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

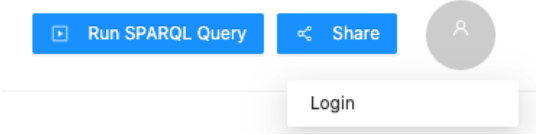
The following sections are written for Solid pods hosted at Inrupt[3]. This application shall be usable with any Solid pod providing service but the details regarding its use might differ.

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.

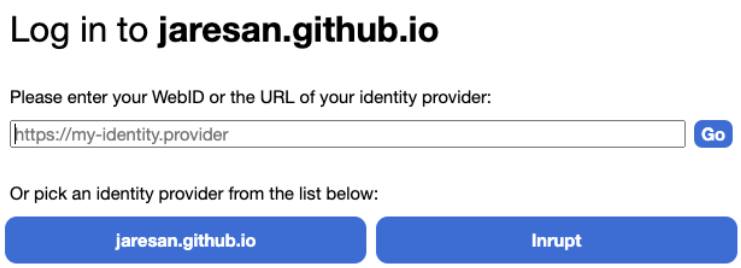
Figure 5.1: Avatar menu



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

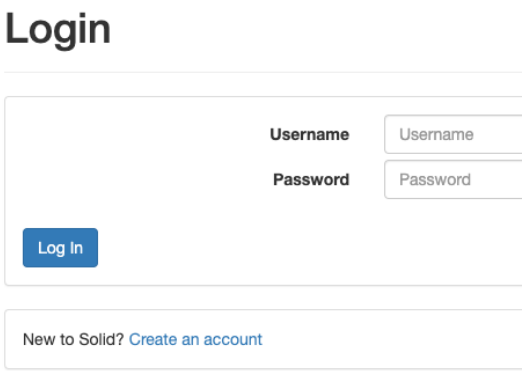
Upon clicking on the login button, the user is required to choose their solid pod provider either from the list provided or by specifying it themselves.

Figure 5.2: Picking solid pod provider



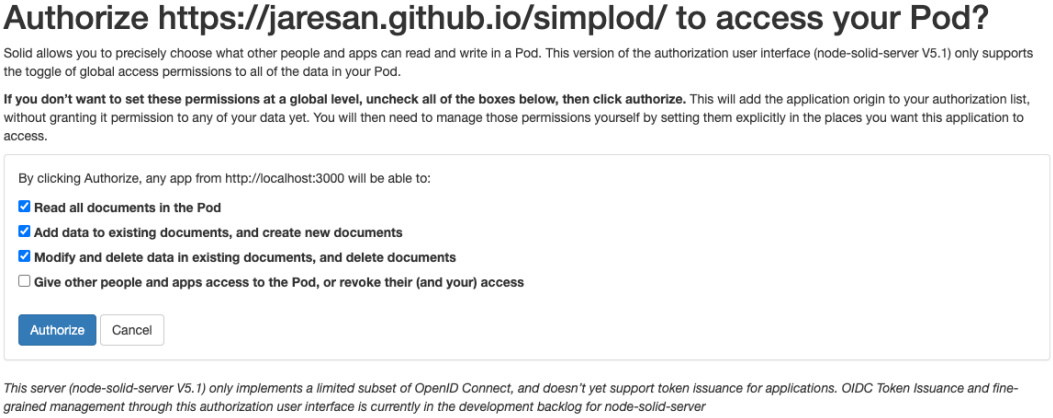
After the provider is chosen, the user is able to authenticate via the provider's login screen.

Figure 5.3: Provider login screen



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



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

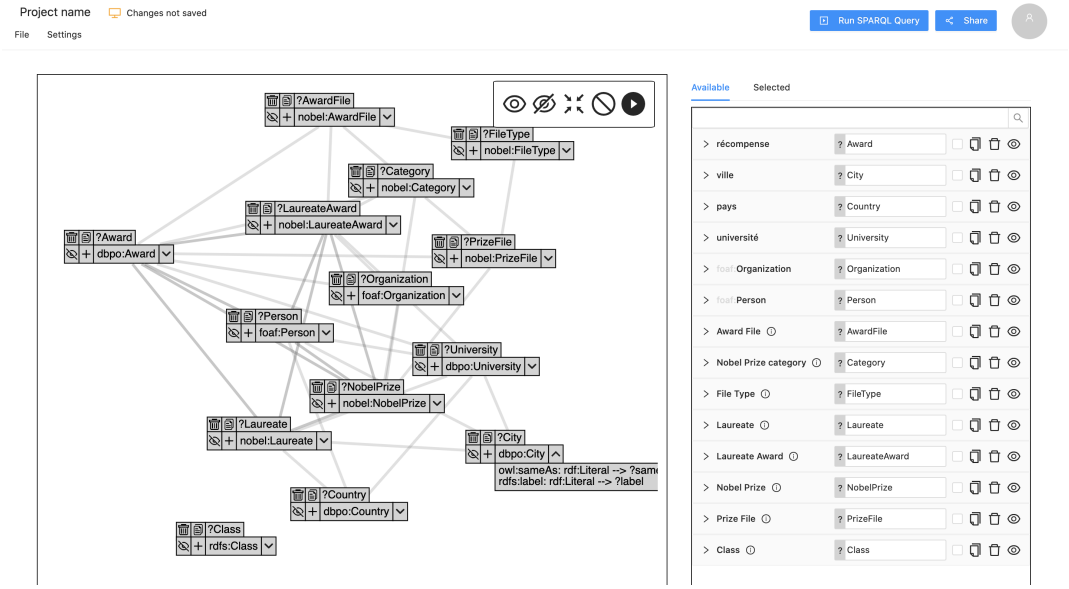


Figure 5.5 displays the overview "main screen" of the application. It is split into three main parts:

- 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

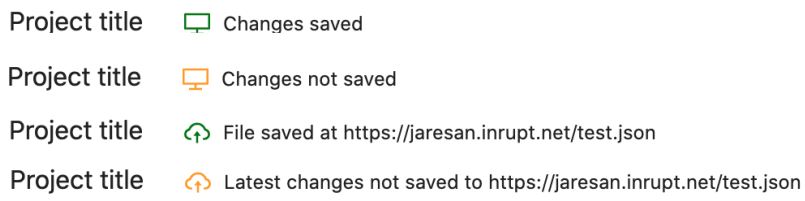


Figure 5.6 contains the following:

- 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

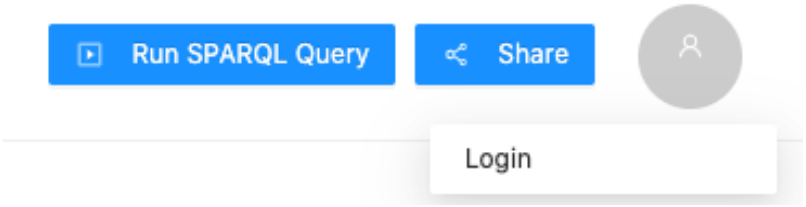


Figure 5.7 contains the following:

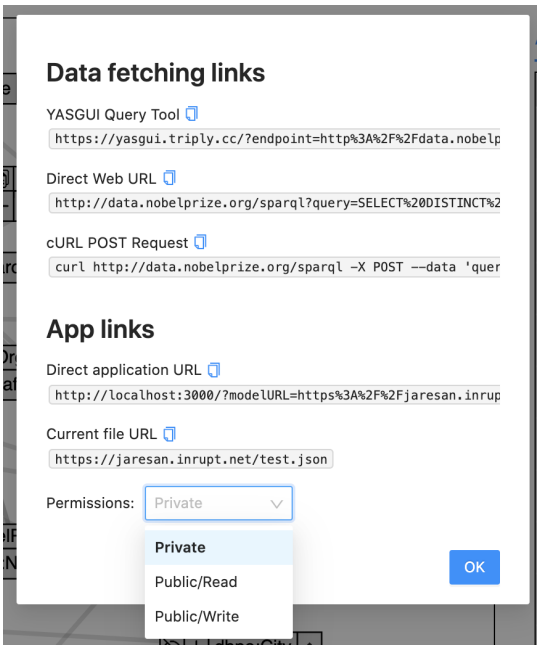
- Run SPARQL Query

  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



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
  - **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

- 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

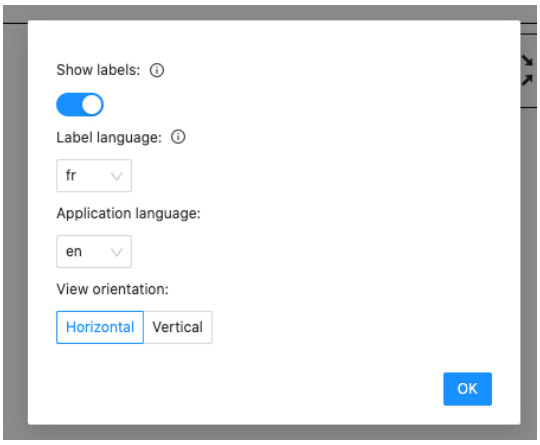
    Public/read - Can be viewed by anyone but not edited

    Public/write - Can be edited by anyone

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

Settings

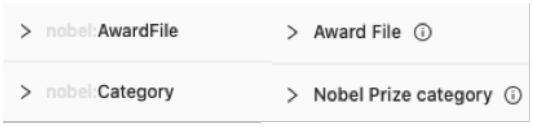
Figure 5.9: Settings menu



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

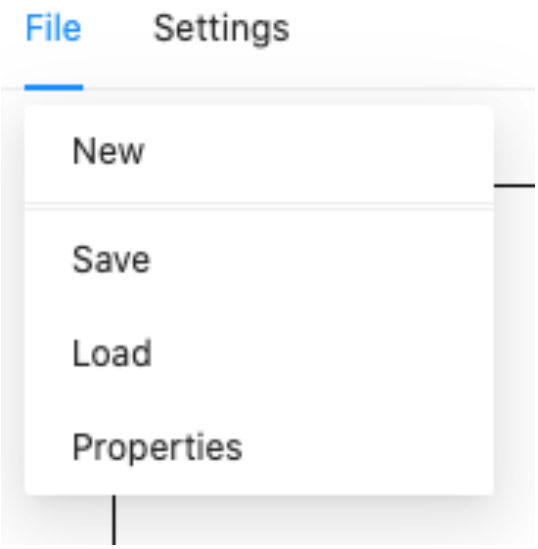


Figure 5.12 allows the user to create a new project, save/load one or change the project's properties.

Figure 5.13: New file

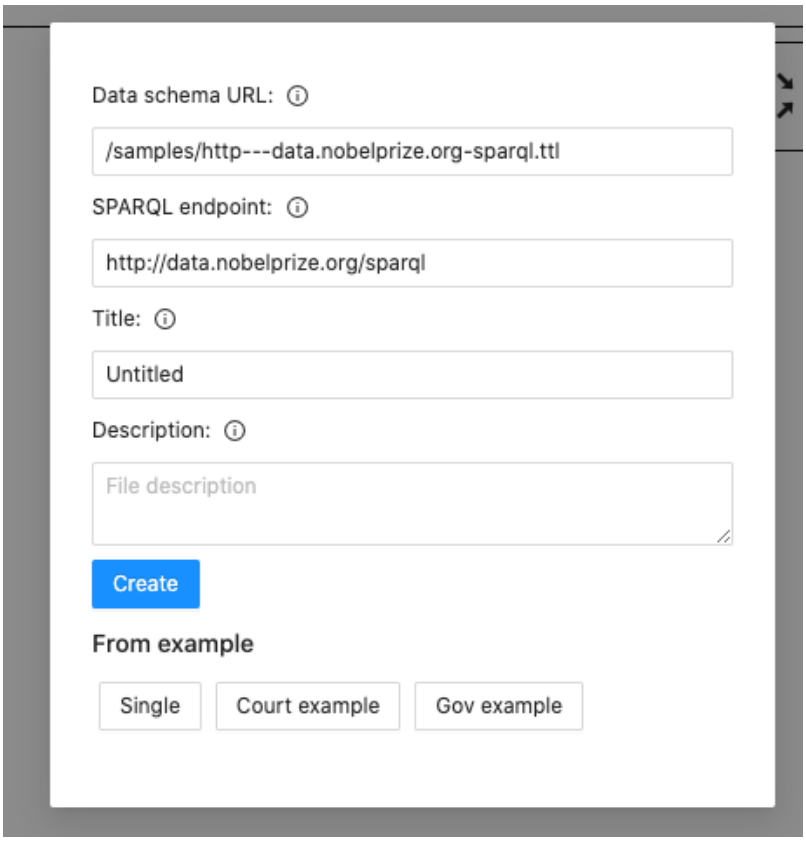


Figure 5.13 shows new project window, allowing the user to create a new project with fields as follows:

- 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.

