Image_thumbnail
```

8. Turn the internet off for the application

9. Save the data to Solid pod via *File* → *Save* → Click the "+" button, name the file **_testScript** and click save

10. ✓ Notification is displayed about the inability to save the file with a retry counter

11. Turn the internet on for the application

12. ✓ Retry counter counts to 0 and the files is saved

13. Execute the query and set the limit to 1

14. Click the "Share" button in the top right corner

15. Copy the "Direct Web URL" by clicking the 🗐 icon

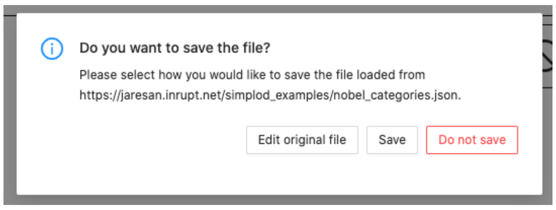
16. Open the copied URL in a new window

17. ✓ Visually check that the loaded URL corresponds to sensible data (can also be presented in the .json format):

```
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
    <head>
        <variable name="Image"/>
        <variable name="title"/>
    </head>
    <results>
        <result>
            <binding name="Image">
                <uri>...</uri>
            </binding>
            <binding name="title">
                <literal xml:lang="en">...</literal>
            </binding>
    </result>
    ...
```

18. Open the "CSV URL" by clicking the ▶ icon
19. ✓ A .csv file with sensible Image and Title information is downloaded
20. Under "Permissions" select "private"
21. Copy the "Current file URL" by clicking the 🗐 icon
22. Open an anonymous window and paste the URL from previous step
23. ✓ Access to the requested resource is denied
24. Under "Permissions" select "Public/read"
25. Refresh the anonymous window from the previous step
26. ✓ The contents of the .json file are displayed
27. **Load the default project**
28. Open the Load menu *File* → *Load*
29. ✓ File **__testScript.json** is present
30. Click **__testScript.json** and click "Load"
31. ✓ The user is greeted with the option to edit the original file:

Figure 6.11: Loaded project

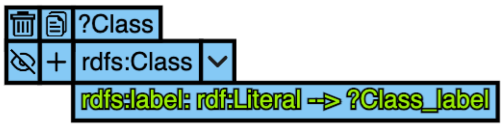


Do you want to save the file?

Please select how you would like to save the file loaded from https://jaresan.inrupt.net/simplod_examples/nobel_categories.json.

32. ✓ The saved project is loaded and looks like this:

Figure 6.12: Loaded project



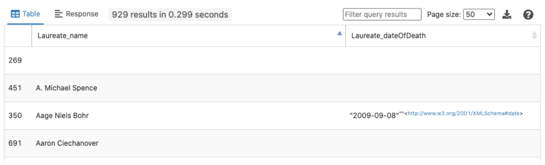
33. Open the Load menu *File* → *Load*
34. Click **__testScript.json** and click "Delete"
35. ✓ The file **__testScript.json** is deleted

6.2.6 T5 - SPARQL Capabilities

The following scenario test the basic capabilities of running a SPARQL query and retrieving the results, it encompasses the requirements F3 and F5.

