Creating a Semantic Portal using the Sampo-UI framework

This tutorial will teach you how to quickly create a web application with faceted search and analytic tools on an existing SPARQL endpoint. We will use DBpedia as an example case.

SETUP

First you need to clone the Sampo-UI repository on Github:
https://github.com/SemanticComputing/sampo-ui

Then set up the requirements:

- Node.js® – a JavaScript runtime built on Chrome's V8 JavaScript engine. (version 16.13.0)
- Nodemon – monitor for any changes in your source and automatically restart your server

NOTE: It is highly recommended that you use the Node Version Manager for installing Node.js!

INSTALLATION

Install the dependencies specified in `package.json` (this command needs to be run only once, as long as you don't modify the dependencies):

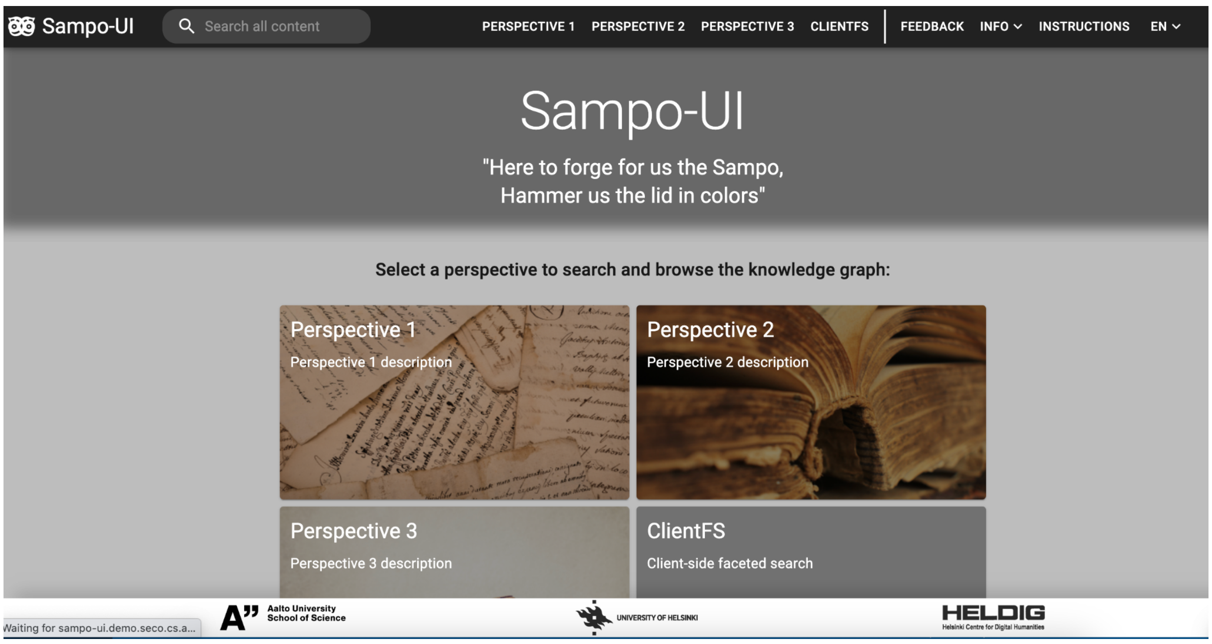
```
npm install
```

RUNNING THE APPLICATION LOCALLY

Run client and server concurrently:

```
npm run dev
```

The application will be available on localhost:8080 and by navigating there with a browser you should see something like this:


Sampo-UI
Search all content
PERSPECTIVE 1
PERSPECTIVE 2
PERSPECTIVE 3
CLIENTFS
FEEDBACK
INFO
INSTRUCTIONS
EN
Sampo-UI
"Here to forge for us the Sampo,
Hammer us the lid in colors"
Select a perspective to search and browse the knowledge graph:
Perspective 1
Perspective 1 description
Perspective 2
Perspective 2 description
Perspective 3
Perspective 3 description
ClientFS
Client-side faceted search
Aalto University School of Science
UNIVERSITY OF HELSINKI
HELDIG
Waiting for sampo-ui.demo.seco.cs.a...

You can now make changes to the code in the repository. Nodemon should automatically update the application when you make changes to the source code on the frontend, but to see backend changes requires refreshing the browser.

You can use any editor for making changes to the code. We recommend Visual Studio Code, but you can pick your own favorite.

QUICKSTART TUTORIAL

In this quickstart tutorial we'll be setting up a new Sampo portal by just editing the files that already exist in the base Sampo-UI framework. If you want to follow along a more detailed configuration tutorial that involves setting up new files, you can move to the next section.

First navigate to the perspective1.json configuration file in the directory of /src/configs/sampo/search_perspectives/. This is the configuration file for the first perspective we see on the landing page, Perspective 1. This will be the file that we will mainly be editing in this tutorial.

We’ll first need to edit the endpoint to the DBpedia endpoint by changing the value of the attribute ‘url’ in the ‘endpoint’ object to https://dbpedia.org/sparql:

```
…
"endpoint": {
        "url": "https://dbpedia.org/sparql",
        "useAuth": false,
        "prefixesFile": "SparqlQueriesPrefixes.js"
},
…
```

Let’s make our lives easier by defining the ‘dbr’ prefix to refer to http://dbpedia.org/resource/ and ‘dbo’ to http://dbpedia.org/ontology/. This is done by adding a new line to the SparqlQueriesPrefixes.js file located in the portal’s SPARQL queries directory /src/server/sparql/sampo/sparql_queries/:

```
export const prefixes = `
  PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
  …
  PREFIX semparls: <http://ldf.fi/schema/semparl/>
  PREFIX dbr: <http://dbpedia.org/resource/>
  PREFIX dbo: <http://dbpedia.org/ontology/>
`
```

This way we can use this prefix in our queries. Let’s define our facet class for this perspective to be DBpedia’s Writer class through the ‘facetClass’ attribute:

```
…
"sparqlQueriesFile": "SparqlQueriesPerspective1.js",
"baseURI": "http://ldf.fi/mmm",
"URITemplate": "<BASE_URI>/work/<LOCAL_ID>",
"facetClass": "dbo:Writer",
…
```

If we hadn’t added the prefix to the list of prefixes, we could’ve spelled out the whole URI here instead.

Now let’s edit the base URI and template to match the one used in DBpedia. For example, Oscar Wilde’s URI in DBpedia is http://dbpedia.org/resource/Oscar_Wilde. We want to take the part ‘Oscar_Wilde’ as the local URI to be used in our URLs. In this case the base URI would then be ‘http://dbpedia.org/resource/’ and the template becomes <BASE_URI><LOCAL_ID>:

```
…
"sparqlQueriesFile": "SparqlQueriesPerspective1.js",
"baseURI": "http://dbpedia.org/resource/",
"URITemplate": "<BASE_URI><LOCAL_ID>",
"facetClass": "dbo:Writer",
…
```

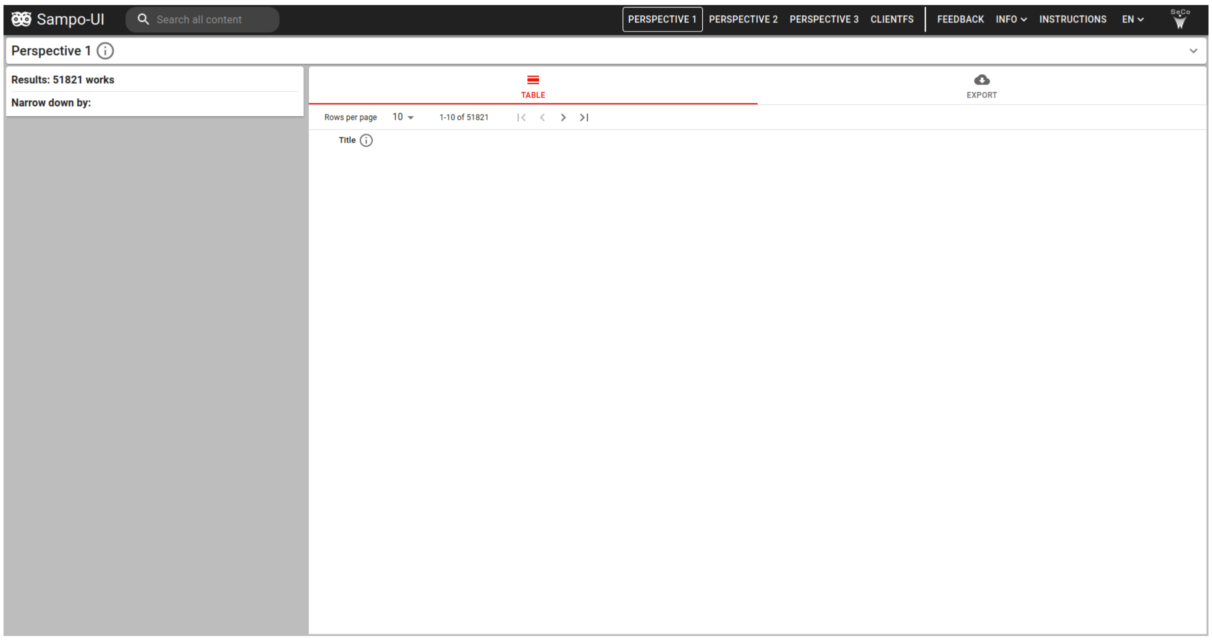
Let’s remove some extra properties and facets for now. Remove all the other properties objects from the properties object apart from the URI (‘uri’) and preferred label (‘prefLabel’):

```
…
"properties": [
	{
		"id": "uri",
		"valueType": "object",
		"makeLink": true,
		"externalLink": true,
		"sortValues": true,
		"numberedList": false,
		"onlyOnInstancePage": true
	},
	{
		"id": "prefLabel",
		"valueType": "object",
		"makeLink": true,
		"externalLink": false,
		"sortValues": true,
		"numberedList": false,
		"minWidth": 250
	}
],
…
```

Remove all the facet objects from the ‘facets’ object except for the preferred label (‘prefLabel’). Inside the preferred label object remove all attributes except for the ‘sortByPredicate’ attribute. Change the value of ‘sortByPredicate’ to ‘rdfs:label’ as this is the label property used in DBpedia:

```
…
"facets": {
	"prefLabel": {
		"sortByPredicate": "rdfs:label"
	}
}
…
```

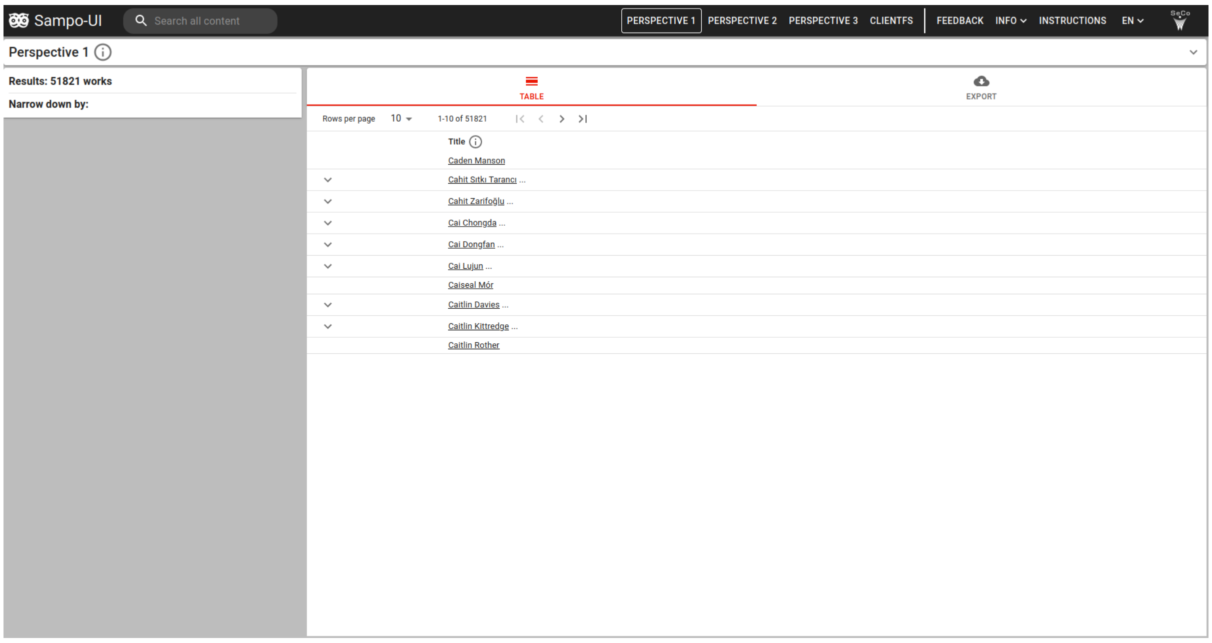
Now, if we check the Perspective 1 view running on localhost by clicking on the Perspective 1 card on the landing page, we can see that the table view is empty, only has one column ‘Title’ and no facets. We’ll need to edit the query file next.