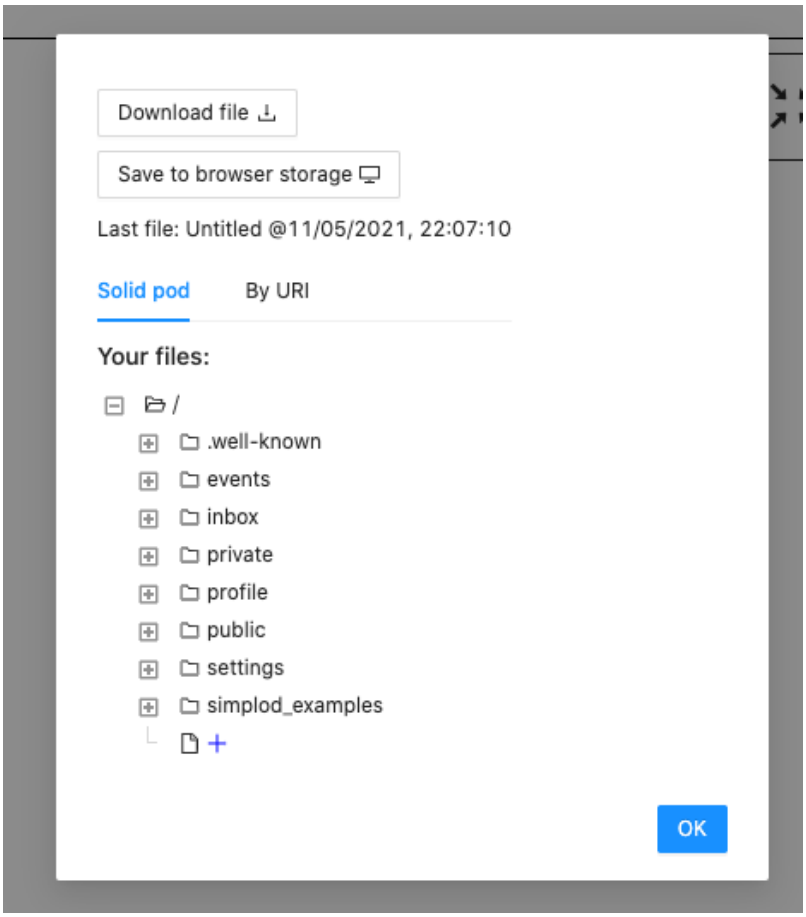
- Create

  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



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

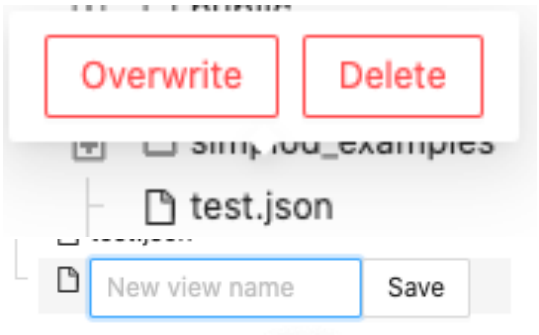
  Saves the current state of the project to the browsers storage, allowing the user to close the application and resume their work later. Due to the nature of this application, only one file can be saved to the browser storage at one time.

- 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

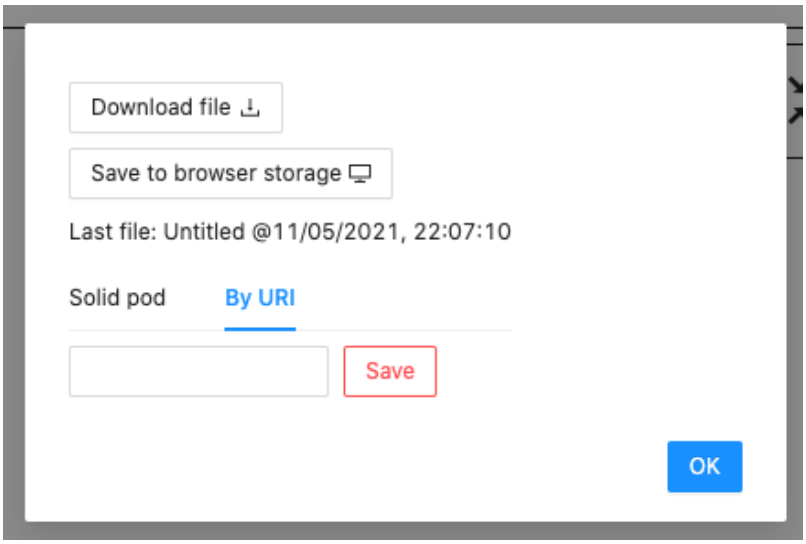
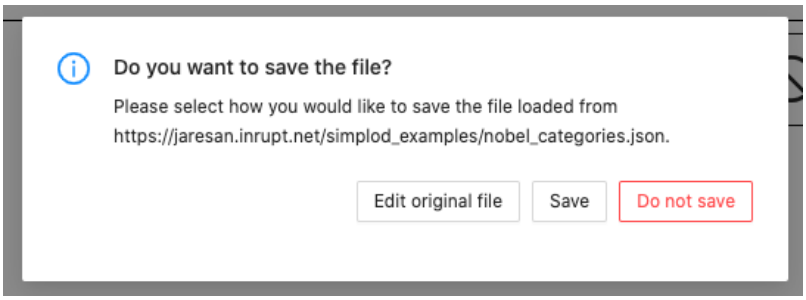


Figure 5.16 allows the user to specify the full URI path where the project file should be saved. The authenticated user has to be granted write permission to be able to save to this location.

Edit original file

Figure 5.17: Properties



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

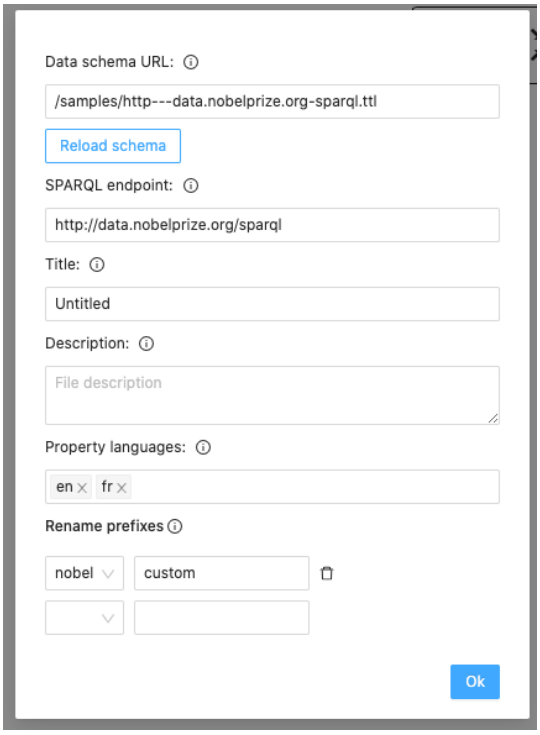


Figure 5.18 allows the user to change the properties of the project. Fields in this menu correspond to the same fields as in Figure 5.13 with extra items as follows:

- 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

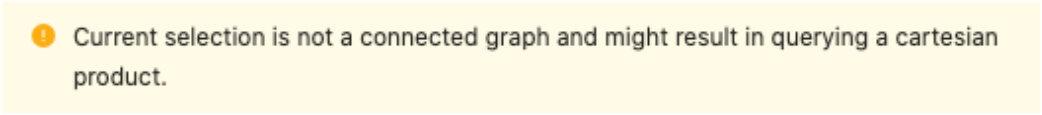
  Allows the users to rename the prefixes found in the application.

  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.

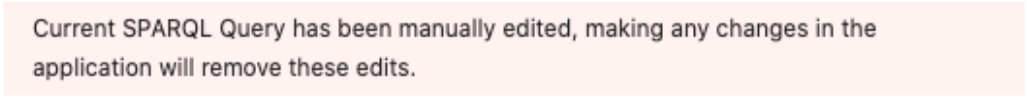
Warnings

Figure 5.19: Cartesian product warning



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



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

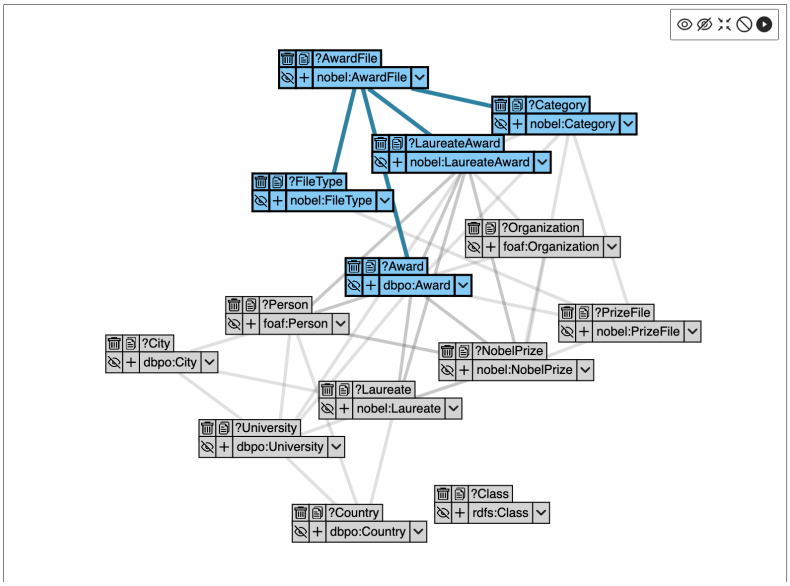


Figure 5.21 shows the data schema as a graph where nodes represent entities and edges represent the relationships between them. The users are able to interact with the graph in the following ways:

- 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

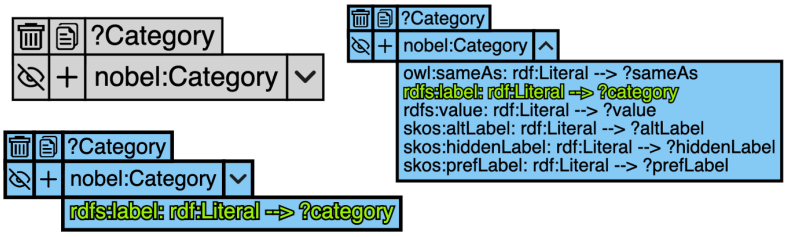


Figure 5.22 represents a single entity from the data schema. Its controls are as follows:

- 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.