1. **Load the default project**
2. For the entity **Laureate** select **dbpedia2:dateOfDeath** as optional via the ? and **foaf:name**
3. Execute the query via the ▶ button
4. Sort the results in ascending order by name by clicking on the name column header
5. ✓ The results are as follows:

Figure 6.13: Death date optional

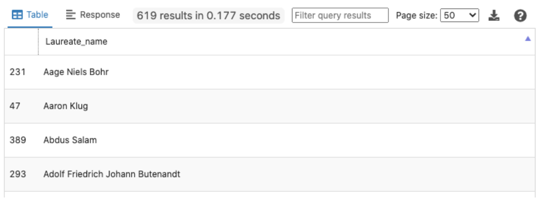


929 results in 0.299 seconds

	Laureate_name	Laureate_dateOfDeath
269		
451	A. Michael Spence	
350	Aage Niels Bohr	"2009-09-08"^^<http://www.w3.org/2001/XMLSchema#date>
691	Aaron Ciechanover	

6. Change **?Laureate_name** to **?Laureate_name2** in the SELECT part of the SPARQL query
7. ✓ SPARQL Query edited warning is shown Figure 5.20
8. Execute the query via the ▶ button
9. ✓ The result column is renamed to **?Laureate_name2**
10. Close the query editor
11. Mark **dbpedia2:dateOfDeath** as not optional via ? and mark it as hidden via 👁
12. Execute the query via the ▶ button
13. ✓ The results are as follows:

Figure 6.14: Death date required but hidden



619 results in 0.177 seconds

	Laureate_name
231	Aage Niels Bohr
47	Aaron Klug
389	Abdus Salam
293	Adolf Friedrich Johann Butenandt

14. Click the ⬇ icon in the top right corner of the result table
15. ✓ .csv file is downloaded with Laureate names

6.2.7 T6 - Graph interactions

The following scenario tests arbitrary features laid out in requirements F6:

1. **Load the default project**
2. Click on any one edge
3. ✓ Menu containing the properties between the entities linked is shown
4. Select a single property from the popup menu and mark it as optional via the ❓
5. ✓ Cartesian product warning is shown Figure 5.19
6. ✓ The edge has a dash pattern
7. Click the 👁 icon in the top right corner of the graph to hide all not selected entities
8. ✓ The graph now only displays the two nodes which are connected by the selected edge
9. Copy any one of the entities via the 🗐 button on the graph node
10. ✓ New node is created with the same entity type, new edge is added between this node and the node that wasn't copied
11. Delete the newly created entity via the 🗑 button node
12. ✓ The newly created entity and its edge are now deleted and the graph is showing only two items and their edge again

If all of the test scenarios laid out above were followed step by step and all ✓ have been successfully validated, the manual testing process has been successful and the requirements laid out in subsection 2.3.1 can be considered working.

7. Evaluation

This chapter describes the final evaluation of this work. In the first part, the author discusses how well the set goal was fulfilled and what was achieved. The second part contains a consolidated output of the System Usability Scale forms filled out by participants and its conclusion.

7.1 Goal fulfillment

The goal of this work was to create an application that would allow users to query Linked Data they would be interested in without the need to learn SPARQL. To achieve this, requirements covering the use cases were laid out in the assignment of this work and were continuously updated in section 2.3 during various meetings held between the author and their supervisor. The established goals were fulfilled completely, as can be confirmed by following the manual test scenarios in section 6.2.

While the application can still be improved in various ways, e.g. by examples provided in section 4.5, the functionality that was required to be implemented is present in the final product and hence allows the users to operate over data schemas in a visual fashion with ease, instead of having to learn SPARQL.

7.2 System usability scale results

To evaluate the usability of the application, we chose to utilize the industry standard SUS[1] due to its ease of administration and reliable results even for smaller samples.

The form consists of 10 questions. On each question, the participants can answer between 0-4, where 0 represents ”strongly disagree” and 4 means ”strongly agree”. The questions are as follows:

1. I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.

[1]`https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html`

9. I felt very confident using the system.

10. I needed to learn a lot of things before I could get going with this system.

"To calculate the SUS score, first sum the score contributions from each item. Each item's score contribution will range from 0 to 4. For items 1,3,5,7,and 9 the score contribution is the scale position minus 1. For items 2,4,6,8 and 10, the contribution is 5 minus the scale position. Multiply the sum of the scores by 2.5 to obtain the overall value of SU." Brooke [1996]

In our evaluation, 4 participants, 3 of which had no prior exposure to Linked Data, were provided with brief introduction to the problem and instructed to follow the examples outlined in subsection 5.1.5. After completing the examples, the participants were asked to fill out the SUS form.