Open the SparqlQueriesPerspective1.js query file located in the directory /src/server/sparql/sampo/sparql_queries/. Remove all query blocks in the query inside the 'workProperties' variable except for the first block and change the 'skos:prefLabel' to 'rdfs:label' here as well:

```
export const workProperties = `
        {
                ?id rdfs:label ?prefLabel__id .
                BIND(?prefLabel__id AS ?prefLabel__prefLabel)
                BIND(CONCAT("/${perspectiveID}/page/", REPLACE(STR(?id), "^.*\\\\/(.+)", "$1"))
AS ?prefLabel__dataProviderUrl)
                BIND(?id as ?uri__id)
                BIND(?id as ?uri__dataProviderUrl)
                BIND(?id as ?uri__prefLabel)
        }
`
```

You'll have to manually refresh the page in your browser every time you make changes to the queries as they are part of the backend. You should now be able to see a list of names in the table view.



Results: 51821 works

Narrow down by:

Rows per page 10 1-10 of 51821

Title
Caden Manson
Cahit Sıtkı Tarancı ...
Cahit Zarifoğlu ...
Cai Chongda ...
Cai Dongfan ...
Cai Lujun ...
Caiseal Mór
Caitlin Davies ...
Caitlin Kittredge ...
Caitlin Rother

Now we can start to expand the properties included in the table view. Let’s add three more properties about the writers: the genre(s) they wrote, their occupation(s) and their alma mater. In DBpedia these properties are denoted with the predicates dbo:genre, dbo:occupation and dbo:almaMater respectively.

Let’s first define the configurations for them in the perspective configuration file. All of the properties receive their values as objects:

```
…
"properties": [
        {
                "id": "uri",
                "valueType": "object",
                "makeLink": true,
                "externalLink": true,
                "sortValues": true,
                "numberedList": false,
                "onlyOnInstancePage": true
        },
        …
        {
                **"id": "genre",**
                **"valueType": "object",**
                "makeLink": false,
                "externalLink": false,
                "sortValues": true,
                "numberedList": false,
                "minWidth": 250
        },
        {
                **"id": "occupation",**
```

```
                    "valueType": "object",
                    "makeLink": true,
                    "externalLink": false,
                    "sortValues": true,
                    "numberedList": false,
                    "minWidth": 250
          },
          {
                    "id": "almaMater",
                    "valueType": "object",
                    "makeLink": true,
                    "externalLink": false,
                    "sortValues": true,
                    "numberedList": false,
                    "minWidth": 250
          }
],
…
```

Now we have to write the queries for these properties. Pay attention to the IDs you give the properties: you have to use the same name for the variables in the queries.

Let’s first write the query for the genres. Since the genre property has objects as its values, we need to first capture the ID of the genre entity under ‘id’. To capture this, you need to store it in a variable with the name format of ‘?[PROPERTY NAME]__id’ (e.g., ?genre__id in this case) with two underscores between. The label of the entity should be captured similarly under ‘prefLabel’, e.g. ?genre__prefLabel. Lastly, since we’re only interested in the English labels of these genres, we filter the language of the label to only English:

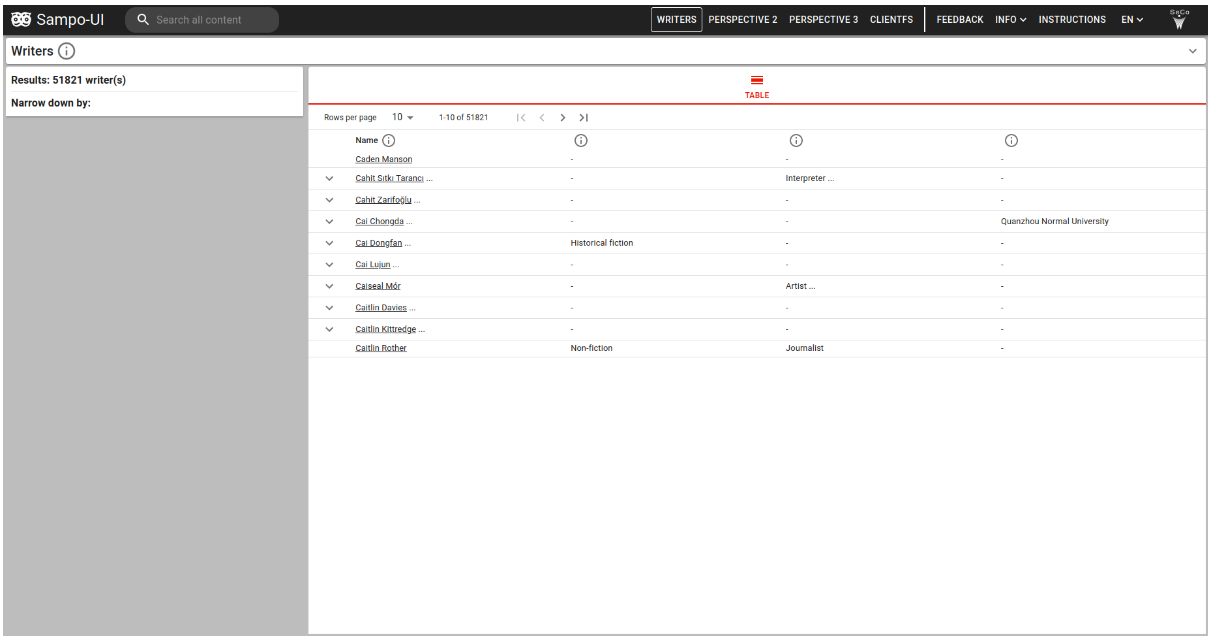
```
export const workProperties = `
          {
                    ?id rdfs:label ?prefLabel__id .
                    …
          }
          UNION
          {
                    ?id dbo:genre ?genre__id .
                    ?genre__id rdfs:label ?genre__prefLabel .
                    FILTER(LANG(?genre__prefLabel) = 'en')
          }
`
```

Now we write the queries for occupations and alma mater:

```
export const workProperties = `
          {
                    ?id rdfs:label ?prefLabel__id .
                    …
```

```
        }
        UNION
        {
                ?id dbo:genre ?genre__id .
                ?genre__id rdfs:label ?genre__prefLabel .
                FILTER(LANG(?genre__prefLabel) = 'en')
        }
        UNION
        {
                ?id dbo:occupation ?occupation__id .
                ?occupation__id rdfs:label ?occupation__prefLabel .
                FILTER(LANG(?occupation__prefLabel) = 'en')
        }
        UNION
        {
                ?id dbo:almaMater ?almaMater__id .
                ?almaMater__id rdfs:label ?almaMater__prefLabel .
                FILTER(LANG(?almaMater__prefLabel) = 'en')
        }
`
```

If we refresh the page now, we should be seeing some values for these properties in the table.



Results: 51821 writer(s)

Narrow down by:

Rows per page 10 1-10 of 51821

Name			
Caden Manson	-	-	-
Cahit Sıtkı Tarancı ...	-	Interpreter ...	-
Cahit Zarifoğlu ...	-	-	-
Cai Chongda ...	-	-	Quanzhou Normal University
Cai Dongfan ...	Historical fiction	-	-
Cai Lujun ...	-	-	-
Caiseal Mór	-	Artist ...	-
Caitlin Davies ...	-	-	-
Caitlin Kittredge ...	-	-	-
Caitlin Rother	Non-fiction	Journalist	-

Let's now add the facets for filtering the data based on these same properties. We'll start with the facet for genres:

```
…
"facets": {
        "prefLabel": {
                "sortByPredicate": "rdfs:label"
        },
        "genre": {
                "containerClass": "ten",
                "facetType": "list",
                "filterType": "uriFilter",
                "facetLabelPredicate": "rdfs:label",
                "facetLabelFilter": "FILTER(LANG(?prefLabel_) = 'en')",
                "predicate": "dbo:genre",
                "searchField": true,
                "sortButton": true,
                "sortBy": "instanceCount",
                "sortByPredicate": "dbo:genre/rdfs:label",
                "sortDirection": "desc"
        }
}
…
```

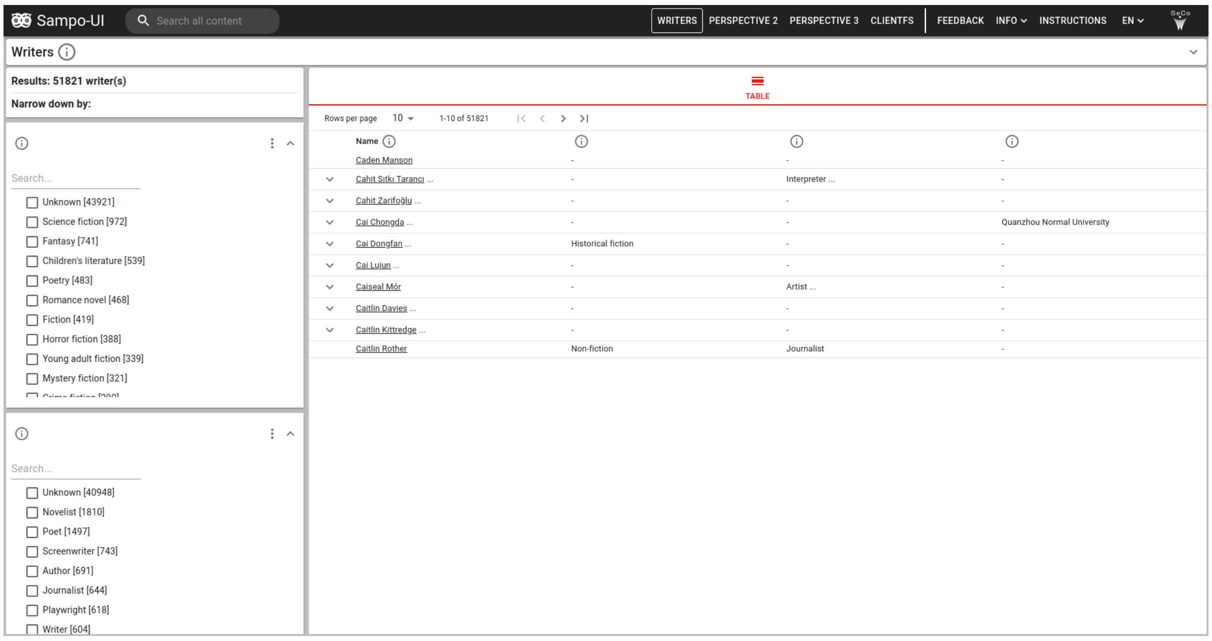
Since the values of genres are objects, we want to use the ‘uriFilter’ filter type and ‘list’ facet type to get a facet with checkboxes. The ‘containerClass’ attribute determines the height of the facet, ‘ten’ is usually used for checkbox facets and ‘four’ for text search facets.