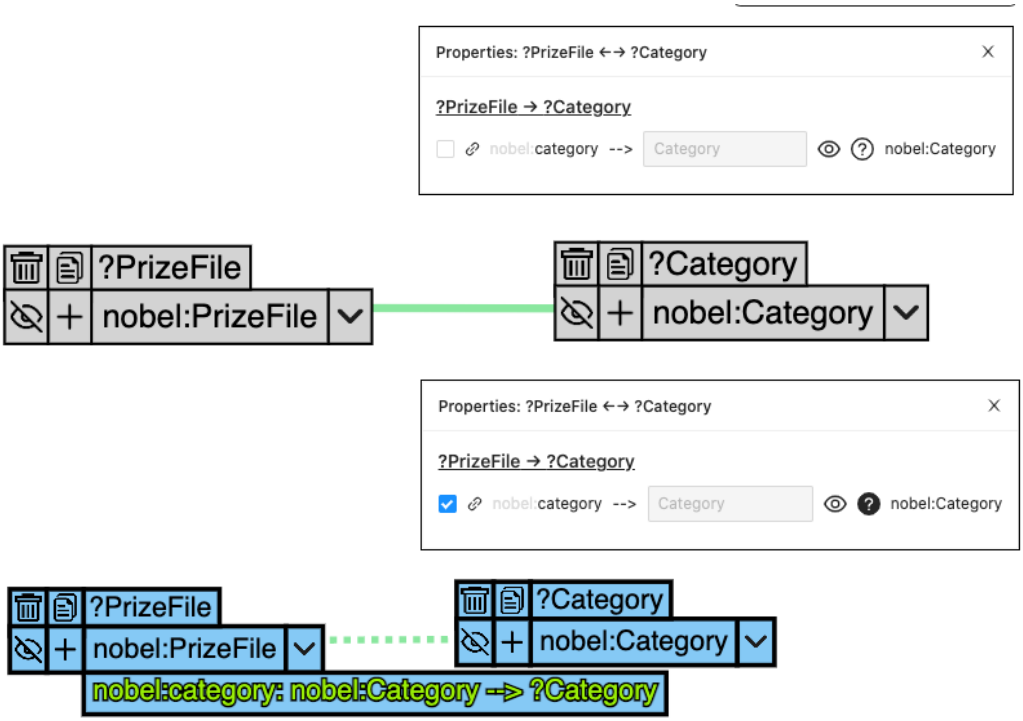
- Expand/Collapse

  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

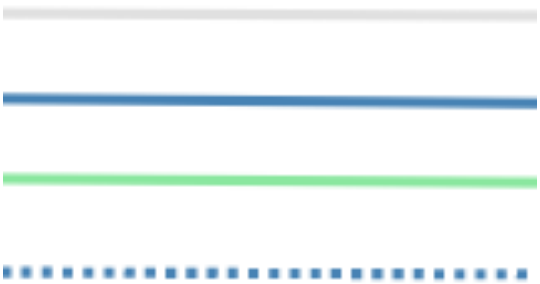
  Hovering over an edge highlights it and also its corresponding nodes. Highlighting a node from Figure 5.22 also highlights its all corresponding edges and their end nodes.

- 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

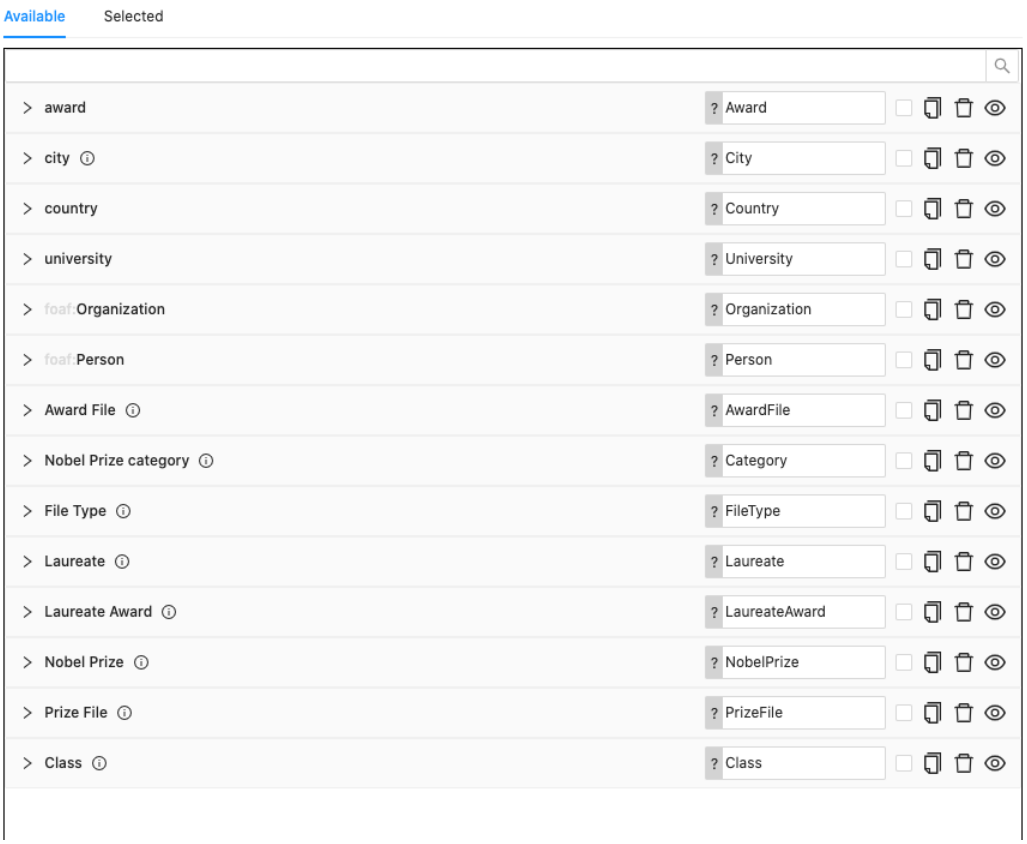
  Opens SPARQL Query editor and runs the query representing user's selection.

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



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

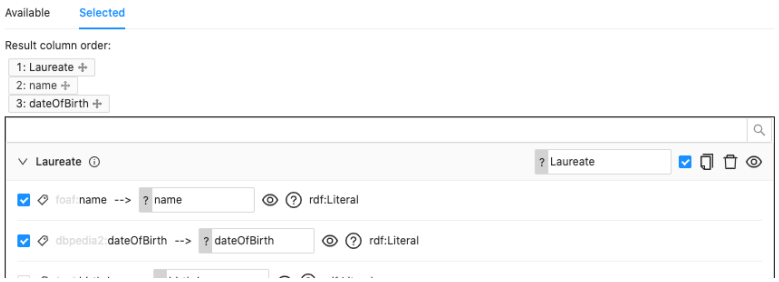
- Available tab

  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



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

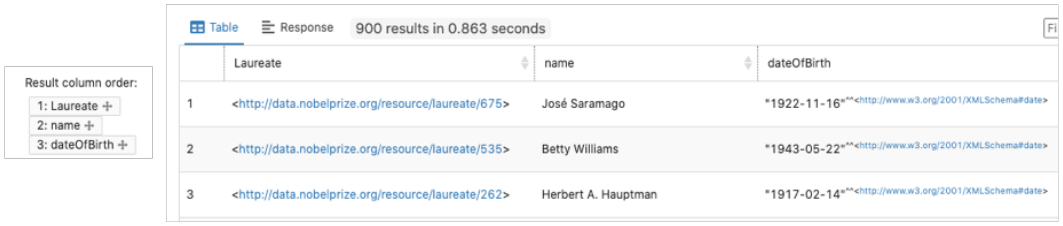
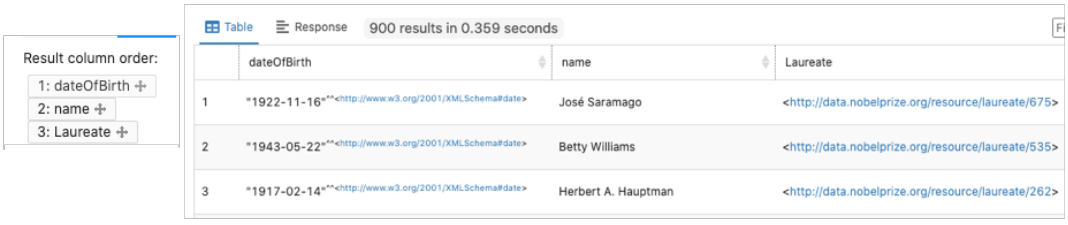


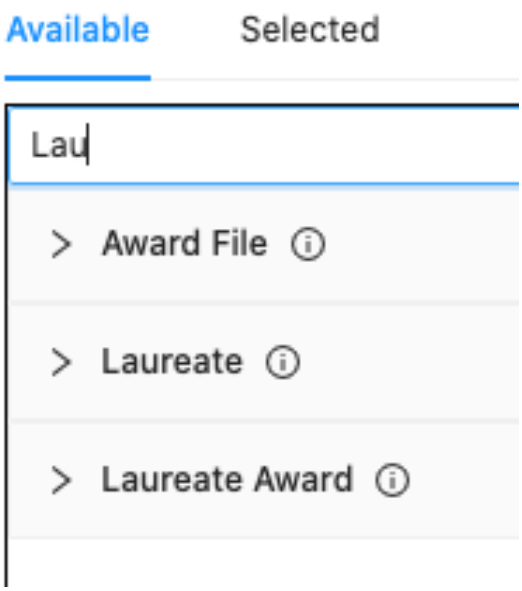
Figure 5.29: Column order example 2



- 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.

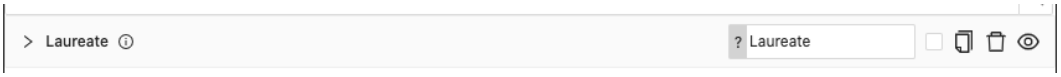
Figure 5.30: Search functionality



Entity rows

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

Figure 5.31: Entity row

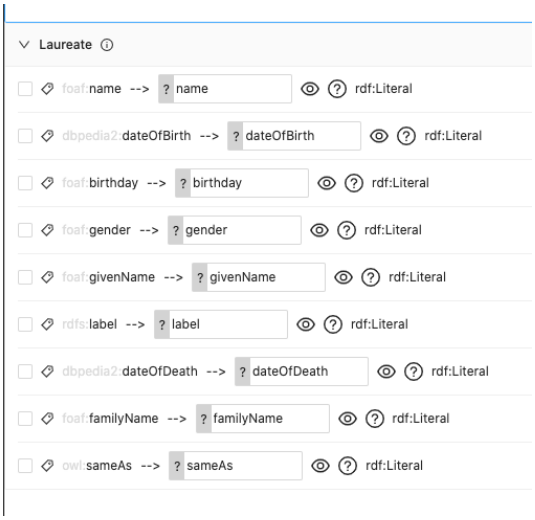


Every such row can be interacted with in the following ways:

- › Expand/Collapse icon

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

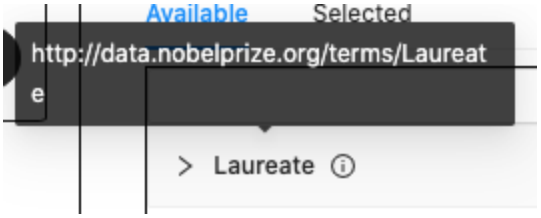
Figure 5.32: Expanded properties



- Title hover

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

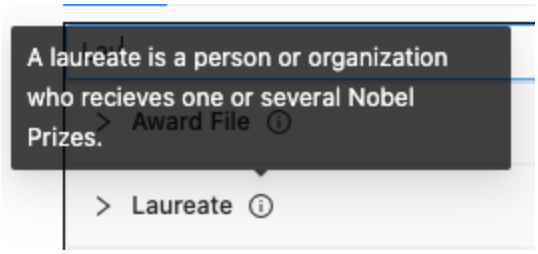
Figure 5.33: Entity title hover



- 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).