The gathered results were as follows - every cell represents a number of occurrences of a given mark for the given question:

Table 7.1: SUS form results

	0	1	2	3	4
Q1	1	1	2	0	0
Q2	1	3	0	0	0
Q3	0	0	1	3	0
Q4	0	1	2	1	0
Q5	0	0	0	2	2
Q6	1	3	0	0	0
Q7	0	0	3	1	0
Q8	1	1	0	1	1
Q9	0	1	3	0	0
Q10	1	3	0	0	0

Applying the calculation system mentioned above on the results provided in Table 7.1, we can calculate the SU value to be 63.125.

Considering 68 to be the average, as per Interpreting Scores[2], this would mark the application as below average in usability. It is important to keep in mind, however, that 3 out of the 4 participants had no prior experience with Linked Data.

The score received from the participant with prior Linked Data experience was 72.25, which might point to the fact that the application or the accompanying materials are not well suited for inexperienced users.

Based on the feedback from the participants, this score could be improved by having a better onboarding experience for the users to better understand the problematic, accompanied by more intuitive controls.

[2]`https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html`

8. Conclusion

During the process of this work, a visualization tool was created to help users generate SPARQL queries without the need of knowledge of SPARQL itself. While other data schema visualizers exist, as described in Table 2.6.2, no such solution really aims at tackling the same problem that is solved in this work.

This work simplifies working with the RDF schemas for the user, but it also allows SPARQL knowledgeable users to edit the query directly, effectively removing any limitation that could stem from using such a tool.

The application could be improved with various features, some of which are described in section 4.5, and would benefit from better designed UI and UX. However, even in its current state, it can be used to better navigate around RDF data schemas and allows the users to further share their created views or catalogue them to later directly download the desired data. The application also seemingly integrates with Solid Pods, allowing the users to manage the files in their Pods and share their findings with ease.

A byproduct of the application development was creating a SPARQL server proxy to be used in case of failing YASGUI requests. This proxy, described in section 4.3, acts as fallback option for executing SPARQL queries against endpoints that might fail due to various reasons, mainly CORS and Same-Origin policy.

One of the downsides of utilizing this tool is having to manually use the pipeline to create the data schema as described in subsection 5.2.2. In the future this could be alleviated by incorporating the pipeline execution as a part of the application, allowing the users to create data schemas in the application directly.

Otherwise the deployment process for administrators is quite simple, as described in subsection 5.2.4. Since the application is a front-end one, it can conveniently be used directly from the demo site[1] with proper set up, without the need for custom deployment.

[1]`https://jaresan.github.io/simplod/build/index.html`

Bibliography

Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, May 2001. `http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21`, `https://www.researchgate.net/publication/225070375_The_Semantic_Web_A_New_Form_of_Web_Content_That_is_Meaningful_to_Computers_Will_Unleash_a_Revolution_of_New_Possibilities`.

John Brooke. *"SUS-A quick and dirty usability scale." Usability evaluation in industry*. CRC Press, June 1996. URL `https://www.crcpress.com/product/isbn/9780748404605`. ISBN: 9780748404605, `https://www.researchgate.net/publication/228593520_SUS_A_quick_and_dirty_usability_scale`.

Jakub Klímek and Petr Skoda. Linkedpipes DCAT-AP viewer: A native DCAT-AP data catalog. In Marieke van Erp, Medha Atre, Vanessa López, Kavitha Srinivas, and Carolina Fortuna, editors, *Proceedings of the ISWC 2018 Posters & Demonstrations, Industry and Blue Sky Ideas Tracks co-located with 17th International Semantic Web Conference (ISWC 2018), Monterey, USA, October 8th - to - 12th, 2018*, volume 2180 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2018. URL `http://ceur-ws.org/Vol-2180/paper-32.pdf`.