To get the correct values for the facet, we need to define the predicate using the ‘predicate’ attribute. Since DBpedia uses rdfs:label instead of skos:prefLabel that is used by default for labels in the Sampo-UI framework, we need to use ‘facetLabelPredicate’ to specify the predicate used for labels. We also need to add the ‘facetLabelFilter’ attribute to just get the English labels.

We can now add the rest of the facets in a similar way:

```
…
"facets": {
        "prefLabel": {
                "sortByPredicate": "rdfs:label"
        },
        "genre": {
                "containerClass": "ten",
                …
        },
        "occupation": {
                "containerClass": "ten",
                "facetType": "list",
                "filterType": "uriFilter",
                "facetLabelPredicate": "rdfs:label",
                "facetLabelFilter": "FILTER(LANG(?prefLabel_) = 'en')",
                "predicate": "dbo:occupation",
                "searchField": true,
                "sortButton": true,
                "sortBy": "instanceCount",
                "sortByPredicate": "dbo:occupation/rdfs:label",
                "sortDirection": "desc"
        },
        "almaMater": {
                "containerClass": "ten",
                "facetType": "list",
                "filterType": "uriFilter",
                "facetLabelPredicate": "rdfs:label",
                "facetLabelFilter": "FILTER(LANG(?prefLabel_) = 'en')",
                "predicate": "dbo:almaMater",
                "searchField": true,
                "sortButton": true,
                "sortBy": "instanceCount",
                "sortByPredicate": "dbo:almaMater/rdfs:label",
                "sortDirection": "desc"
        }
}
…
```

If you refresh the page, we should now have a table with four columns and three facets in the facet menu. We're almost ready with our new simple Sampo portal! Now we just need to add the localizations to add the labels for the new columns and facets.



Results: 51821 writer(s)

Narrow down by:

- Unknown [43921]
- Science fiction [972]
- Fantasy [741]
- Children's literature [539]
- Poetry [483]
- Romance novel [468]
- Fiction [419]
- Horror fiction [388]
- Young adult fiction [339]
- Mystery fiction [321]

- Unknown [40948]
- Novelist [1810]
- Poet [1497]
- Screenwriter [743]
- Author [691]
- Journalist [644]
- Playwright [618]
- Writer [604]

Name			
Caden Manson	-	-	-
Cahit Sıtkı Tarancı ...	-	Interpreter ...	-
Cahit Zarifoğlu ...	-	-	-
Cai Chongda ...	-	-	Quanzhou Normal University
Cai Dongfan ...	Historical fiction	-	-
Cai Lujun ...	-	-	-
Caiseal Mór	-	Artist ...	-
Caitlin Davies ...	-	-	-
Caitlin Kittredge ...	-	-	-
Caitlin Rother	Non-fiction	Journalist	-

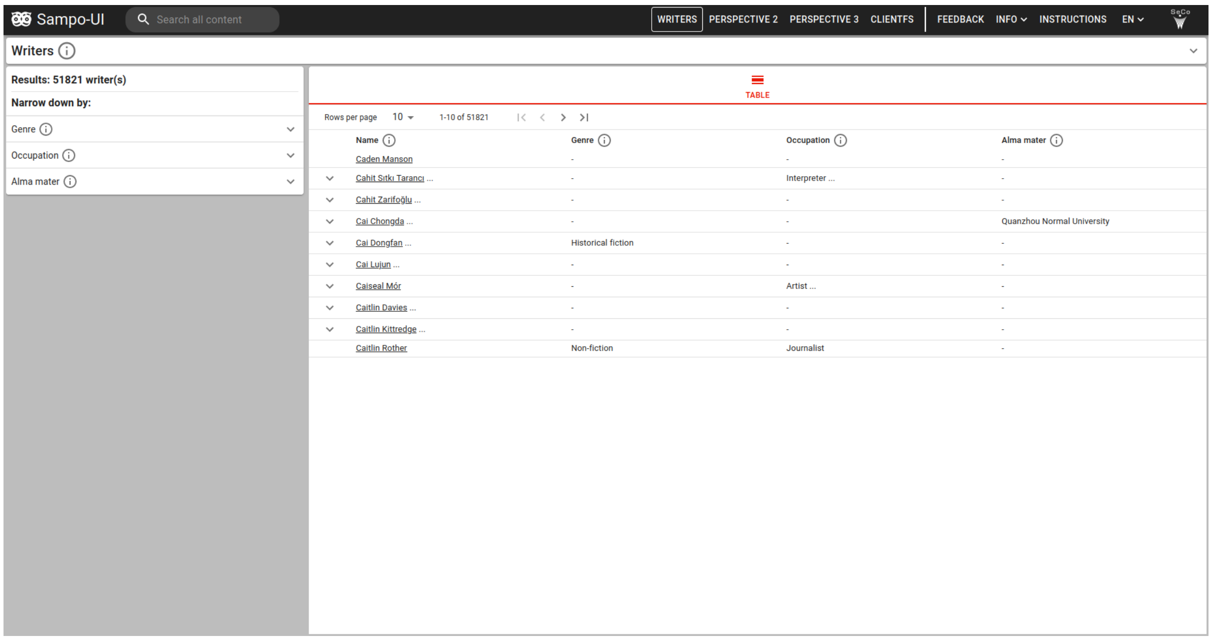
Let's open the localeEN.json file located in the directory /src/client/translations/sampo/. Find the part where the labels for 'perspective1' are defined. Let's first change some general labels for the perspective:

```
…
"perspective1": {
        "label": "Writers",
        "facetResultsType": "writer(s)",
        "shortDescription": "A perspective for browsing and searching writers",
        "longDescription": "",
        "instancePage": {
                "label": "Writer",
                "description": ""
        },
        …
```

Next you can remove the labels for the extra properties we removed and add labels for our newly created properties using the IDs we defined earlier:

```
…
"properties": {
        "uri": {
                "label": "URI",
                "description": "Uniform Resource Identifier"
        },
        "prefLabel": {
                "label": "Name",
                "description": "The name of the writer.",
                "textFacetInputPlaceholder": "Search..."
        },
        "genre": {
                "label": "Genre",
                "description": "The genre(s) the author has written."
        },
        "occupation": {
                "label": "Occupation",
                "description": "The occupation of the writer."
        },
        "almaMater": {
                "label": "Alma mater",
                "description": "The alma mater of the writer."
        }
},
…
```

The English labels now should've updated for your perspective. If you want to define labels in other languages, you just need to do the same changes to other locale files you have.



Results: 51821 writer(s)

Name	Genre	Occupation	Alma mater
Caden Manson	-	-	-
Cahit Sıtkı Tarancı ...	-	Interpreter ...	-
Cahit Zarifoğlu ...	-	-	-
Cai Chongda ...	-	-	Quanzhou Normal University
Cai Dongfan ...	Historical fiction	-	-
Cai Lujun ...	-	-	-
Caiseal Mór	-	Artist ...	-
Caitlin Davies ...	-	-	-
Caitlin Kittredge ...	-	-	-
Caitlin Rother	Non-fiction	Journalist	-

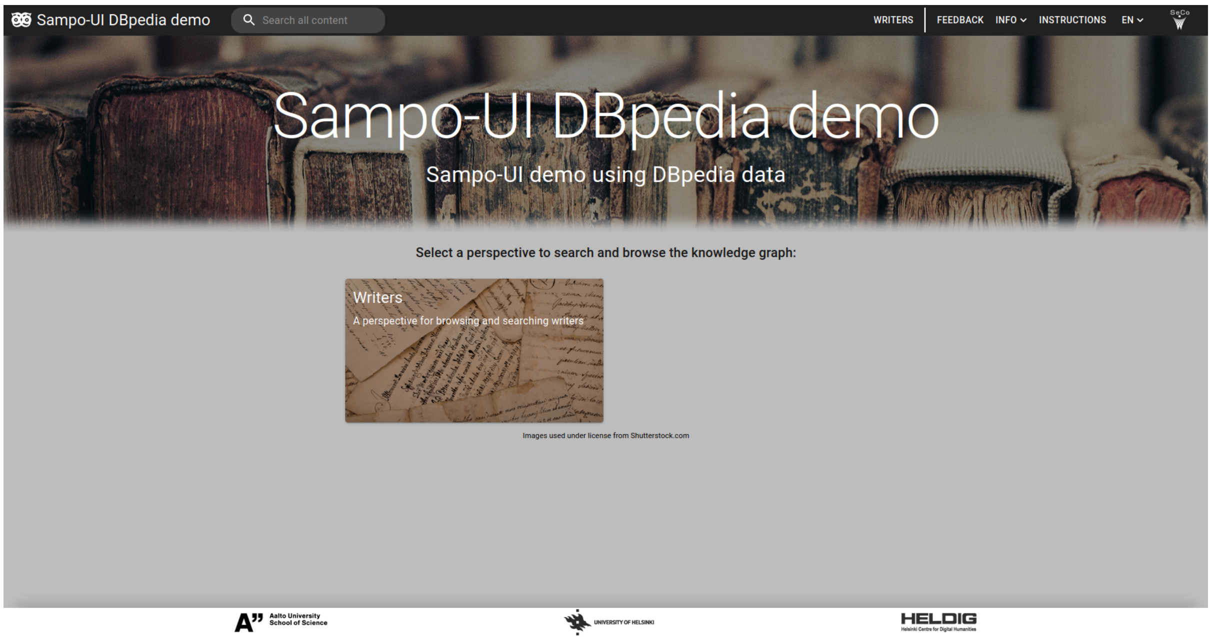
Let's just add the last finishing touches. We can edit the portal title and basic information at the start of the localization files:

```
…
"appTitle": {
            "short": "Sampo-UI DBpedia demo",
            "mobile": "S-UI Demo",
            "long": "&nbsp; &nbsp; Sampo-UI DBpedia demo &nbsp; &nbsp;",
            "subheading": "Sampo-UI demo using DBpedia data"
},
…
```

If we want to only show our 'Writers' perspective, we could hide the other perspectives by removing them from the 'perspectives' object in portalConfig.json:

```
…
"perspectives": {
            "searchPerspectives": [
                        "perspective1"
            ],
            "onlyInstancePages": [
            ]
},
…
```

Or, if you want to, you could transform the other available perspectives into new perspectives like we did with 'perspective1'! Our final portal looks like this:


Sampo-UI DBpedia demo
Search all content
WRITERS
FEEDBACK
INFO
INSTRUCTIONS
EN
Sampo-UI DBpedia demo
Sampo-UI demo using DBpedia data
Select a perspective to search and browse the knowledge graph:
Writers
A perspective for browsing and searching writers
Images used under license from Shutterstock.com

You’re now finished with the quickstart tutorial. Congratulations!