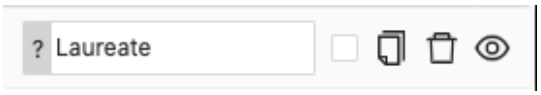
Figure 5.34: Entity description



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

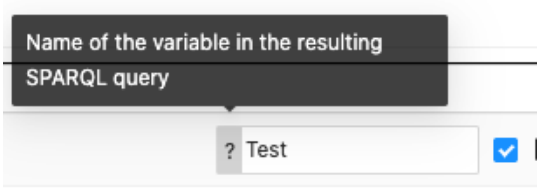
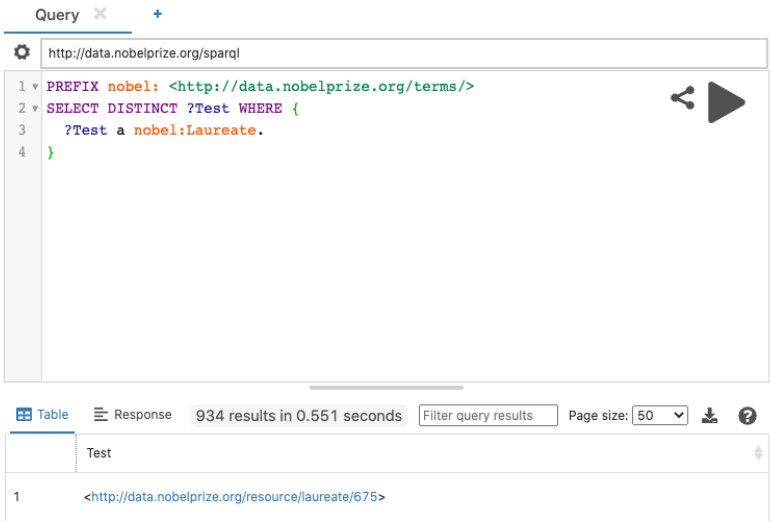


Figure 5.37: Renamed variable result



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

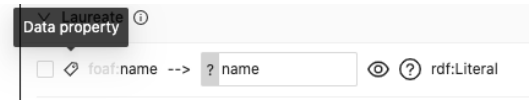
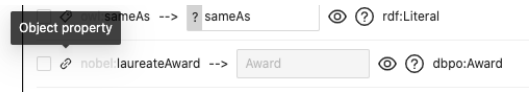


Figure 5.40: Object property row



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.

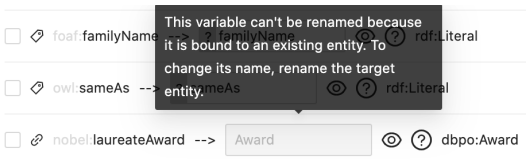
- "XXX → **?** or grey field"

  XXX represents the property's predicate.

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

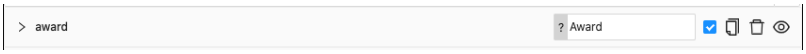
  Greyed out field is present on **object properties**, informing the user that the name of the target entity has to changed in order to change the name of this property as per the tooltip:

Figure 5.41: Object property variable field tooltip



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%3A%2F%2Fjaresan.github.io%2Fsimplod%2Fexamples%2Fnobel_prizes.ttl&endpointURL=https%3A%2F%2Fdata.nobelprize.org%2Fstore%2sparql

2. Create a New project with the following configuration:

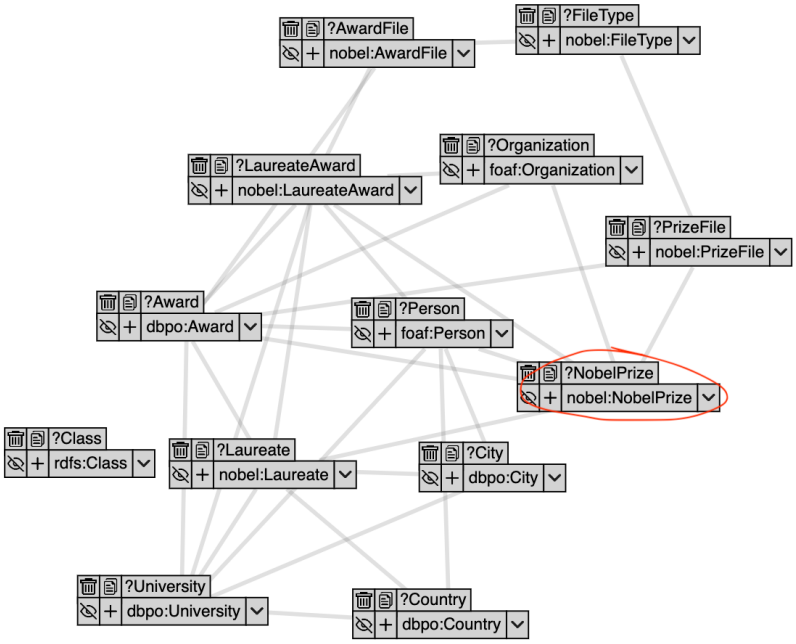
   Data Schema:

   `https://jaresan.github.io/simplod/examples/nobel_prizes.ttl`

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

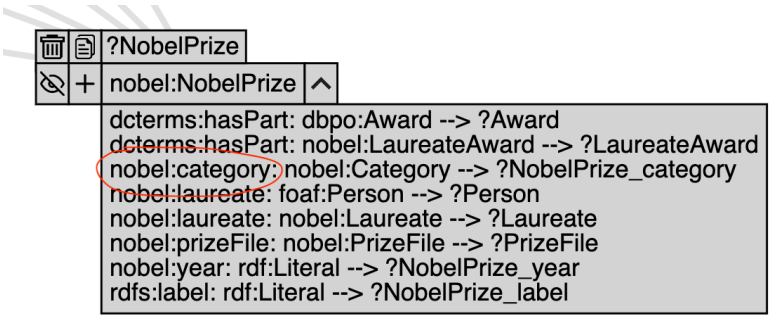
3. Click Create

Figure 5.43: Nobel prize example default view



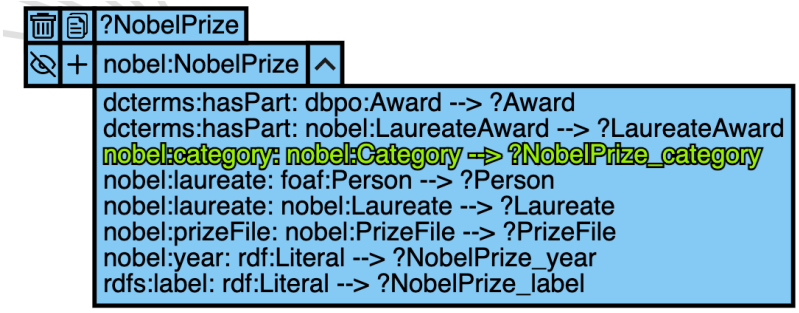
When we open the application (depicted in Figure 5.43) we can notice the "NobelPrize" entity in the graph.

Figure 5.44: NobelPrize entity



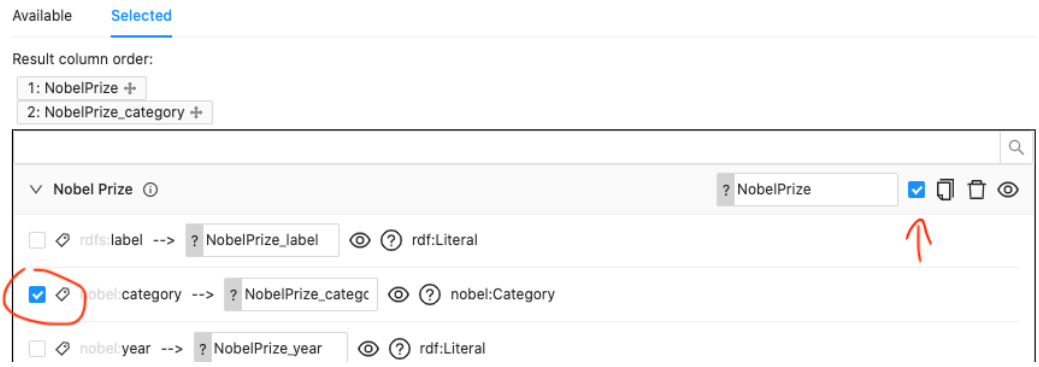
Clicking on the entity, we can expand its properties and see what it is linked to. Let's go ahead and select the **nobel:category** property.

Figure 5.45: Selected property



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



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

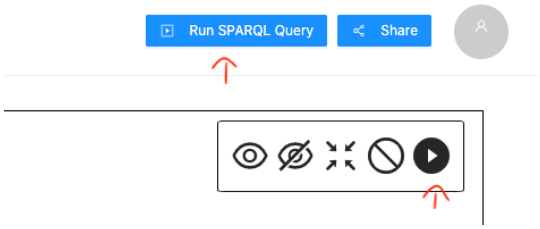
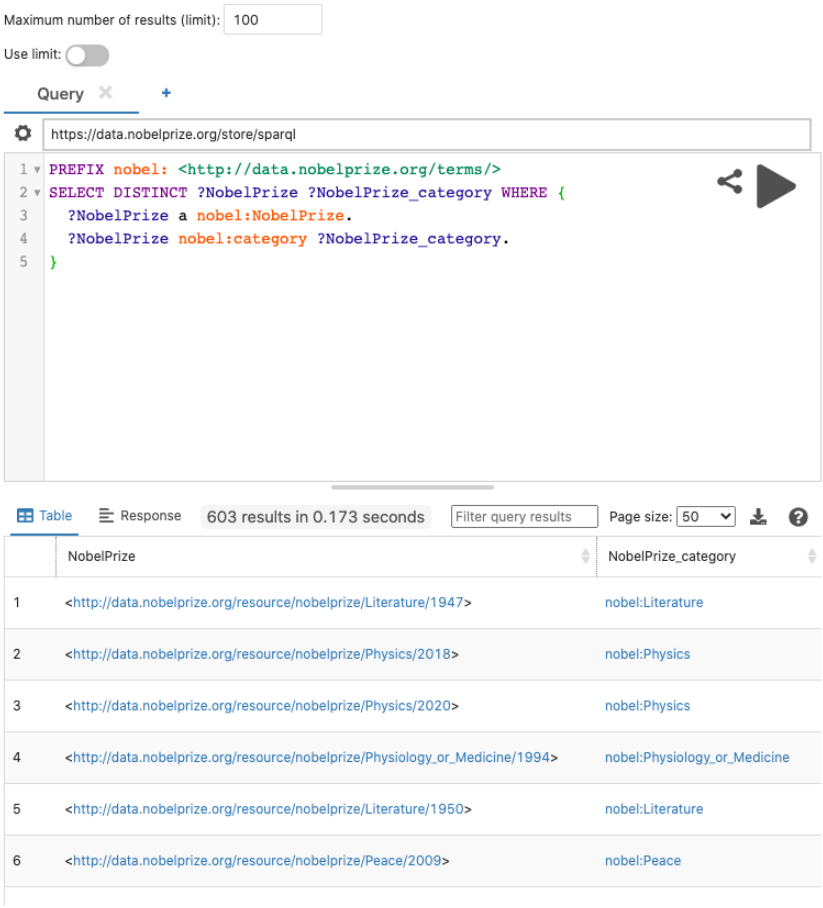


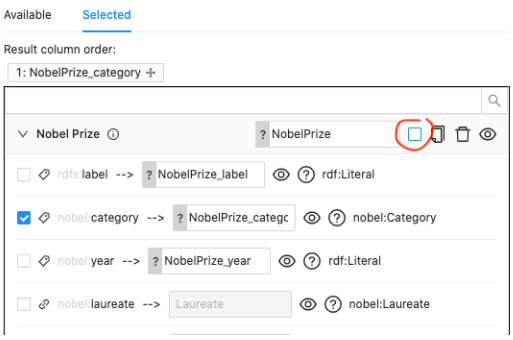
Figure 5.48: SPARQL Results



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.