Jakub Klímek and Petr Škoda. LinkedPipes ETL in use: practical publication and consumption of linked data. In Maria Indrawan-Santiago, Matthias Steinbauer, Ivan Luiz Salvadori, Ismail Khalil, and Gabriele Anderst-Kotsis, editors, *Proceedings of the 19th International Conference on Information Integration and Web-based Applications & Services, iiWAS 2017, Salzburg, Austria, December 4-6, 2017*, pages 441–445. ACM, 2017. doi: 10.1145/3151759.3151809. URL `https://doi.org/10.1145/3151759.3151809`.

Roger Menday and Nandana Mihindukulasooriya. Linked Data Platform 1.0 Primer. W3C note, W3C, April 2015. URL `https://www.w3.org/TR/2015/NOTE-ldp-primer-20150423/`.

Guus Schreiber and Yves Raimond. RDF 1.1 Primer. W3C note, W3C, June 2014. URL `https://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/`.

Andy Seaborne and Steven Harris. SPARQL 1.1 Query Language. W3C recommendation, W3C, March 2013. URL `https://www.w3.org/TR/2013/REC-sparql11-query-20130321/`.

Hernán Vargas, Carlos Buil-Aranda, Aidan Hogan, and Claudia López. *RDF Explorer: A Visual SPARQL Query Builder*, pages 647–663. Aidan Hogan, 10 2019. ISBN 978-3-030-30792-9. doi: 10.1007/978-3-030-30793-6_37.

List of Figures

1.1 London Linked Data example 5

2.1 Use case diagram 14
2.2 Data schema example 17
2.3 Data schema example mockup 18
2.4 Lakes found in Chile 20
2.5 Drugs for cancer that target genes related to cell proliferation . . 20
2.6 Example of querying people's alma maters and birth place 21

3.1 Overview of the application life cycle 30
3.2 Main screen 31
3.3 User information - Anonymous 31
3.4 User information - Logged in 31
3.5 Main area 32
3.6 Data entity 33
3.7 Query editor 34

4.1 Overview of the implemented application architecture 36
4.2 SPARQL proxy 37
4.3 Default use case 38
4.4 Selection order 39
4.5 Selection and hiding always have the same state 39
4.6 Laureate removed from the result set 40
4.7 Optional link 40
4.8 Nested optionals 41
4.9 Two connected components 42
4.10 Two components connected by an optional edge 42
4.11 Optionality cycle 43
4.12 AntV edge direction 44
4.13 Same type distinction 45
4.14 Double optional target 46

5.1 Avatar menu 48
5.2 Picking Solid Pod provider 49
5.3 Provider login screen 49
5.4 App permission prompt 49
5.5 Layout 50
5.6 Project status 51
5.7 Avatar menu 51
5.8 Share menu 52
5.9 Settings menu 53
5.10 Label 53
5.11 Label language 54
5.12 File menu 54
5.13 New file 55
5.14 Save & load 56

5.15 Solid pod 57
5.16 By URI 57
5.17 Properties 58
5.18 Properties 58
5.19 Cartesian product warning 59
5.20 Customized query warning 59
5.21 Graph area 60
5.22 Graph node 61
5.23 Edge 62
5.24 Edge states 63
5.25 Graph toolbar 63
5.26 List overview 64
5.27 Selected tab 65
5.28 Column order example 1 65
5.29 Column order example 2 65
5.30 Search functionality 66
5.31 Entity row 66
5.32 Expanded properties 67
5.33 Entity title hover 67
5.34 Entity description 67
5.35 Entity description 68
5.36 Variable name field 68