The next part will cover the creation of what we achieved in the quickstart tutorial but with more information on how configuration is done in the Sampo-UI framework.

BASIC CONFIGURATION

Configuration in Sampo-UI is mostly done through editing JSON files. The relevant directories are the following:

1. **src/client/translations/**
   This directory contains the translation files that are used for fetching all the text/labels in the portal. Each language should have its own JSON file. Two files for English (localeEN) and Finnish (localeFI) are included in the base Sampo-UI framework and can be used as a base for further localizations.
2. **src/configs/**
   This directory has the configuration files for the portal itself as well as all the perspectives included in the portal.
3. **src/server/sparql/**
   This directory has the JS files with the actual SPARQL queries used. The files for forming general SPARQL queries (e.g., how facet values are fetched based on the perspective configuration) are also included here if you need to tweak and/or troubleshoot the used general queries.

If you look inside these folders, you can see that each of them has a folder with the name ‘sampo’ that includes some of these relevant files. This folder name must match the ID of the portal, which in the case of the base Sampo-UI framework is ‘sampo’ (configured through the ‘portalID’ attribute in the portalConfig.json file in src/configs/).

optional
If you want to change the ID of your portal, you have to either rename these folders *or* duplicate them and then rename them to match your portal ID. If you decide to change the ID of your portal, there’s also a ‘sampo’ folder in src/client/components/perspectives/ that you need to rename or duplicate in the same way.

Now onto the actual configuration of your portal. Let’s first start with the configuration of the general portal.

PORTAL CONFIGURATION

Most of the configuration for the portal is done through the portalConfig.json file in src/configs/. This file is used for configuring what application perspectives are included in your portal as well as general configuration for things like available languages, the items in the top navigation menu, etc.

There are two types of application perspectives you can use in Sampo-UI:

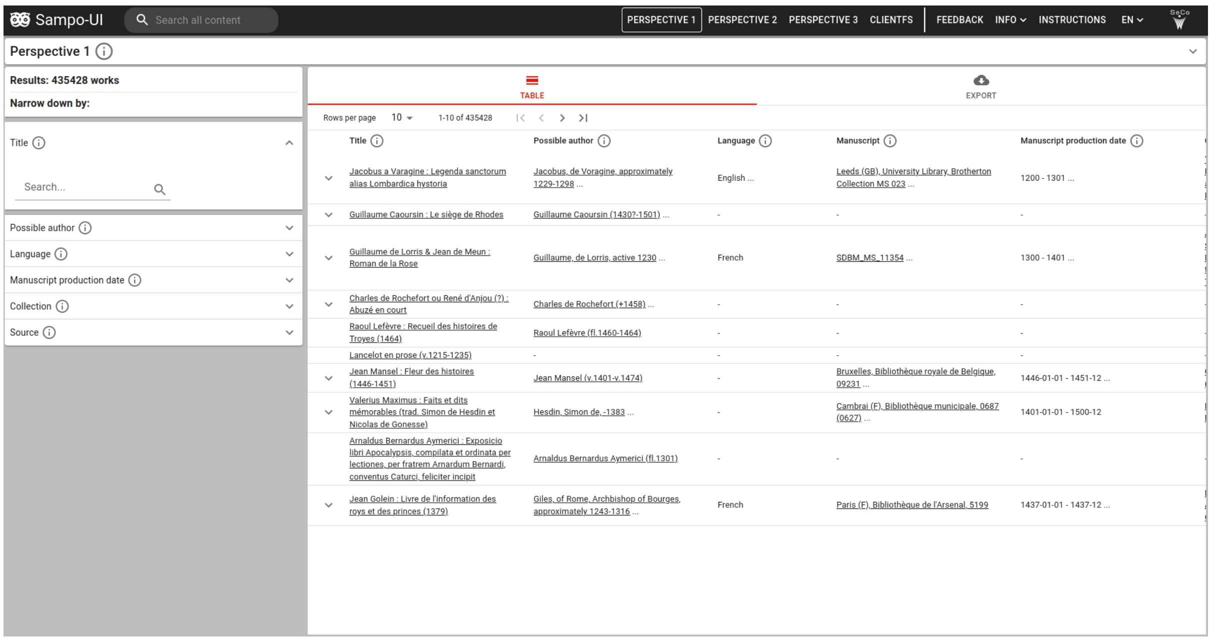
1. search perspectives

```
"searchPerspectives": [
        "perspective1",
        "perspective2",
        "perspective3",
        "perspective4",
            "fullTextSearch"
]
```

2. only instance pages perspectives

```
"onlyInstancePages": [
            "manuscripts",
            "collections",
            "works",
            "expressions",
            "events",
            "actors",
            "places"
]
```

Search perspectives are the perspectives that show up on the landing page and include a full faceted search view. Only instance pages perspectives are used when you only want to display instance pages for a certain type of entity but don't want to include a faceted search view for them.

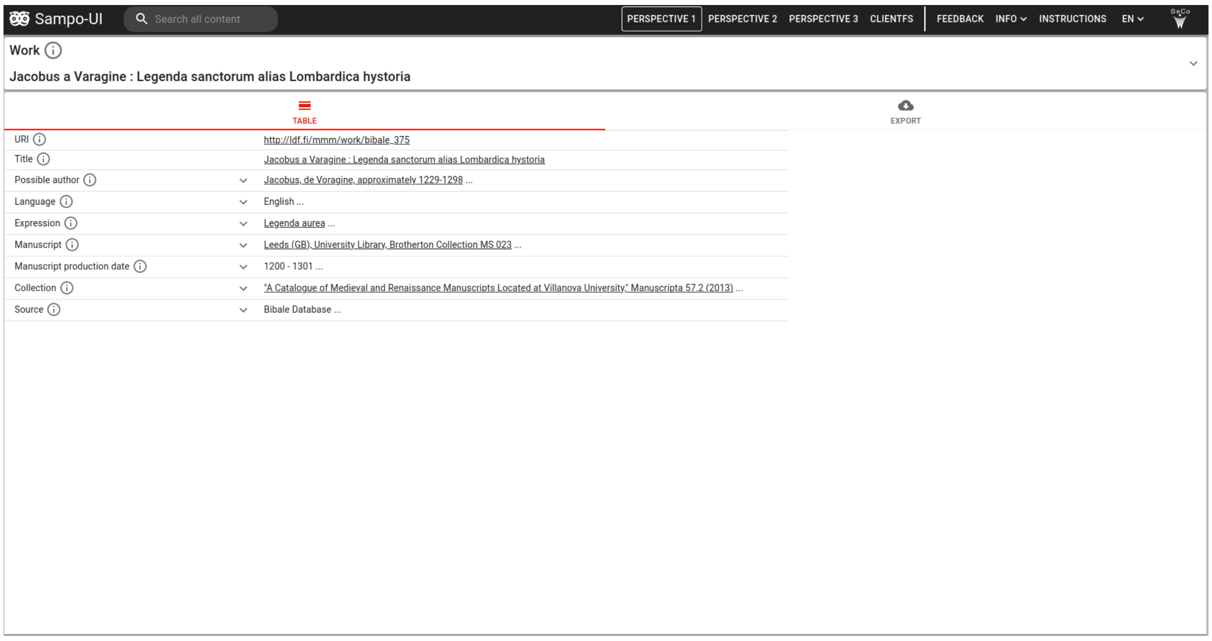


Results: 435428 works

Narrow down by:

Title	Possible author	Language	Manuscript	Manuscript production date
Jacobus a Varagine : Legenda sanctorum alias Lombardica hystoria	Jacobus, de Voragine, approximately 1229-1298 ...	English ...	Leeds (GB), University Library, Brotherton Collection MS 023 ...	1200 - 1301 ...
Guillaume Caoursin : Le siège de Rhodes	Guillaume Caoursin (1430?-1501) ...	-	-	-
Guillaume de Lorris & Jean de Meun : Roman de la Rose	Guillaume, de Lorris, active 1230 ...	French	SDBM_MS_11354 ...	1300 - 1401 ...
Charles de Rochefort ou René d'Anjou (?) : Abuzé en court	Charles de Rochefort (+1458) ...	-	-	-
Raoul Lefèvre : Recueil des histoires de Troyes (1464)	Raoul Lefèvre (fl.1460-1464)	-	-	-
Lancelot en prose (v.1215-1235)	-	-	-	-
Jean Mansel : Fleur des histoires (1446-1451)	Jean Mansel (v.1401-v.1474)	-	Bruxelles, Bibliothèque royale de Belgique, 09231 ...	1446-01-01 - 1451-12 ...
Valerius Maximus : Faits et dits mémorables (trad. Simon de Hesdin et Nicolas de Gonesse)	Hesdin, Simon de, -1383 ...	-	Cambrai (F), Bibliothèque municipale, 0687 (0627) ...	1401-01-01 - 1500-12
Arnaldus Bernardus Aymerici : Exposicio libri Apocalypsis, compilata et ordinata per lectiones, per fratrem Arnardum Bernardi, conventus Caturci, feliciter incipit	Arnaldus Bernardus Aymerici (fl.1301)	-	-	-
Jean Golein : Livre de l'information des roys et des princes (1379)	Giles, of Rome, Archbishop of Bourges, approximately 1243-1316 ...	French	Paris (F), Bibliothèque de l'Arsenal, 5199	1437-01-01 - 1437-12 ...

Faceted search view



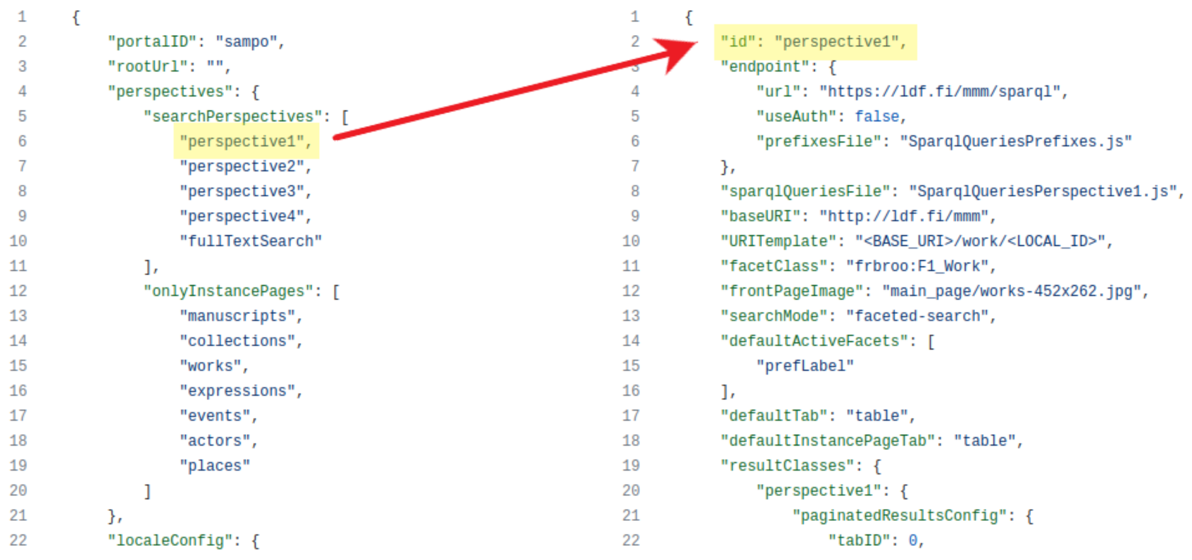
Work

Jacobus a Varagine : Legenda sanctorum alias Lombardica hystoria

URI	http://ldf.fi/mmm/work/bibale_375
Title	Jacobus a Varagine : Legenda sanctorum alias Lombardica hystoria
Possible author	Jacobus, de Voragine, approximately 1229-1298 ...
Language	English ...
Expression	Legenda aurea ...
Manuscript	Leeds (GB), University Library, Brotherton Collection MS 023 ...
Manuscript production date	1200 - 1301 ...
Collection	"A Catalogue of Medieval and Renaissance Manuscripts Located at Villanova University," Manuscripta 57.2 (2013) ...
Source	Bibale Database ...