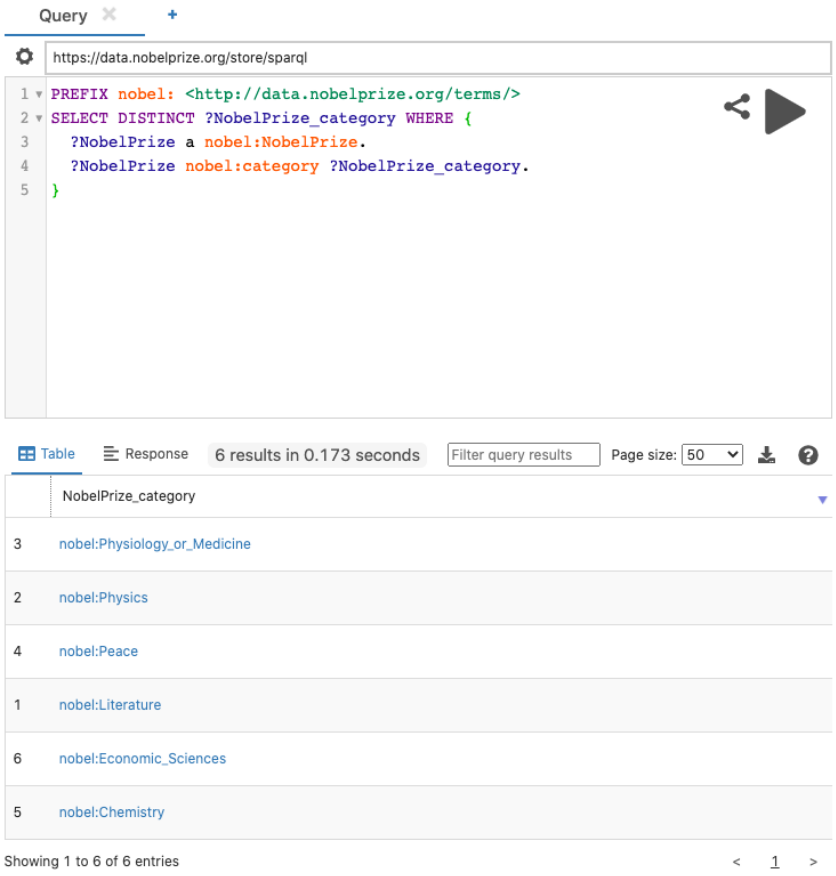
Considering we wanted to find only what categories Nobel prizes are awarded in, this result is superfluous. To get rid of the prizes, we can go back to the list view to deselect them:

Figure 5.49: List view



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

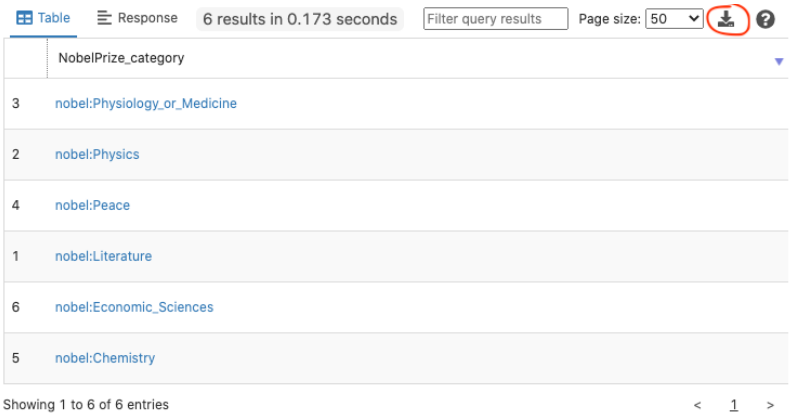
Figure 5.50: Cleaner results



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



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

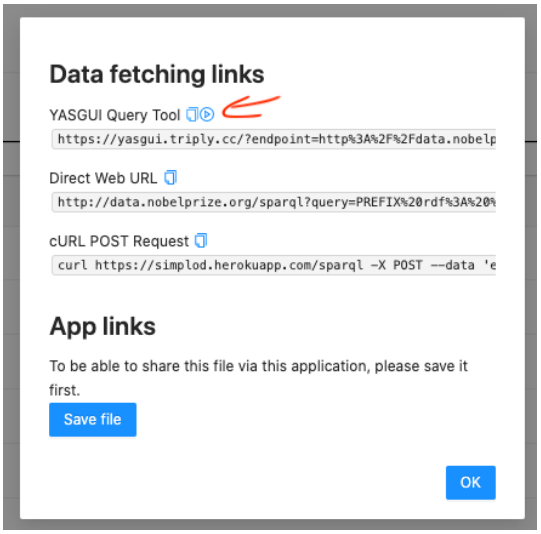
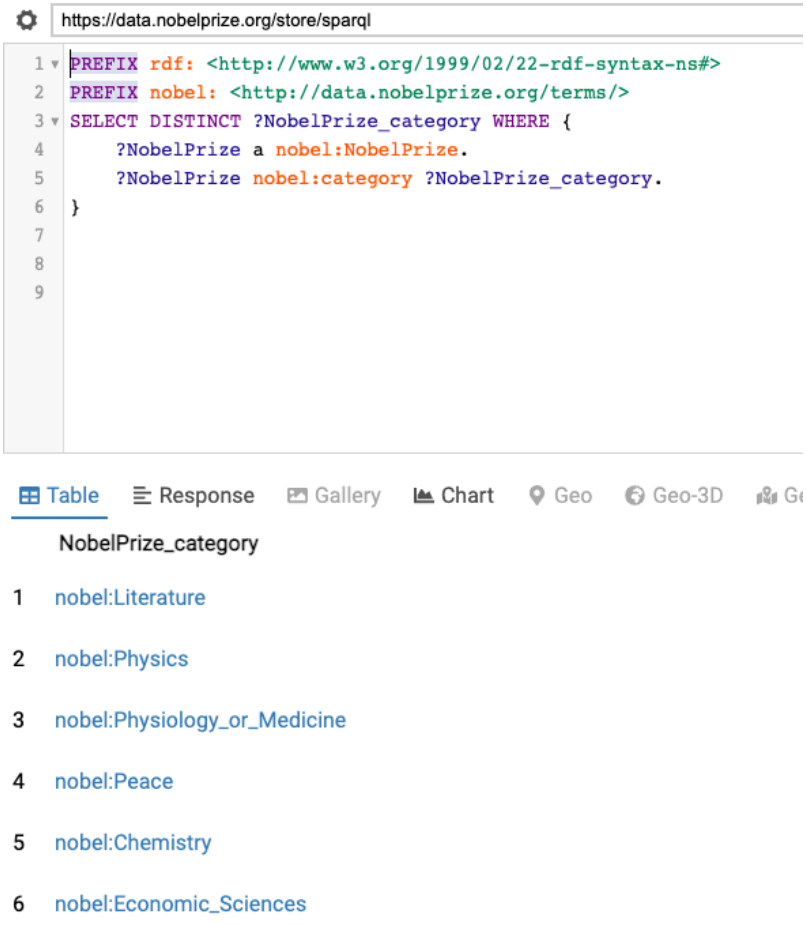


Figure 5.53: Yasgui query results



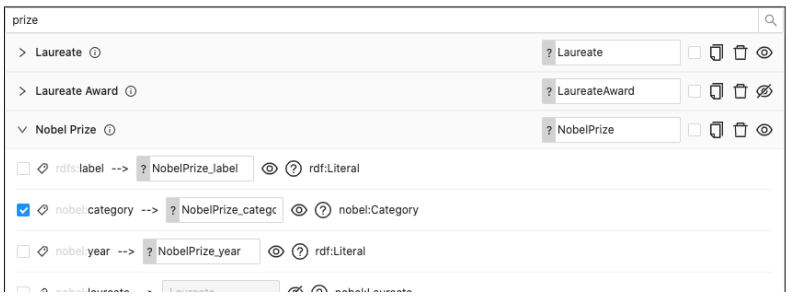
We can also use "Direct access URL" which returns the results directly, or get the cURL POST request to use in the terminal. All of the share options are described in Figure 5.1.2.

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



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:

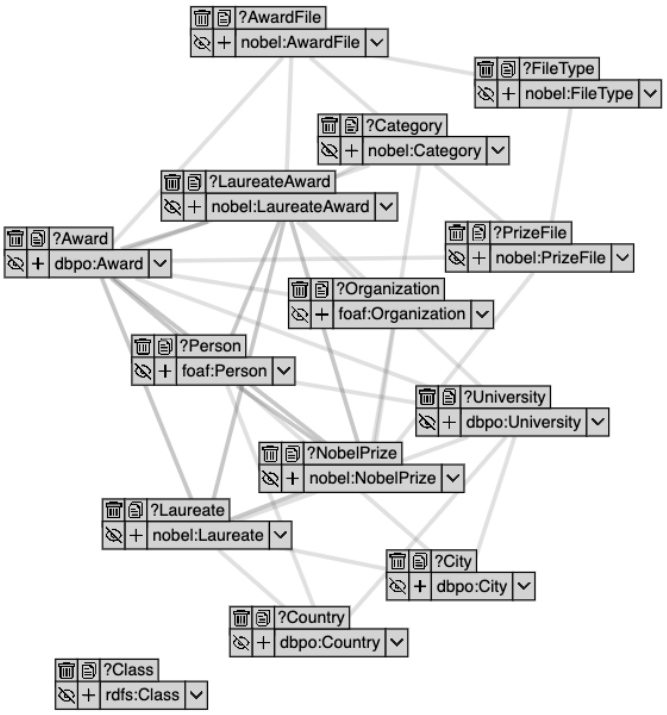
   Data Schema:

   `https://jaresan.github.io/simplod/examples/nobel_prizes.ttl`

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

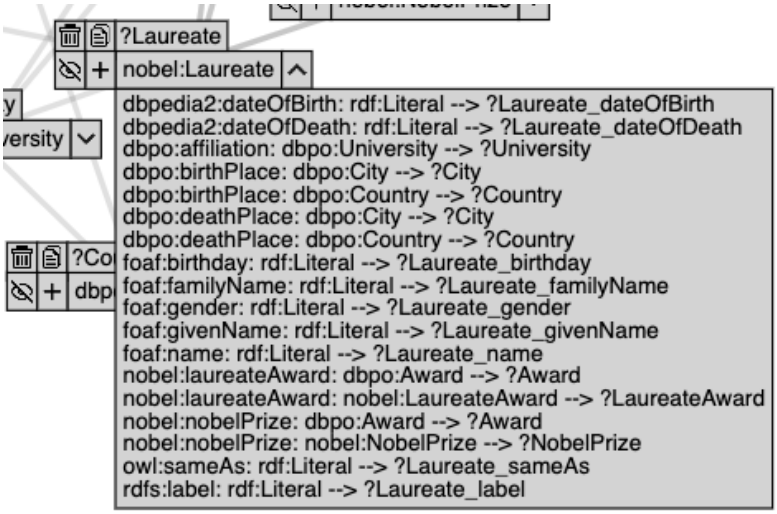
[9] `https://jaresan.github.io/simplod/demo.html?schemaURL=https%3A%2F%2Fjaresan.github.io%2Fsimplod%2Fexamples%2Fnobel_prizes.ttl&endpointURL=https%3A%2F%2Fdata.nobelprize.org%2Fstore%2Fsparql`

Figure 5.55: Nobel prize example default view



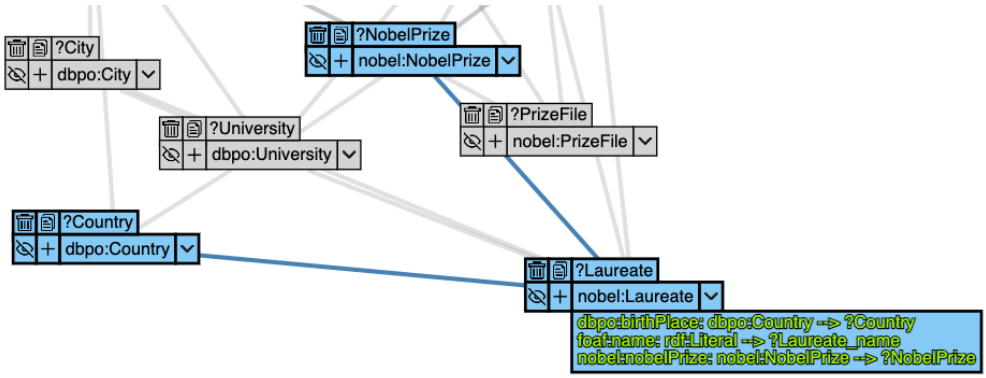
Since we are interested in nobel prize laureates, we can click the **Laureate** entity to see what relationships there are:

Figure 5.56: Laureate entity properties



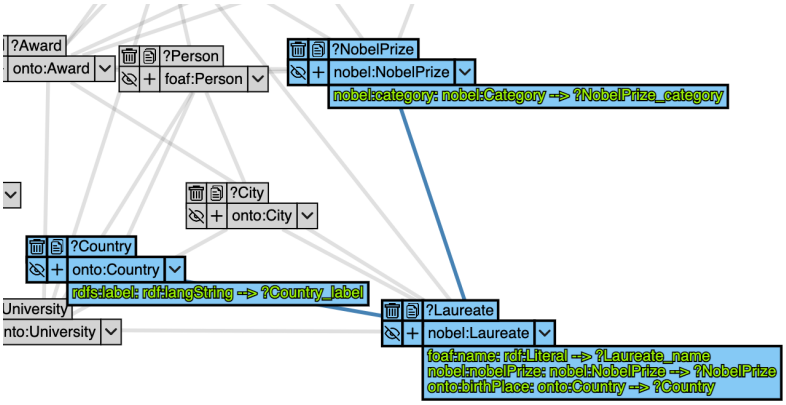
Let's say we're interested in the laureate's birth country, their name and additional information about the prize they received. If we take a look at the properties, the corresponding ones would be **dbpo:birthPlace (dbpo:Country)**, **foaf:name** and **nobel:nobelPrize (nobel:NobelPrize)** respectively.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

Figure 5.57: Laureate properties of interest



We can notice that the respective entities for **nobel:NobelPrize**, **dbpo:Country** are highlighted, 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

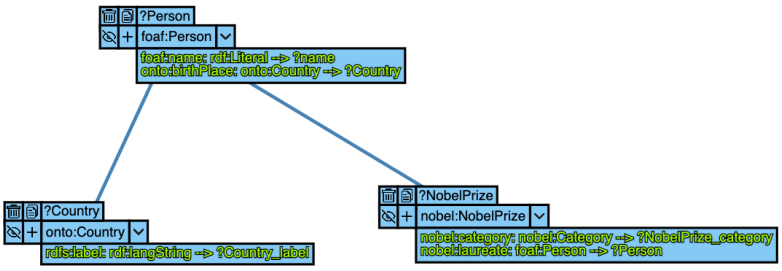


We can clean up the graph by removing data we aren't interested in. We can do this by pressing the 👁 icon in the top right of the graph, which will in turn hide all the data that is not requested.