5.37 Renamed variable result 68
5.38 Entity row actions 68
5.39 Data property row 69
5.40 Object property row 69
5.41 Object property variable field tooltip 70
5.42 Object property target 70
5.43 Nobel prize example default view 71
5.44 NobelPrize entity 71
5.45 Selected property 72
5.46 List view 72
5.47 Run Query 72
5.48 SPARQL Results 73
5.49 List view 73
5.50 Cleaner results 74
5.51 Download CSV 74
5.52 Copy yasgui query URL 75
5.53 Yasgui query results 76
5.54 List view 76
5.55 Nobel prize example default view 77
5.56 Laureate entity properties 78
5.57 Laureate properties of interest 78
5.58 All properties of interest 79
5.59 Cleaned up selection 79
5.60 Execute query button 79
5.61 Results 80
5.62 Results 80

5.63 Results 81
5.64 Results 81
5.65 Selected entities 81
5.66 Deselected entities 82
5.67 Result set without IRIs 82
5.68 Selection with place of death 82
5.69 Results with place of death 83
5.70 Selection with cloned Country entity 83
5.71 Newly listed properties 84
5.72 Proper selection 84
5.73 Proper death place results 84
5.74 Death place optional 85
5.75 Results with death place optional 85
5.76 Load by URI 86
5.77 Graph loaded 86
5.78 Edge descriptions 87
5.79 Edge descriptions 87
5.80 Graph selection 88
5.81 List selection 88
5.82 Query limit 89
5.83 Results 89
5.84 Save menu 89
5.85 Solid pod files 90
5.86 New filename 90
5.87 File saved notification 90
5.88 Status bar after remote save 90
5.89 Permission drop-down 91
5.90 Permission drop-down 91
5.91 Permissions changed successfully 92
5.92 Upload button 93
5.93 Upload detail 94
5.94 Pipeline graph 94
5.95 Pipeline start node 94
5.96 Start node options 95
5.97 Execute button 95
5.98 Execute button 95
5.99 Execute button 96
5.100Results location 96
5.101Download results 96
5.102Download as zip 97

6.1 Default data schema view 108
6.2 Sparql endpoint 109
6.3 Settings in English 109
6.4 Settings in Czech 110
6.5 English labels 110
6.6 French labels 110
6.7 Labels turned off 111

6.8 Saved state 112
6.9 Loaded state 112
6.10 Selection to save 113
6.11 Loaded project 114
6.12 Loaded project 114
6.13 Death date optional 115
6.14 Death date required but hidden 115

A.1 Solid Pod menu 126
A.2 Trusted applications 127
A.3 Creating a folder 128

List of Tables

2.1 Comparison criteria 19
3.1 Application type comparison 23
6.1 Requirements tested by scenarios 107
7.1 SUS form results 118

A. Solid Pod troubleshooting

This chapter describes manual setup of Solid Pods in detail, should the steps outlined in subsection 5.1.1 fail.

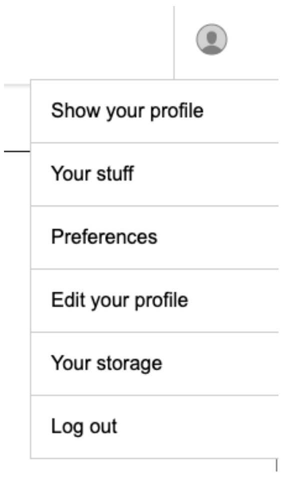
A.1 Logging in with Solid Pod

To set up all permissions for the application, the user has to sign in[1] to their Solid Pod first.

After signing in, the user can navigate in the application utilizing a tooltip menu by hovering over their avatar in the top right corner.

In the following sections navigating to a certain part of the Solid Pod takes this menu into account and references it.