Entity instance page

Adding new perspectives is done by creating a new perspective configuration file and adding the ID of that new perspective to either of the lists depending on what type of perspective you want it to be. This ID should match the ID you define for the perspective in its respective configuration file through the ‘id’ attribute.



```
{
    "portalID": "sampo",
    "rootUrl": "",
    "perspectives": {
        "searchPerspectives": [
            "perspective1",
            "perspective2",
            "perspective3",
            "perspective4",
            "fullTextSearch"
        ],
        "onlyInstancePages": [
            "manuscripts",
            "collections",
            "works",
            "expressions",
            "events",
            "actors",
            "places"
        ]
    },
    "localeConfig": {
```

```
{
    "id": "perspective1",
    "endpoint": {
        "url": "https://ldf.fi/mmm/sparql",
        "useAuth": false,
        "prefixesFile": "SparqlQueriesPrefixes.js"
    },
    "sparqlQueriesFile": "SparqlQueriesPerspective1.js",
    "baseURI": "http://ldf.fi/mmm",
    "URITemplate": "<BASE_URI>/work/<LOCAL_ID>",
    "facetClass": "frbroo:F1_Work",
    "frontPageImage": "main_page/works-452x262.jpg",
    "searchMode": "faceted-search",
    "defaultActiveFacets": [
        "prefLabel"
    ],
    "defaultTab": "table",
    "defaultInstancePageTab": "table",
    "resultClasses": {
        "perspective1": {
            "paginatedResultsConfig": {
                "tabID": 0,
```

Excerpt from portalConfig.json on the left and perspective1.json on the right

For this tutorial, let’s remove all the existing perspectives from both the ‘searchPerspective’ and ‘onlyInstancePagePerspectives’ and add a new perspective with the ID ‘writers’ to the list. The ‘searchPerspective’ list should now only include one string element inside it and ‘onlyInstacePagePerspectives’ should be an empty list:

```
…
"perspectives": {
            "searchPerspectives": [
                "writers"
            ],
            "onlyInstancePages": [
            ]
},
…
```

We’ll also need to create a matching perspective configuration file. You can use the perspective1.json file as a base or copy the following perspective configuration file base as a starting point:

```
{
        "id": "writers",
        "endpoint": {
                "url": "https://ldf.fi/mmm/sparql",
                "useAuth": false,
                "prefixesFile": "SparqlQueriesPrefixes.js"
        },
        "sparqlQueriesFile": "SparqlQueriesPerspective1.js",
        "baseURI": "http://ldf.fi/mmm",
        "URITemplate": "<BASE_URI>/work/<LOCAL_ID>",
        "facetClass": "frbroo:F1_Work",
        "frontPageImage": "main_page/works-452x262.jpg",
        "searchMode": "faceted-search",
        "defaultActiveFacets": [
                "prefLabel"
        ],
        "defaultTab": "table",
        "defaultInstancePageTab": "table",
        "resultClasses": {
                "writers": {
                        "paginatedResultsConfig": {
                                "tabID": 0,
                                "component": "ResultTable",
                                "tabPath": "table",
                                "tabIcon": "CalendarViewDay",
                                "propertiesQueryBlock": "workProperties",
                                "pagesize": 10,
                                "sortBy": null,
                                "sortDirection": null
                        },
                        "instanceConfig": {
                                "propertiesQueryBlock": "workProperties",
                                "instancePageResultClasses": {
                                        "instancePageTable": {
                                                "tabID": 0,
                                                "component": "InstancePageTable",
                                                "tabPath": "table",
                                                "tabIcon": "CalendarViewDay"
                                        }
                                        }
                              }
                    }
          },
          "properties": [
                    {
                              "id": "uri",
                              "valueType": "object",
                              "makeLink": true,
                              "externalLink": true,
                              "sortValues": true,
                              "numberedList": false,
                              "onlyOnInstancePage": true
                    },
                    {
                              "id": "prefLabel",
                              "valueType": "object",
                              "makeLink": true,
                              "externalLink": false,
                              "sortValues": true,
                              "numberedList": false,
                              "minWidth": 250
                    }
          ],
          "facets": {
                    "prefLabel": {
                              "sortByPredicate": "skos:prefLabel"
                    }
          }
}
```

Change the name of the file to writers.json. If you decide to use the perspective1.json file as a base, you also need to change all the ‘perspective1’ mentions inside the file to ‘writers’. Leave the query file reference (‘sparqlQueriesFile’) to refer to ‘SparqlQueriesPerspective1.js’ for the time being; we’ll handle writing new queries in the perspective configuration section. Now the landing page should have been updated to only include our newly added perspective.

Locales are configured in the ‘localeConfig’ object. You define the default locale through the ‘defaultLocale’ attribute inside that object. The available locales are defined in the ‘availableLocales’ list of objects. The default locale ID should match the ID for the locale you define in this list. Adding new locales works similarly to adding new perspectives: You add a new object for each new language and give the name of the translation file located in src/client/translations/sampo/ through the ‘filename’ attribute. For this tutorial case the existing locales are enough.

```
…
"localeConfig": {
        "defaultLocale": "en",
        "readTranslationsFromGoogleSheets": false,
        "availableLocales": [
                {
                        "id": "en",
                        "label": "English",
                        "filename": "localeEN.json"
                },
                {
                        "id": "fi",
                        "label": "Finnish",
                        "filename": "localeFI.json"
                }
        ]
},
…
```

optional
If you want to offer the users the possibility to view the queries they have made in the portal in YASGUI, you also need to configure the SPARQL data endpoint used for those queries in the portal configuration file. This is done through the 'endpoint' attribute in 'yasguiParams' in the 'yasguiConfig' object. The endpoints used in the perspectives themselves are defined in the perspective configuration files.

```
…
"yasguiConfig": {
        "yasguiBaseURL": "https://yasgui.triply.cc",
        "yasguiParams": {
                "contentTypeConstruct": "text/turtle",
                "contentTypeSelect": "application/sparql-results+json",
                "endpoint": "https://dbpedia.org/sparql",
                "requestMethod": "POST",
                "tabTitle": "Exported query"
        }
},
…
```

PERSPECTIVE CONFIGURATION

General configuration for the perspective

Perspective configuration is done through perspective specific configuration files. The first thing you define in a perspective file is the ID ('id' attribute) you use to refer to it in the portal configuration file. You also need to change the name of the object inside 'resultClasses' object to match this ID. Let's now create a new perspective configuration file for our writers perspective.

The SPARQL endpoint used in the perspective is configured in the 'endpoint' object. This object should define the URL for the endpoint through the 'url' attribute and whether the endpoint requires an authorization token to be sent with the query ('useAuth' attribute). In this case we use the DBpedia endpoint https://dbpedia.org/sparql, which doesn't require an authentication token, so we set 'useAuth' to false:

```
…
"endpoint": {
            "url": "https://dbpedia.org/sparql",
            "useAuth": false,
            "prefixesFile": "SparqlQueriesPrefixes.js"
},
…
```

If you specify the need for the authorization token, you need to include the token in a .env file with the variable name SPARQL_ENDPOINT_BASIC_AUTH.

The prefixes are added to the queries from a separate file and this file can be defined with the 'prefixesFile' attribute if you want to change it from the default 'SparqlQueriesPrefixes.js' file located in src/server/sparql/sampo/sparql_queries/.

For our DBpedia data, let's make our lives easier by at least defining the 'dbr' prefix to refer to http://dbpedia.org/resource/ and 'dbo' to http://dbpedia.org/ontology/:

```
export const prefixes = `
  PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
  …
  PREFIX semparls: <http://ldf.fi/schema/semparl/>
  PREFIX dbr: <http://dbpedia.org/resource/>
  PREFIX dbo: <http://dbpedia.org/ontology/>
`
```

You also need to specify the file containing the queries used for the table and other possible visualizations. This is done with the 'sparqlQueriesFile' attribute. The referred file should be located in src/server/sparql/sampo/sparql_queries/.

The Sampo-UI framework can form local IDs to be used for instance page URLs based on a URI template, e.g., the local part of https://example.org/12345 would be 12345 and the instance page link would end with /page/1245. You should define the base of the URI (common part for all the URIs in this perspective, e.g., 'https://example.org/) in the attribute 'baseURI'. Then in the 'URITemplate' you should define how the base URI (<BASE_URI>) and local ID (<LOCAL_ID>) are combined to form the used URIs, e.g., for our example URI https://example.org/12345 the template would be '<BASE_URI><LOCAL_ID>'.

For our perspective the base URI will be http://dbpedia.org/resource/ with the local ID following that, e.g. http://dbpedia.org/resource/Oscar_Wilde:

```
…
"sparqlQueriesFile": "SparqlQueriesPerspective1.js",
"baseURI": "http://dbpedia.org/resource/",
"URITemplate": "<BASE_URI><LOCAL_ID>",
"facetClass": "frbroo:F1_Work",
…
```

The ‘facetClass’ attribute should be used to define the type of the entities being searched in the perspective. Its value should be the value you would use as the value for rdf:type in a SPARQL query. Prefixes can be used as long as they are included in the previously defined prefix file like we did with the prefix ‘dbo’:

```
…
"sparqlQueriesFile": "SparqlQueriesPerspective1.js",
"baseURI": "http://dbpedia.org/resource/",
"URITemplate": "<BASE_URI><LOCAL_ID>",
"facetClass": "dbo:Writer",
…
```

Configuration of the content included in the perspective

View and tabs

The different views inside the perspective are configured inside the ‘resultClasses’ object. This object should include one object that shares its name with the ID of the perspective. This object shall be referred to as the ‘writers’ object in this section from this point on.

```
…
"defaultInstancePageTab": "table",
"resultClasses": {
        "writers": {
                "paginatedResultsConfig": {
                        "tabID": 0,
                        "component": "ResultTable",
                        "tabPath": "table",
                        "tabIcon": "CalendarViewDay",
                        "propertiesQueryBlock": "workProperties",
                        "pagesize": 10,
                        "sortBy": null,
                        "sortDirection": null
                },
                "instanceConfig": {
                        "propertiesQueryBlock": "workProperties",
                        "instancePageResultClasses": {
```

```
                                        "instancePageTable": {
                                                  "tabID": 0,
                                                  "component": "InstancePageTable",
                                                  "tabPath": "table",
                                                  "tabIcon": "CalendarViewDay"
                                        }
                              }
                    }
          }
},
"properties": [
          …
```

The ‘writers’ object should include at least two objects inside it: (1) a ‘paginatedResultsConfig’ object and (2) a ‘instanceConfig’ object. If you’re using the perspective1.json file as a base, you can remove the other tab configurations (not present in the above minimal code) for now.