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



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

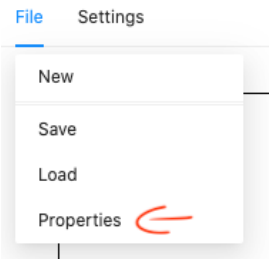
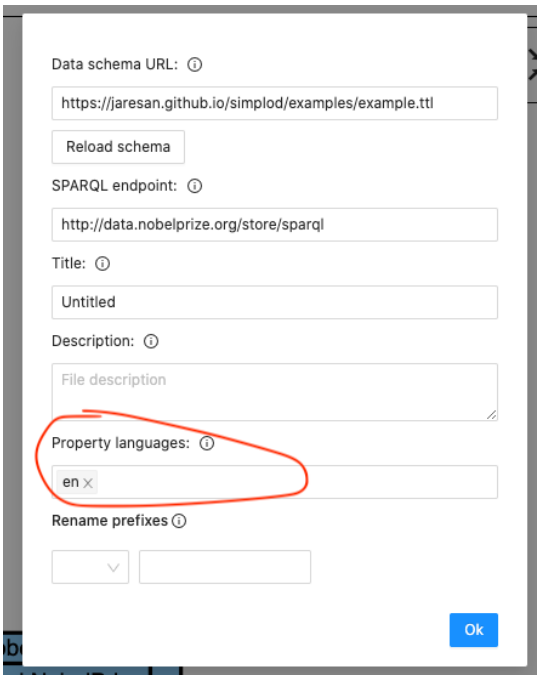
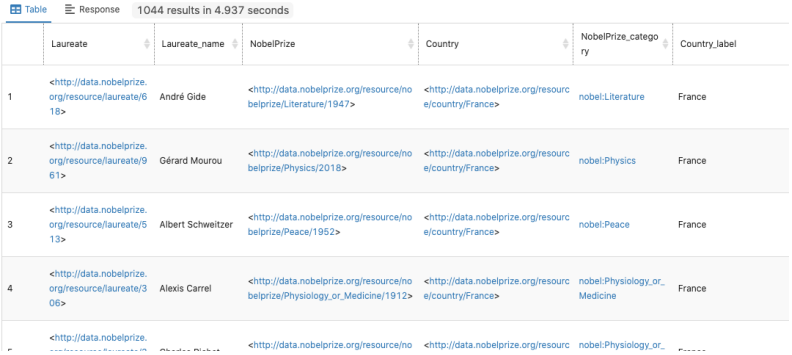


Figure 5.63: Results



This in turn ensures that all of the queried properties that are of type **rdfs:langString** will be queried as their English variant. 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



	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
5	<http://data.nobelprize.		<http://data.nobelprize.org/resource/no	<http://data.nobelprize.org/resourc	nobel:Physiology_or_	

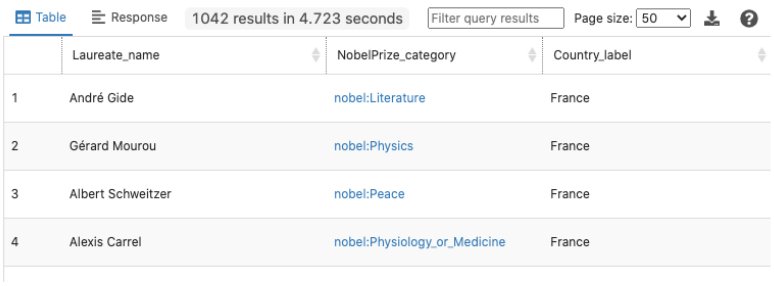
While this query is valid, a person interested in this information might not want the IRIs to be present in the result set, the textual representation provided via labels might be sufficient. To get rid of the IRIs in the result set, we have to deselect the entities via the list view as follows:

Figure 5.65: Selected entities

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

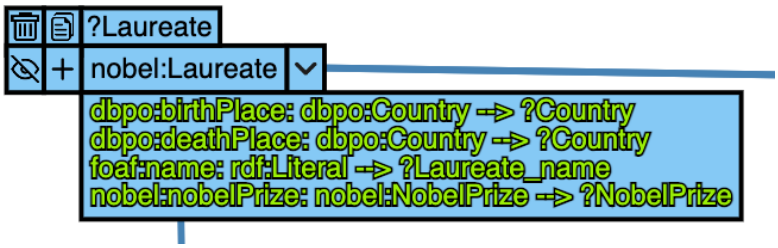
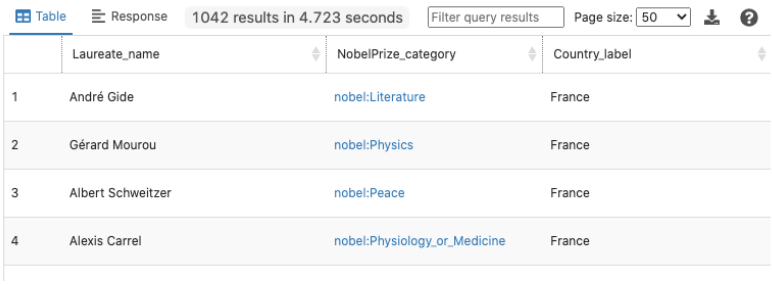


Figure 5.69: Results with place of death

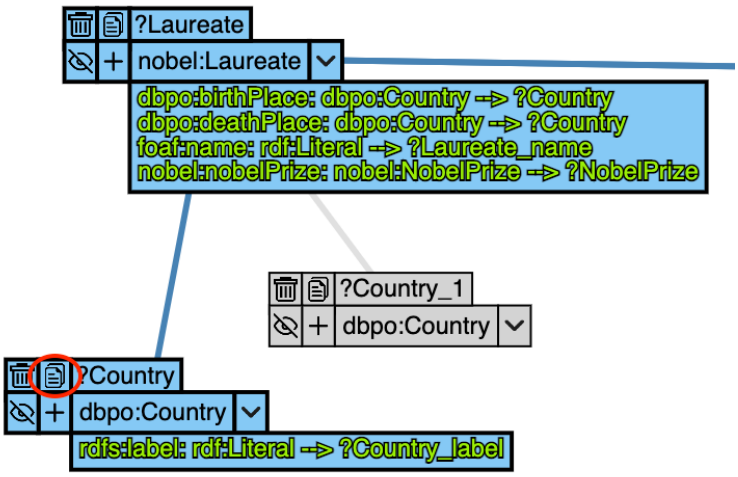


	Laureate_name	NobelPrize_category	Country_label
1	André Gide	nobel:Literature	France
2	Gérard Mourou	nobel:Physics	France
3	Albert Schweitzer	nobel:Peace	France
4	Alexis Carrel	nobel:Physiology_or_Medicine	France

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.

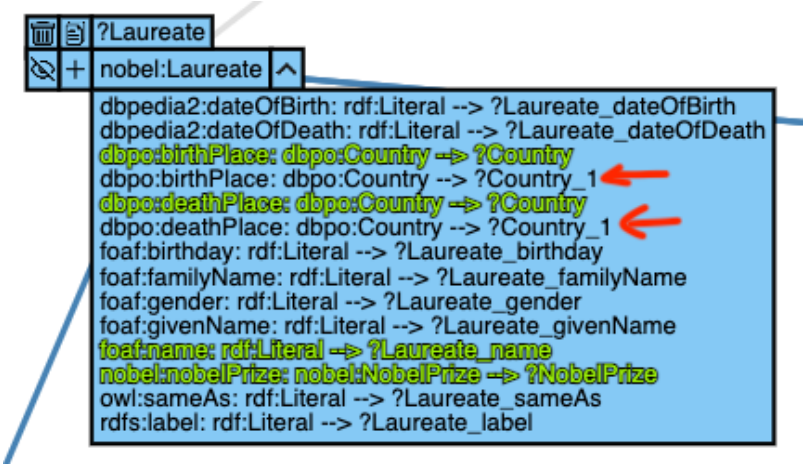
Clicking the icon on the **Country** entity, we get the following:

Figure 5.70: Selection with cloned Country entity



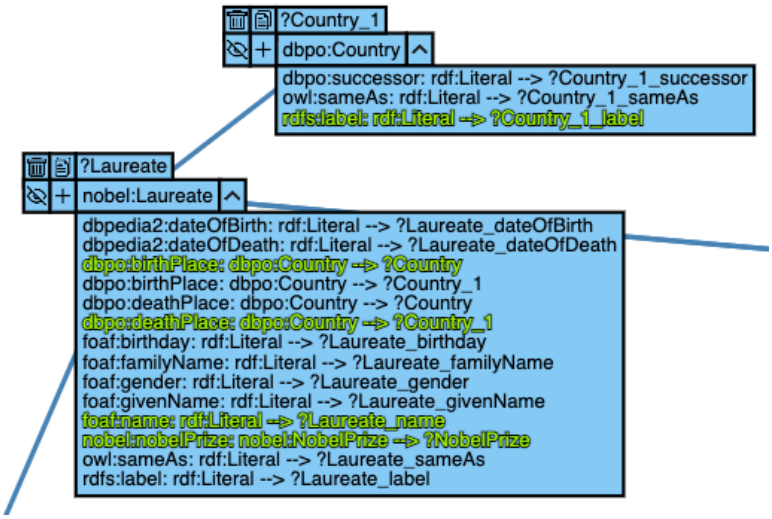
By copying the node, the **Laureate** entity has new properties added that target the new **Country** entity:

Figure 5.71: Newly listed properties



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

Figure 5.72: Proper selection



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

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].

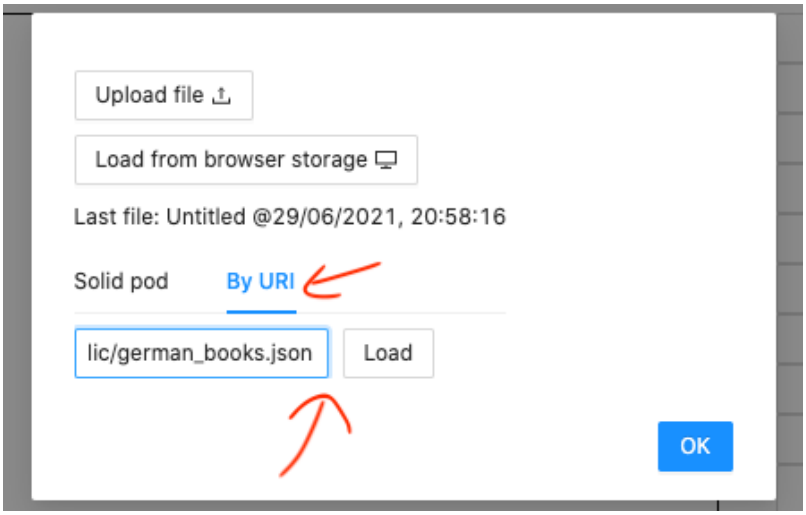
We will also be utilizing a Solid pod for data persistence. Before proceeding further, it is important we follow the steps outlined in subsection 5.1.1 and have the Solid pod set up with the necessary application permissions up correctly.