Figure A.1: Solid Pod menu



A.2 Enabling trusted apps

To allow the application to manipulate data in the Solid Pod, it has to be added to the Solid Pod's trusted sources. The application can be marked as trusted by following these steps:

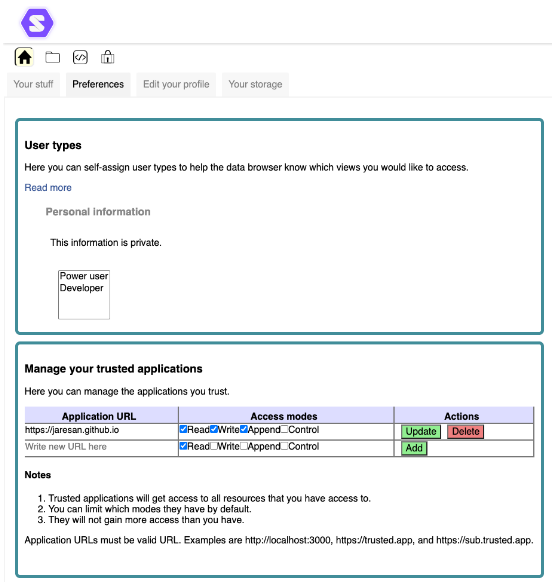
- Go to "Preferences" in the menu
- Go to "Manage your trusted applications" section and add the URL where the application is hosted, as a demo example `https://jaresan.github.io` can be used as the application URL

  Note: Make sure you omit the trailing slash, i.e. `https://hosting.com`, not `https://hosting.com/`
- Check "Read", "Write", "Append" privileges
- Click on "Add"

[1]`https://solid.community/login`

This allows the application to manipulate data in the user's Solid Pod when they are logged in via their Solid Pod providing entity.

Figure A.2: Trusted applications



User types

Here you can self-assign user types to help the data browser know which views you would like to access.

Read more

Personal information

This information is private.

Power user
Developer

Manage your trusted applications

Here you can manage the applications you trust.

Application URL	Access modes	Actions
https://jaresan.github.io	☑Read ☑Write ☑Append ☐Control	Update Delete
Write new URL here	☑Read ☐Write ☐Append ☐Control	Add

Notes

1. Trusted applications will get access to all resources that you have access to.
2. You can limit which modes they have by default.
3. They will not gain more access than you have.

Application URLs must be valid URL. Examples are http://localhost:3000, https://trusted.app, and https://sub.trusted.app.

A.3 Creating a save destination

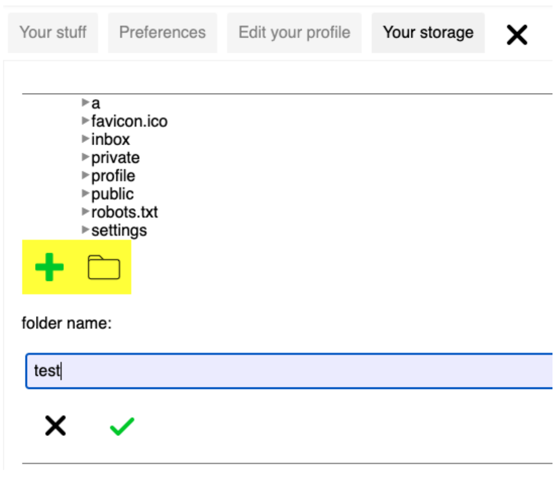
To create a new folder where the application data could be stored, the user can follow the steps below:

- Go to "Your storage" in the menu
- Click the green plus button
- Select the folder icon and choose new folder name
- Click the green check button
- The newly created folder can be found at

  `https://USERNAME.solid.community/FOLDER_NAME`

  where USERNAME and FOLDER_NAME represent the user's Solid Pod login and the folder name chosen in the previous steps respectively

Figure A.3: Creating a folder



▸a
▸favicon.ico
▸inbox
▸private
▸profile
▸public
▸robots.txt
▸settings

folder name:

test

B. Resources

- Attached `.zip` file

  This work comes attached with a `.zip` archive consisting of the GitHub repository archive for this application and an archive for the proxy application described in section 4.3. Both archives correspond to the repository versions under tag `base`.

- `https://jaresan.github.io/simplod/`

  GitHub Pages with description of the application and its live demo.

- `https://github.com/jaresan/simplod`

  GitHub repository of the application. The version this work is submitted with corresponds to tag `base`.

- `https://github.com/jaresan/sparql-proxy/`

  GitHub repository of the SPARQL proxy described in section 4.3.

- `https://github.com/jaresan/lod-cloud`

  Repository with resources for proper execution of the data schema pipeline described in subsection 5.2.2.