The ‘paginatedResultsConfig’ object is used for configuring the default result table of the faceted search view. The most important attribute to pay attention to here is the ‘propertiesQueryBlock’ attribute.

```
…
"paginatedResultsConfig": {
          "tabID": 0,
          "component": "ResultTable",
          "tabPath": "table",
          "tabIcon": "CalendarViewDay",
          "propertiesQueryBlock": "workProperties",
          "pagesize": 10,
          "sortBy": null,
          "sortDirection": null
},
…
```

This attribute refers to the variable holding the relevant SPARQL query for the properties you want to show in the table in the SPARQL queries file:

```
const perspectiveID = 'perspective1'

export const workProperties = `
          {
                    ?id skos:prefLabel ?prefLabel__id .
                    BIND(?prefLabel__id AS ?prefLabel__prefLabel)
                    …
```

We can leave this attribute alone for now. We’ll return to it when we start working on our own queries later on in this tutorial.

If you want to display more items on the page (ten rows shown by default), you can adjust the value by changing the ‘pagesize’ attribute number. The other attributes you probably won’t be needing to adjust for the paginated results configuration, but will be relevant for any future tabs you might add:

The ‘tabID’ attribute determines the order the tabs are shown in the faceted search view. So by default the table view is going to be the first one with the ID 0 and any additional tabs come after it (tab ID 1, tab ID 2, …). The ‘tabIcon’ attribute just determines what icon is used for the tab and is purely a visual change. The ‘tabPath’ attribute determines the end of the URL for that particular tab. This path is also used for defining the labels to be used for the paths in the localization files.

The ‘component’ attribute determines what kind of component (e.g., table, chart, map) is used. The need for additional configuration options is also based on this, so you should preferably look at some of the perspective files included in Sampo-UI to check all the needed configuration for other component types if you decide to add them.

All the various tabs you want to include in the perspective are added to this ‘writers’ object after the previous configuration objects. Adding new tabs and more on possible components you can use are covered later in this tutorial.

The ‘instanceConfig’ object on the other hand is for configuring the instance pages of this perspective. All the different tabs included in the instance pages are defined in this object.

```
…
"instanceConfig": {
        "propertiesQueryBlock": "workProperties",
        "instancePageResultClasses": {
                "instancePageTable": {
                        "tabID": 0,
                        "component": "InstancePageTable",
                        "tabPath": "table",
                        "tabIcon": "CalendarViewDay"
                }
        }
}

…
```

Similarly to the paginated results configuration object, you need to define the variable holding the query for getting information about that particular object. In most cases this will be the same query as the one you defined for the paginated results.

If you want to define additional tabs, you add them to the ‘instancePageResultClasses’ object after the main ‘instancePageTable’ object. These new tabs work similarly to the perspective tabs: you need to define their order with IDs, the component used and configuration for path and the icon shown.

Properties

The information that you want to be shown about entities should be listed in the ‘properties’ object. Each of these properties is shown as a column in the table view unless explicitly hidden from pages other than instance pages (‘onlyOnInstancePage’ attribute set to true). The order of the columns is determined by the order you place them in in the ‘properties’ object.

```
…
"properties": [
          {
                    "id": "uri",
                    "valueType": "object",
                    "makeLink": true,
                    "externalLink": true,
                    "sortValues": true,
                    "numberedList": false,
                    "onlyOnInstancePage": true
          },
          …
```

Properties are identified with their ‘id’ property. This is the variable name you should use for the values of this property in the SPARQL queries to capture the relevant information for all entities. This ID is also used for getting the label for this column from the locale files:

```
…
"perspective1": {
          "label": "Perspective 1",
          …
          "properties": {
                    "uri": {
                              "label": "URI",
                              "description": "Uniform Resource Identifier"
                    },
                    …
```

For each property you need to define their type. This type is determined by what kind of data you have for the property. If the property just gets literal values, you should define the type to be ‘string’ so the value is just shown as string. In the SPARQL queries in this case you just need to capture the property value to the variable with the same name as the property’s ID in the configuration file.

On the other hand, if your property gets entities as its values, you should use the ‘object’ type. When you use the ‘object’ type, Sampo-UI expects your query to return data on this entity in an object format that includes at least an ‘id’ and ‘prefLabel’ attributes. The formulation of this for the SPARQL queries will be covered later on in this tutorial when we get to the SPARQL query writing part.

For property values that refer to other entities, you might also want to provide a link to that entity’s own instance page. This is done by also defining a ‘dataProviderUrl’ attribute in addition to the ‘id’ and ‘prefLabel’ attributes. In the perspective configuration you then set the ‘makeLink’ attribute for that property to true and the property values should now be shown as links. If the provided link is an external link, you need to set ‘externalLink’ to true as well.

If you want to include a column with images (e.g. portraits of people), you’ll need to define the property type to be ‘image’. For images you should capture the ‘id’ for the image as well as the URL for it (‘url’ attribute) and a description to be shown under the image when it is clicked (‘description’ attribute).

Now we can add some new properties to our Writers perspective. If you’re using the perspective1.json as a base, you can delete all the additional properties from the list apart from the initial ‘uri’ and ‘prefLabel’ properties. Let’s add three more properties about the writers: the genre(s) they wrote, their occupation(s) and their alma mater. In DBpedia these properties are denoted with the predicates dbo:genre, dbo:occupation and dbo:almaMater respectively:

```
…
"properties": [
        …
        {
                "id": "genre",
                "valueType": "object",
                "makeLink": false,
                "externalLink": false,
                "sortValues": true,
                "numberedList": false,
                "minWidth": 250
        },
        {
                "id": "occupation",
                "valueType": "object",
                "makeLink": true,
                "externalLink": false,
                "sortValues": true,
                "numberedList": false,
                "minWidth": 250
        },
        {
                "id": "almaMater",
                "valueType": "object",
                "makeLink": true,
                "externalLink": false,
                "sortValues": true,
                "numberedList": false,
                "minWidth": 250
        }
]
```

You should now see four columns in the table view, but all of them are missing labels. The table doesn't have any values for these columns yet as we haven't written the queries for these properties.

Let's now create our SPARQL query file with the name SparqlQueriesWriters.js. You can either use SparqlQueriesPerspective1.js as a base or just use the following base code:

```
const perspectiveID = 'perspective1'

export const workProperties = `
        {
                ?id skos:prefLabel ?prefLabel__id .
                BIND(?prefLabel__id AS ?prefLabel__prefLabel)
                BIND(CONCAT("/${perspectiveID}/page/", REPLACE(STR(?id), "^.*\\\\/(.+)", "$1"))
AS ?prefLabel__dataProviderUrl)
                BIND(?id as ?uri__id)
                BIND(?id as ?uri__dataProviderUrl)
                BIND(?id as ?uri__prefLabel)
        }
`
```

First we'll need to configure the perspective ID to match the ID defined in our perspective file:

```
const perspectiveID = 'writers'

export const workProperties = `
        …
```

Let's also adjust the name of the query variable to better match what it's querying:

```
…
export const writerProperties = `
        {
                ?id skos:prefLabel ?prefLabel__id .
                …
```

Now we can adjust the mentions in writers.json to refer to the correct file and queries:

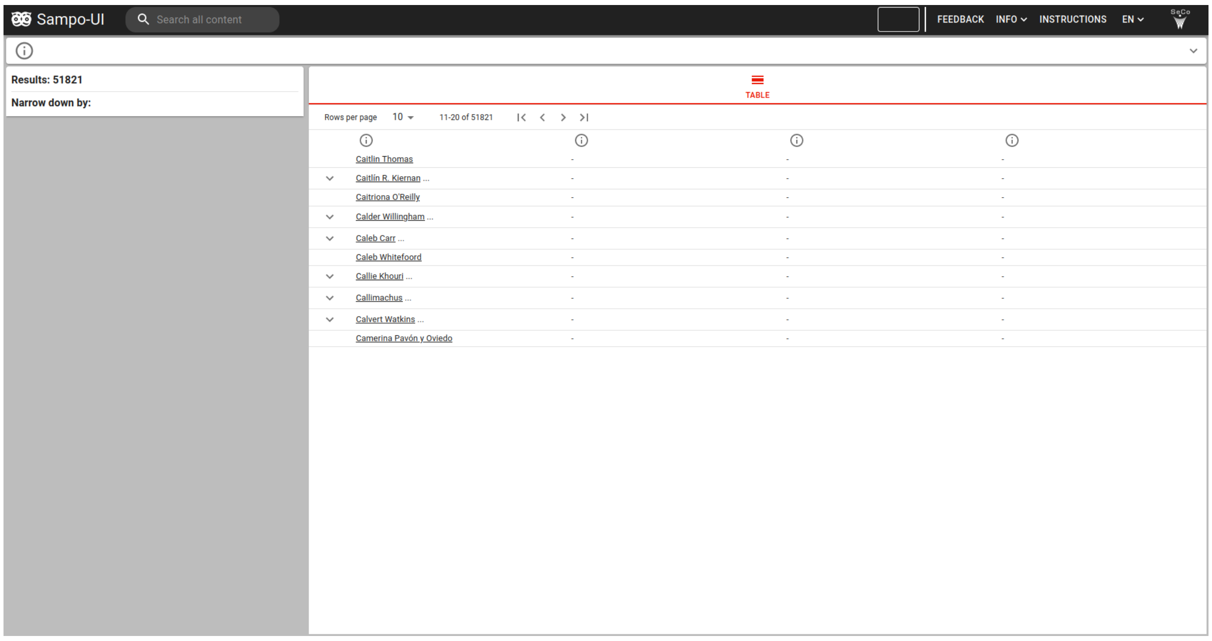
```
"sparqlQueriesFile": "SparqlQueriesWriters.js",
…
"resultClasses": {
        "writers": {
                "paginatedResultsConfig": {
                        …
                        "propertiesQueryBlock": "writerProperties",
                        …
                },
                "instanceConfig": {
                        "propertiesQueryBlock": "writerProperties",
                        …
```

Now to get at least the names of the writers showing up in our table, we need to edit the predicate used for getting the labels from skos:prefLabel to the predicate used in DBpedia data, rdfs:label:

```
const perspectiveID = 'writers''

export const writerProperties = `
        {
                ?id rdfs:label ?prefLabel__id .
                BIND(?prefLabel__id AS ?prefLabel__prefLabel)
                BIND(CONCAT("/${perspectiveID}/page/", REPLACE(STR(?id), "^.*\\\\/(.+)", "$1"))
AS ?prefLabel__dataProviderUrl)
                BIND(?id as ?uri__id)
                BIND(?id as ?uri__dataProviderUrl)
                BIND(?id as ?uri__prefLabel)
        }
`
```

Now if we update the page, we should be able to see the names of the writers in the table. These names should also already be underlined and lead to that particular entity's instance page, as the code above also forms their data provider URLs.