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*.

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

Figure 5.76: Load by URI

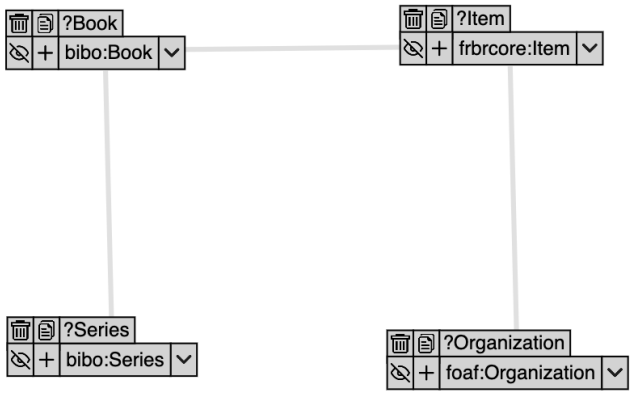


Upon loading the project file, we can see the default view for the curated book data set:

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

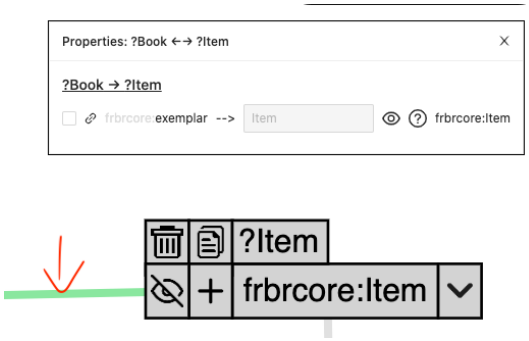
[11] `https://raw.githubusercontent.com/jaresan/simplod/master/public/german_books.ttl`

Figure 5.77: Graph loaded



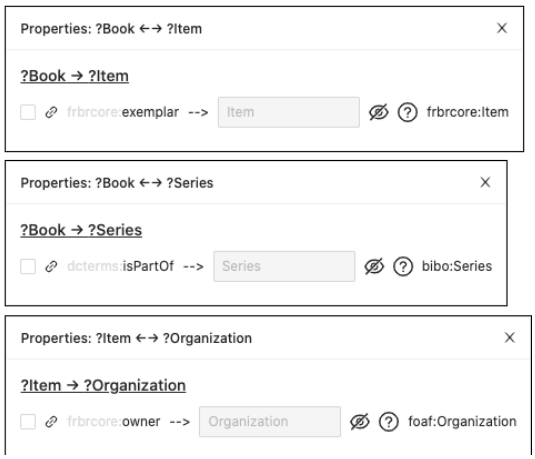
By clicking on the edges between the nodes, we can inspect the relationships they represent:

Figure 5.78: Edge descriptions



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



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

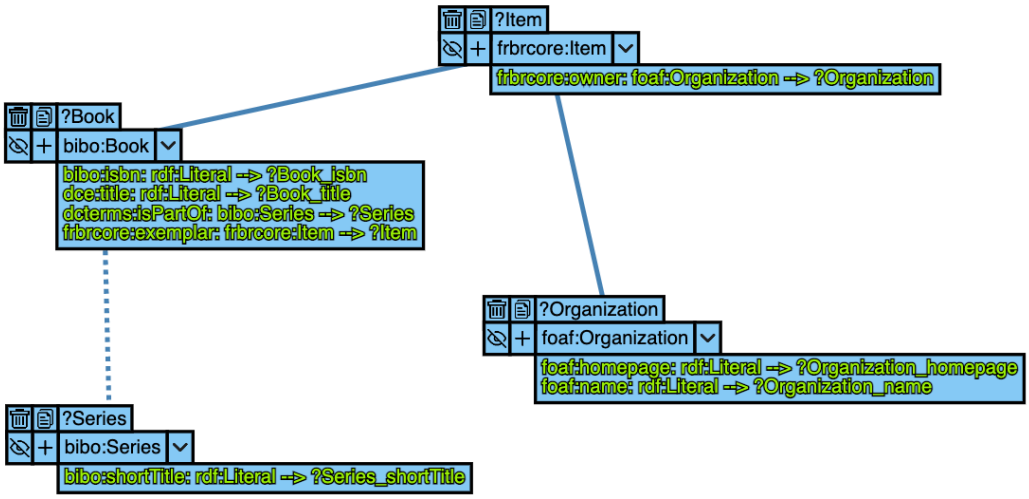
  frbrcore:owner - Organizational owner of the exemplar

- Organization

  foaf:homepage - Homepage of the organization

  foaf:name - Name of the organization

Figure 5.80: Graph selection



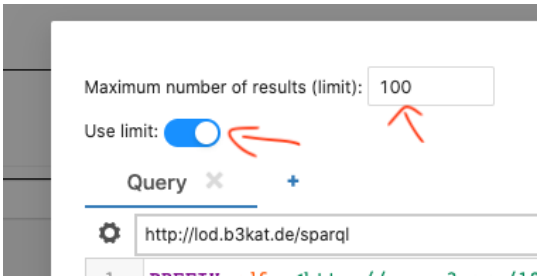
Notice the dashed edge, this means the relationship between book and its series is optional, meaning we will query for all books and return their respective

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

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



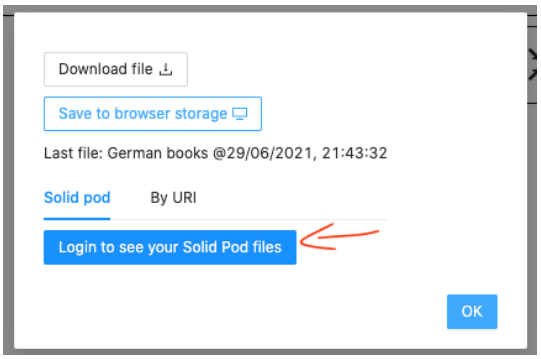
After changing the limit and running the query again by pressing the ▶, we can get the following results:

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>

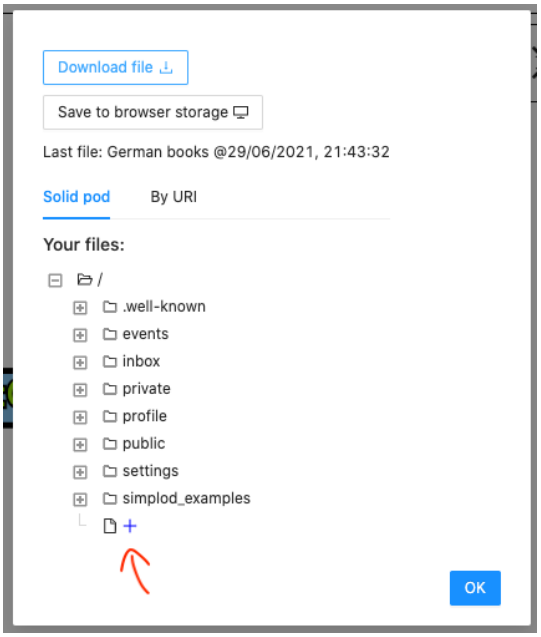
What if we want to save this result to our Solid pod? We just go to *File* → *Save*. If we are logged in, we already see our Solid pod files. If we are not logged in, we can log in either directly through the button in this menu, or through the top right avatar menu.

Figure 5.84: Save menu



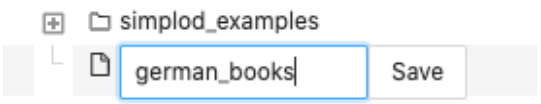
After successfully logging in, we can see our the list of our Solid pod files. We can list through the folders and in each one click the ✚ icon to save the file in that location. Let's go with the root folder and click the ✚ icon.

Figure 5.85: Solid pod files



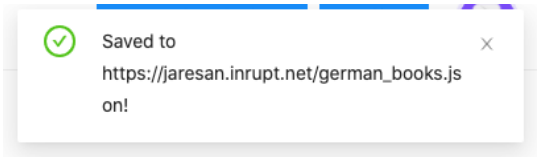
After clicking the ✚ icon, we just have to pick a file name. Let's type in *german_books* and click on save or hit enter.

Figure 5.86: New filename



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

Figure 5.87: File saved notification

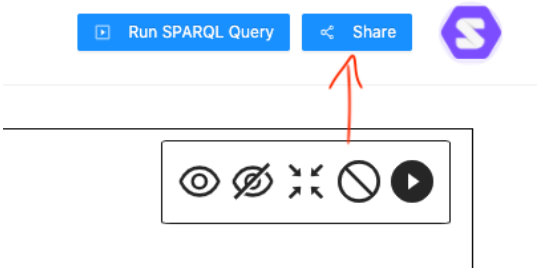


We can also see the new file location in the status bar. Hitting **ctrl+s** now saves the changes to the new remote location.

Figure 5.88: Status bar after remote save

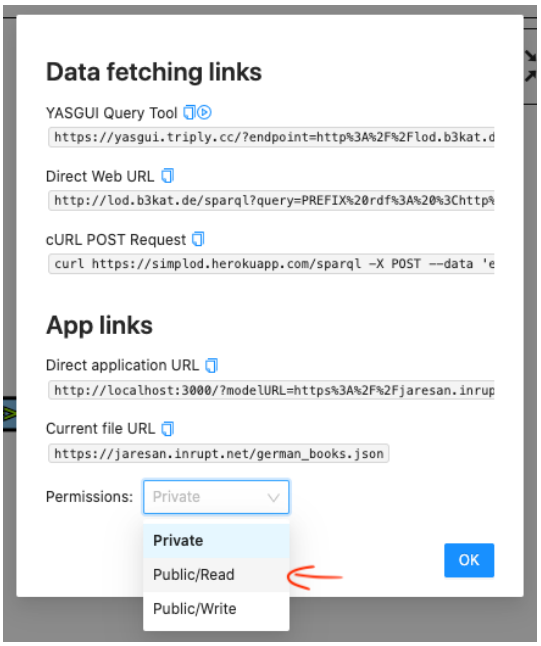
Finally, we might want to share this project file among other users. To do that, first we need to set its permission appropriately from the **Share** menu in the top-right.

Figure 5.89: Permission drop-down



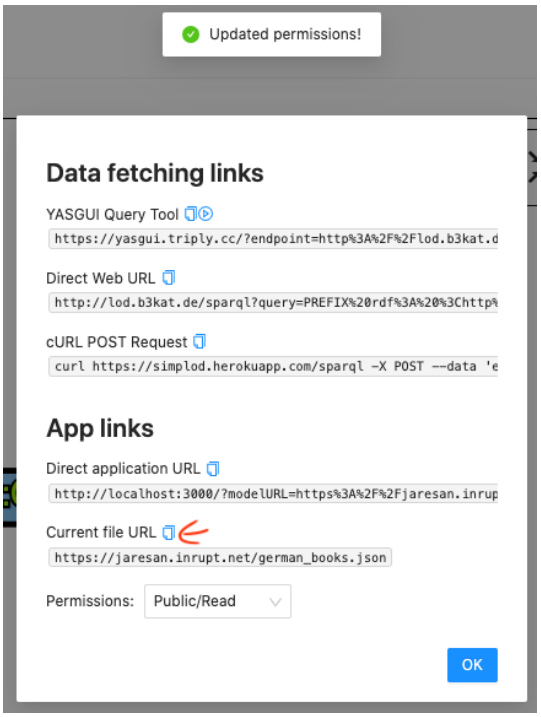
From the drop-down list, we can select **Public/read**, so that every user will be able to read this project file, but only we, as the owner, will be able to edit it.

Figure 5.90: Permission drop-down



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



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.

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 pipeline[15] created specifically for this purpose. This subsection describes how to create a data schema from an endpoint utilizing this pipeline.

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[16] 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[17]. However, in this case, the pipeline aggregates all the results into one .ttl file, which might not be desirable. To prevent this, we can always update the 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[18].

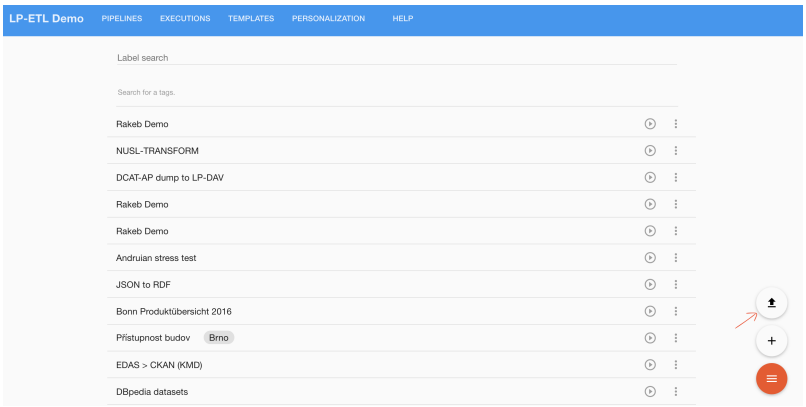
[12] https://nodejs.org/en/
[13] https://www.npmjs.com/
[14] https://yarnpkg.com/
[15] https://demo.etl.linkedpipes.com/#/pipelines
[16] https://github.com/jaresan/lod-cloud/blob/master/endpoint.ttl
[17] https://github.com/jaresan/lod-cloud/blob/master/LODCloud_SPARQL_Endpoints.ttl
[18] https://raw.githubusercontent.com/jaresan/lod-cloud/master/endpoint.ttl

Running the pipeline

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

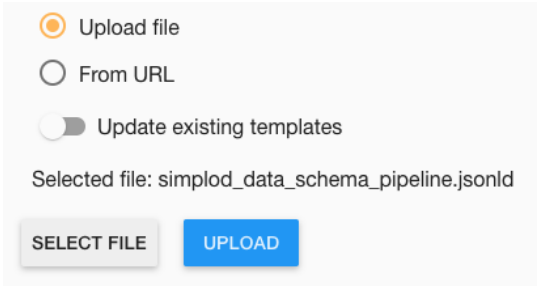
1. Access `https://demo.etl.linkedpipes.com/#/pipelines` and click on the upload button:

Figure 5.92: Upload button



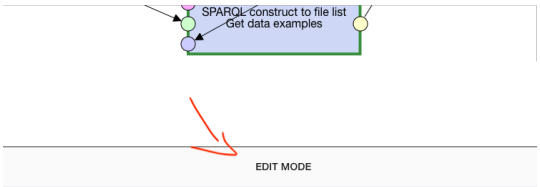
2. Provide the .jsonld file specification[19] and click "Upload":

Figure 5.93: Upload detail



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:

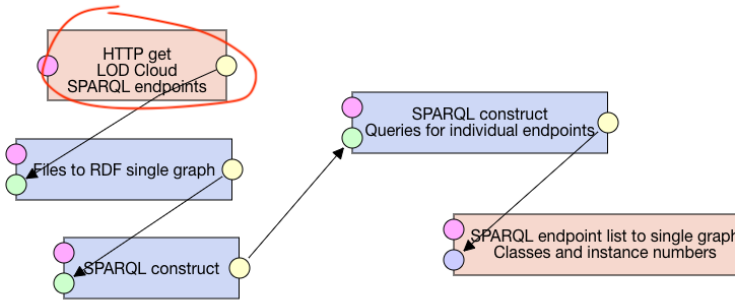
Figure 5.94: Pipeline graph



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

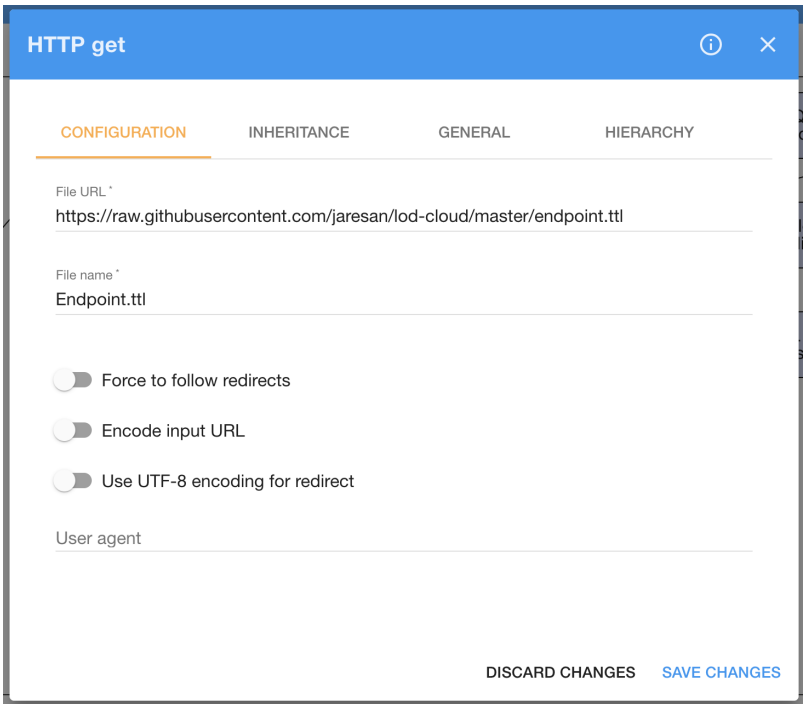
4. Double click on the first node in the graph marked as ”HTTP get” to enter its edit mode:

Figure 5.95: Pipeline start node



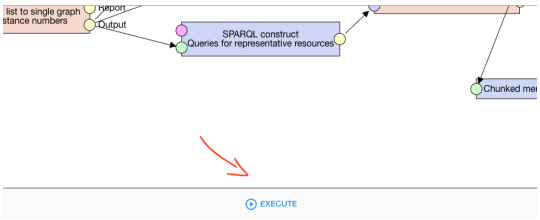
5. Change the File URL to point to your .ttl file with the endpoint specification. The filename is not important for our use case, but make sure to specify .ttl as suffix:

Figure 5.96: Start node options



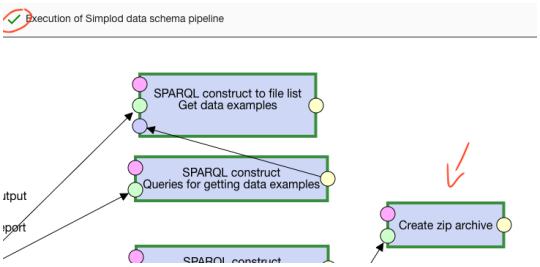
6. After the data is properly changed, click on the ”execute” button at the bottom of the graph:

Figure 5.97: Execute button



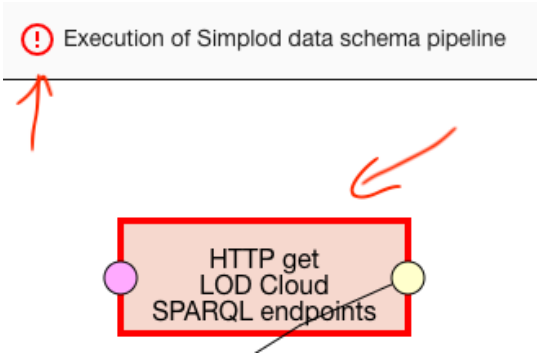
7. Wait for the execution of the pipeline to finish. The success of the run is symbolized by the status bar or by the final node's edge being green:

Figure 5.98: Execute button



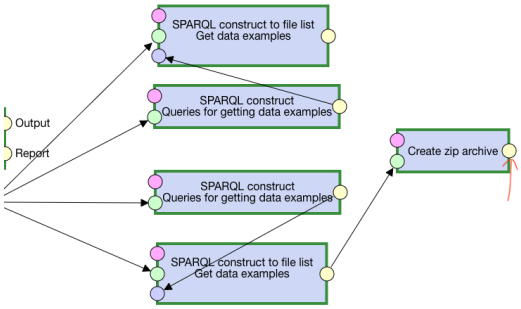
8. The process can also fail for various reasons, this fact is similarly by an icon and by the edge of the failing node being red. This work does not cover troubleshooting for failing pipeline executions.

Figure 5.99: Execute button



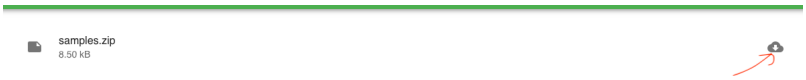
9. Click on the output (yellow) socket on the last node of the graph:

Figure 5.100: Results location



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[20].

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[21] application, that can be hosted and used as the proxy.

- Clone the repository

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

- 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

  Change `const root = 'YOUR_PROXY_PATH_HERE';` and `const useProxy = true;` in `src/@@constants/api.js`.

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.

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

5.2.4 Deployment

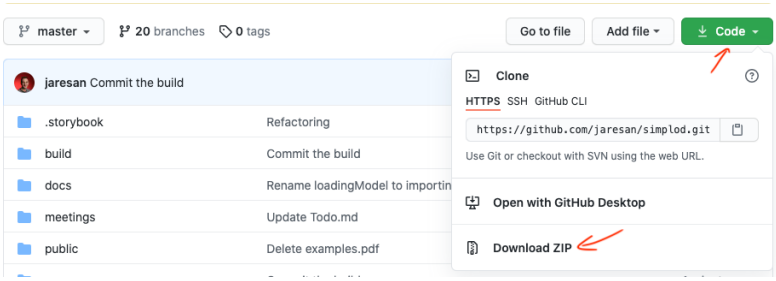
There are only few steps the administrator has to take to deploy the application. The steps are written utilizing npm[22], but the same pattern can be followed when using other package managers:

Direct build download

For convenience, the build files[23] are committed as well in the repository. The deployment steps in that case are as follows:

1. Download the repository[24] as zip

Figure 5.102: Download as zip



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

Project deployment

Since this application has been developed with Creact React App[25], the administrators can also follow the steps outlined on the React deployment page[26] 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.

[22] `https://www.npmjs.com/`
[23] `https://github.com/jaresan/simplod/tree/master/build`
[24] `https://github.com/jaresan/simplod/`
[25] `https://github.com/facebook/create-react-app`
[26] `https://create-react-app.dev/docs/deployment/`

This work also includes a heroku[27] postbuild hook[28], 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/examples/nobel_prizes.ttl`

- `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/examples/nobel_prizes.ttl`

- Endpoint

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

[27] `https://www.heroku.com/`

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

- Project

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

The link pointing to the application would have these two encoded as URL params[29] named **schemaURL** and **endpointURL**.

Utilizing JavaScript's `encodeURIComponent`[30] 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.

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`.

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

[30] `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/encodeURIComponent`

[31] `https://jaresan.github.io/simplod/build/index.html?schemaURL=https%3A%2F%2Fjaresan.github.io%2Fsimplod%2Fexamples%2Fnobel_prizes.ttl&endpointURL=https%3A%2F%2Fdata.nobelprize.org%2Fstore%2Fsparql`

[32] `https://jaresan.github.io/simplod/build/index.html?modelURL=https%3A%2F%2Fjaresan.inrupt.net%2Fgerman_books.json`

[33] `https://nodejs.org/en/`

[34] `https://www.npmjs.com/`

[35] `https://yarnpkg.com/`

- 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.

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

  The complete application information that can be exported/imported. 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.

[36] `https://simplod.herokuapp.com`

- 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.

- @@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:

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

  - 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.

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))
  ));
```

[38] https://jsdoc.app/

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

```
  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.

[40] https://github.com/jaresan/simplod/pull/28

[41] https://g6.antv.vision/en/docs/api/Items/itemMethods

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

    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