Results: 51821
Narrow down by:

Rows per page 10 11-20 of 51821

Caitlin Thomas	-	-	-
Caitlín R. Kiernan ...	-	-	-
Caitriona O'Reilly	-	-	-
Calder Willingham ...	-	-	-
Caleb Carr ...	-	-	-
Caleb Whitefoord	-	-	-
Callie Khouri ...	-	-	-
Callimachus ...	-	-	-
Calvert Watkins ...	-	-	-
Camerina Pavón y Oviedo	-	-	-

We can add the queries for our other properties with unions in the query. Since all the properties get objects as the values, we should define both IDs and preferred labels for all the property values:

(1) genre__id, genre__prefLabel
(2) occupation__id, occupation__prefLabel
(3) almaMater__id, almaMater__prefLabel

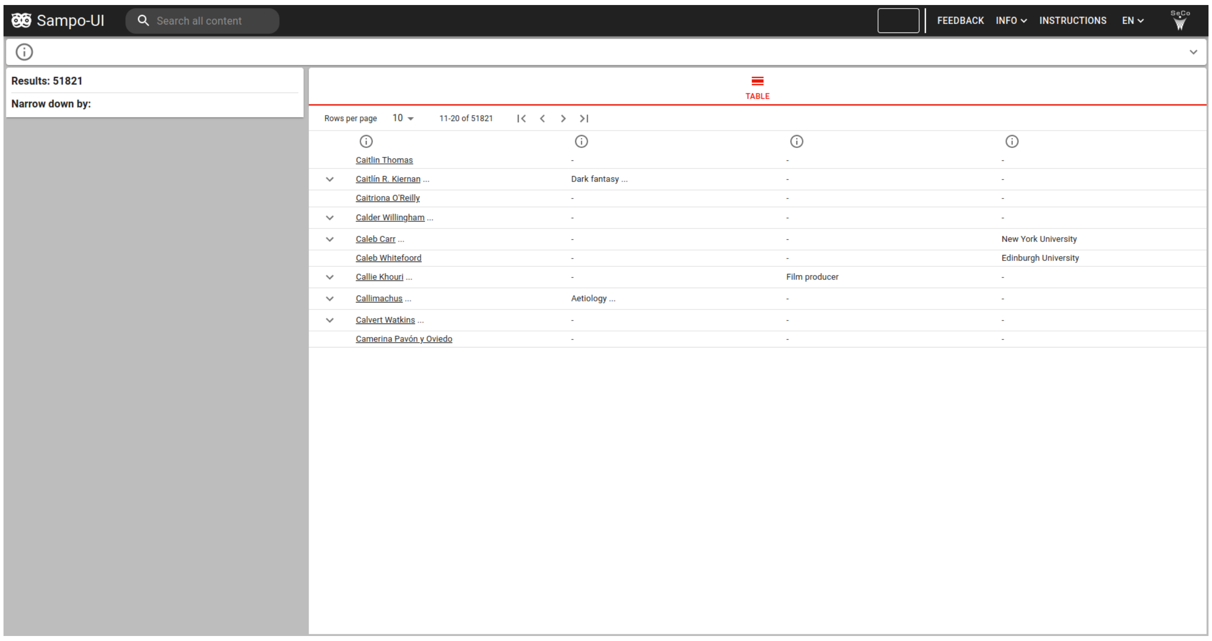
Note that there are **two underscores** instead of just one between the name of the property and the attribute.

```
…
export const writerProperties = `
	{
		?id rdfs:label ?prefLabel__id .
		BIND(?prefLabel__id AS ?prefLabel__prefLabel)
		BIND(CONCAT("/${perspectiveID}/page/", REPLACE(STR(?id), "^.*\\\\/(.+)", "$1"))
AS ?prefLabel__dataProviderUrl)
		BIND(?id as ?uri__id)
		BIND(?id as ?uri__dataProviderUrl)
		BIND(?id as ?uri__prefLabel)
	}
	UNION
	{
		?id dbo:genre ?genre__id .
		?genre__id rdfs:label ?genre__prefLabel .
	}
	UNION
	{
		?id dbo:occupation ?occupation__id .
		?occupation__id rdfs:label ?occupation__prefLabel .
	}
	UNION
	{
		?id dbo:almaMater ?almaMater__id .
		?almaMater__id rdfs:label ?almaMater__prefLabel .
	}
`
```

If you update the page now, you should see some values filled in for the last three columns. You might however see some values in languages other than English. For the purpose of this tutorial we'll only want to include the English labels. This problem is fixed by adding filters for the language of the preferred labels:

```
…
UNION
{
	?id dbo:genre ?genre__id .
	?genre__id rdfs:label ?genre__prefLabel .
	FILTER(LANG(?genre__prefLabel) = 'en')
}
UNION
{
	?id dbo:occupation ?occupation__id .
```

```
        ?occupation__id rdfs:label ?occupation__prefLabel .
        FILTER(LANG(?occupation__prefLabel) = 'en')
}
UNION
{
        ?id dbo:almaMater ?almaMater__id .
        ?almaMater__id rdfs:label ?almaMater__prefLabel .
        FILTER(LANG(?almaMater__prefLabel) = 'en')
}
…
```



Results: 51821

Narrow down by:

Rows per page 10 11-20 of 51821

Caitlin Thomas	-	-	-
Caitlín R. Kiernan ...	Dark fantasy ...	-	-
Caitriona O'Reilly	-	-	-
Calder Willingham ...	-	-	-
Caleb Carr ...	-	-	New York University
Caleb Whitefoord	-	-	Edinburgh University
Callie Khouri ...	-	Film producer	-
Callimachus ...	Aetiology ...	-	-
Calvert Watkins ...	-	-	-
Camerina Pavón y Oviedo	-	-	-

After updating the page, there should only be English labels present in the table. Now we can just add the labels for all the columns and then move onto facets.

We'll need to add a new object for our Writers perspective into the localization files inside the 'perspectives' object. You could for example place this after the 'perspective4' object in the file. You can use the following object as your base for the labels:

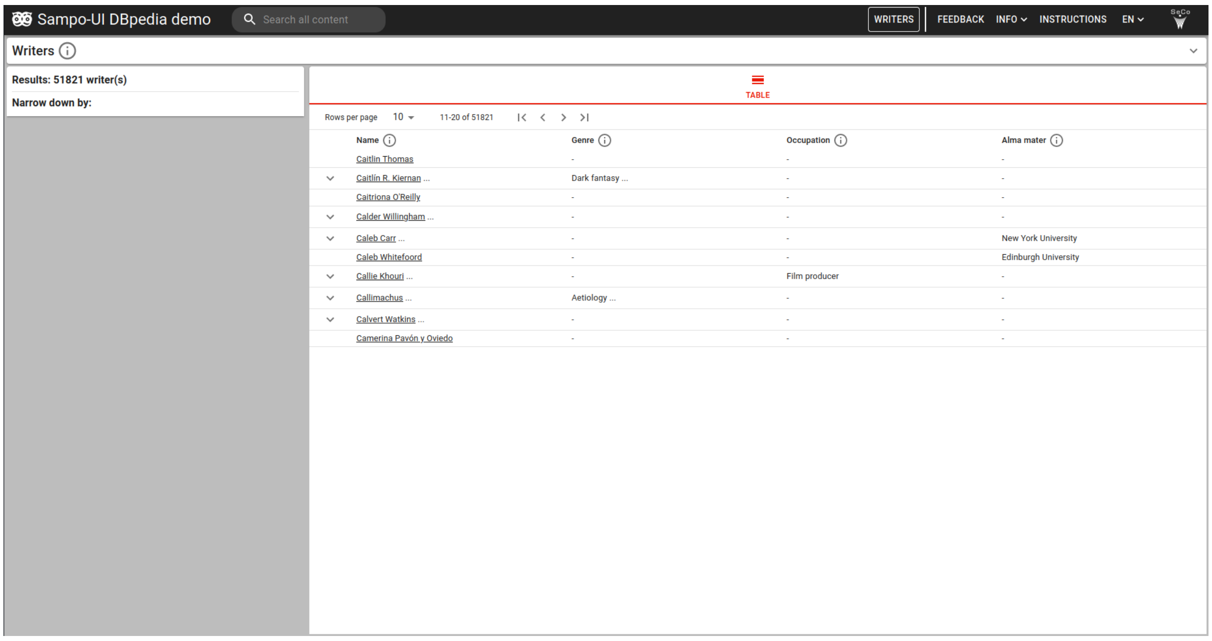
```
…
"writers": {
        "label": "Writers",
        "facetResultsType": "writer(s)",
        "shortDescription": "A perspective for browsing and searching writers",
        "longDescription": "",
        "instancePage": {
                "label": "Writer",
                "description": ""
        },
        "properties": {
                "uri": {
```

```
            "label": "URI",
            "description": "Uniform Resource Identifier"
        },
        "prefLabel": {
            "label": "Name",
            "description": "The name of the writer."
        }
    }
}
…
```

This base code should already add some text into the view. The preferred label column should now have the label ‘Name’ and we see that the perspective card on the landing page now has text as well.

Let’s now define the labels for genre, occupation and alma mater similarly to the already defined properties by creating new attributes holding objects into the ‘properties’ object with the property ID as their attribute name:

```
…
"properties": {
    "uri": {
        "label": "URI",
        "description": "Uniform Resource Identifier"
    },
    "prefLabel": {
        "label": "Name",
        "description": "The name of the writer."
    },
    "genre": {
        "label": "Genre",
        "description": "The genre(s) the author has written."
    },
    "occupation": {
        "label": "Occupation",
        "description": "The occupation of the writer."
    },
    "almaMater": {
        "label": "Alma mater",
        "description": "The alma mater of the writer."
    }
}
…
```



Results: 51821 writer(s)

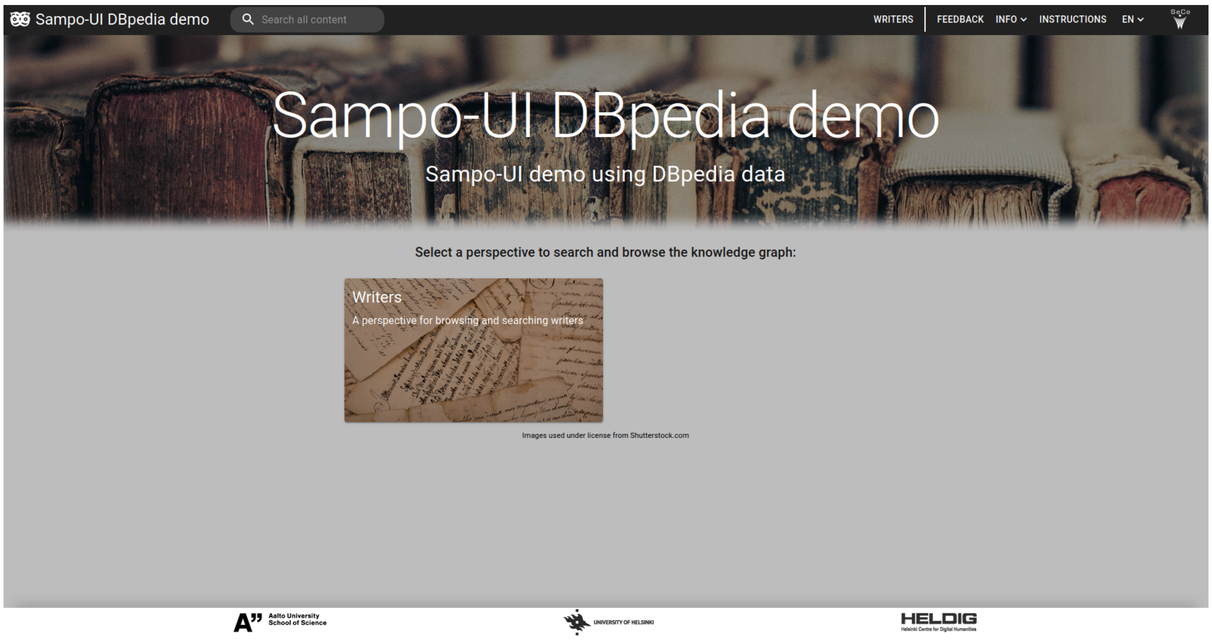
Narrow down by:

Rows per page 10 11-20 of 51821

Name	Genre	Occupation	Alma mater
Caitlin Thomas	-	-	-
Caitlín R. Kiernan ...	Dark fantasy ...	-	-
Caitriona O'Reilly	-	-	-
Calder Willingham ...	-	-	-
Caleb Carr ...	-	-	New York University
Caleb Whitefoord	-	-	Edinburgh University
Callie Khouri ...	-	Film producer	-
Callimachus ...	Aetiology ...	-	-
Calvert Watkins ...	-	-	-
Camerina Pavón y Oviedo	-	-	-

The rest of the columns should now also have the labels we defined. While we're editing the locale files, let's also adjust some of the general portal text at the start of the file to better match our portal:

```
…
"html": {
        **"title": "Sampo-UI DBpedia demo",**
        "description": "The Sampo-UI framework makes it possible to create highly customizable, user-friendly, and responsive user interfaces using current state-of-the-art JavaScript libraries and data from SPARQL endpoints."
},
"appTitle": {
        **"short": "Sampo-UI DBpedia demo",**
        "mobile": "S-UI",
        **"long": "&nbsp; &nbsp; Sampo-UI DBpedia demo &nbsp; &nbsp;",**
        **"subheading": "Sampo-UI demo using DBpedia data"**
},
…
```



Sampo-UI DBpedia demo

Sampo-UI demo using DBpedia data

Select a perspective to search and browse the knowledge graph:

A perspective for browsing and searching writers

Images used under license from Shutterstock.com

If you want to include other languages than English, you need to do all the same changes to the other locale files as well.

Facets

The facets you want to include in the facet menu are defined in the ‘facets’ object. Similarly to properties, the order of the facets in the menu is determined by the order they are listed in the ‘facets’ object.

```
…
"facets": {
        "prefLabel": {
                "sortByPredicate": "skos:prefLabel"
        }
}
…
```

If you’re using the perspective1.json file as the base, your ‘facets’ object will contain facets other than just the one for the preferred label, ‘prefLabel’. You can remove these for now. You should also shorten the ‘prefLabel’ facet to the form above. Sampo-UI’s text search is configured to work on Apache Lucene text search engine and may require further configuration on your part to work with your own data. By just leaving the ‘sortByPredicate’ attribute to the facet, the facet is removed from the facets menu but enables sorting by the column in the table view.

Since the name labels for writers in DBpedia are annotated using the rdfs:label-property, we also need to change the predicate to match:

```
…
```

```
"facets": {
        "prefLabel": {
                "sortByPredicate": "rdfs:label"
        }
}
…
```

In general, Sampo-UI by default tends to use skos:prefLabel for trying to get labels for things. If your data uses another predicate (e.g., rdfs:label), you might need to specifically configure the used predicate using additional configuration options introduced later in this section.

Let's now add a new facet for the genre(s) a writer has written:

```
…
"prefLabel": {
        "sortByPredicate": "skos:prefLabel"
},
"genre": {
        "containerClass": "ten",
        "facetType": "list",
        "filterType": "uriFilter",
        "predicate": "dbo:genre",
        "searchField": true,
        "sortButton": true,
        "sortBy": "instanceCount",
        "sortByPredicate": "dbo:genre/rdfs:label",
        "sortDirection": "desc"
}
…
```

The first attribute, 'containerClass', determines the height of the facet when it is expanded. For larger facets like checkbox facets, 'ten' is usually used as the values. For smaller facets like text search facets, 'four' or 'three' is usually used.

The type of facet is determined by the 'facetType' and 'filterType' attributes. The possible types of facets included in the Sampo-UI framework are 'text', 'list', 'hierarchical', 'timespan' and 'integer'. You then need to combine it with an appropriate 'filterType'. Sampo-UI includes different facet types in the four default perspectives it comes with, which you can use as a reference for their correct configuration. For our facets, we'll want to use checkbox facets so we set the type as 'list' and filter type as 'uriFilter'.

The different facets rely on partially automatically formed queries. For this reason you'll need to define the used predicate ('predicate' attribute) for getting the values as well as the label predicate ('sortByPredicate' attribute) for sorting based on labels. So for our case of genres, we'll set 'dbo:genre' as the predicate and then the predicate chain 'dbo:genre/rdfs:label' as the sorting predicate.

The default sorting behavior is configured with the ‘sortBy’ and ‘sortDirection’ attributes. The ‘sortBy’ attribute determines whether the default sorting should be done based on instance counts or labels and the ‘sortDirection’ attribute determines the direction, either ascending or descending.

You can also toggle a search field for facet values and a button for changing the sorting behavior by setting the ‘searchField’ and ‘sortButton’ attributes to either true or false respectively.

Now if we look at the facet we created with the above code, we see that the genres have their URIs listed instead of labels. Sampo-UI by default uses skos:prefLabel for getting the labels for facets, so we need to add an additional configuration option to our facet here for getting the labels with ‘rdfs:label’ instead:

```
…
"genre": {
        "containerClass": "ten",
        "facetType": "list",
        "filterType": "uriFilter",
        "facetLabelPredicate": "rdfs:label",
        "predicate": "dbo:genre",
        "searchField": true,
        "sortButton": true,
        "sortBy": "instanceCount",
        "sortByPredicate": "dbo:genre/rdfs:label",
        "sortDirection": "desc"
}
…
```

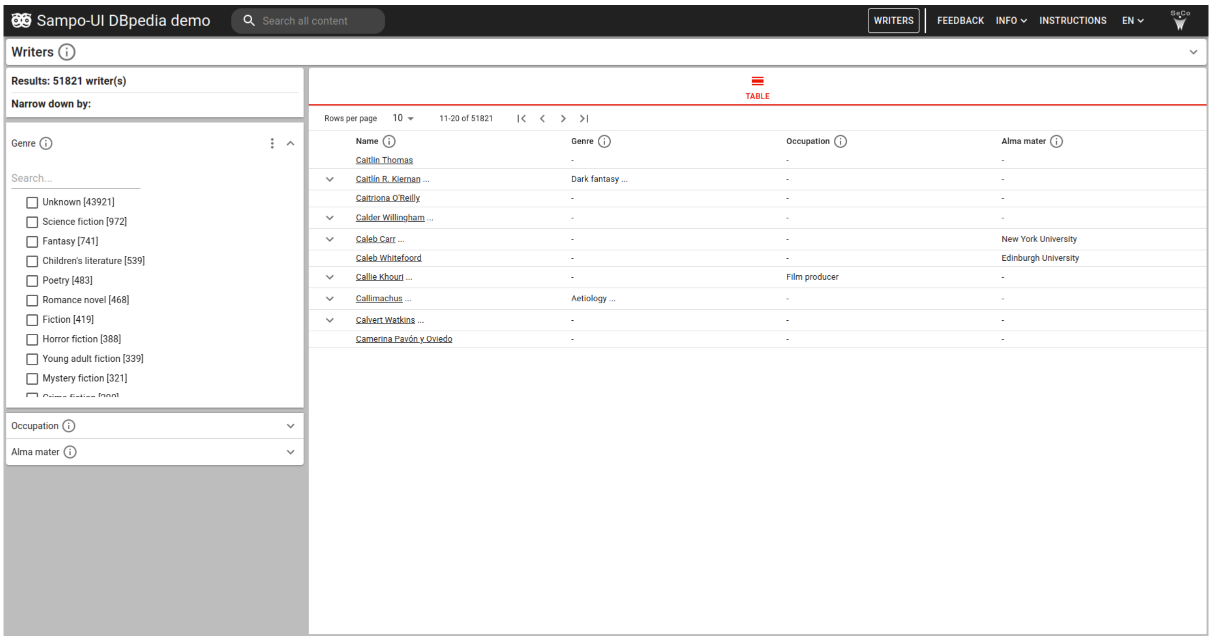
Now we have the labels in the facet, but we can see that there are multiple checkboxes for the same genres but just with labels in different languages. Since we just want one label for every genre and for that label to be the English label, we’ll need to add one more configuration options to our facet:

```
…
"genre": {
        "containerClass": "ten",
        "facetType": "list",
        "filterType": "uriFilter",
        "facetLabelPredicate": "rdfs:label",
        "facetLabelFilter": "FILTER(LANG(?prefLabel_) = 'en')",
        "predicate": "dbo:genre",
        "searchField": true,
        "sortButton": true,
        "sortBy": "instanceCount",
        "sortByPredicate": "dbo:genre/rdfs:label",
        "sortDirection": "desc"
}
…
```

With the facet for the genres now configured, we can use this configuration for defining the other two facets. You just need to change the name of the attribute holding the object as well as the predicates:

```
…
"facets": {
            …
            "genre": {
                        …
            },
            "occupation": {
                        "containerClass": "ten",
                        "facetType": "list",
                        "filterType": "uriFilter",
                        "facetLabelPredicate": "rdfs:label",
                        "facetLabelFilter": "FILTER(LANG(?prefLabel_) = 'en')",
                        "predicate": "dbo:occupation",
                        "searchField": true,
                        "sortButton": true,
                        "sortBy": "instanceCount",
                        "sortByPredicate": "dbo:occupation/rdfs:label",
                        "sortDirection": "desc"
            },
            "almaMater": {
                        "containerClass": "ten",
                        "facetType": "list",
                        "filterType": "uriFilter",
                        "facetLabelPredicate": "rdfs:label",
                        "facetLabelFilter": "FILTER(LANG(?prefLabel_) = 'en')",
                        "predicate": "dbo:almaMater",
                        "searchField": true,
                        "sortButton": true,
                        "sortBy": "instanceCount",
                        "sortByPredicate": "dbo:almaMater/rdfs:label",
                        "sortDirection": "desc"
            }
}
…
```

Since we used the same names for the attribute names in the ‘facets’ object as we did for the IDs in the ‘properties’ object, all the facets you have in your facet menu should have their labels there. If we had for some reason used names that weren’t previously used as properties, we would’ve had to add the labels in the locale files similarly to how we did with the properties.



Results: 51821 writer(s)

Narrow down by:

Genre

- ☐ Unknown [43921]
- ☐ Science fiction [972]
- ☐ Fantasy [741]
- ☐ Children's literature [539]
- ☐ Poetry [483]
- ☐ Romance novel [468]
- ☐ Fiction [419]
- ☐ Horror fiction [388]
- ☐ Young adult fiction [339]
- ☐ Mystery fiction [321]

Rows per page 10 11-20 of 51821

Name	Genre	Occupation	Alma mater
Caitlin Thomas	-	-	-
Caitlín R. Kiernan ...	Dark fantasy ...	-	-
Caitriona O'Reilly	-	-	-
Calder Willingham ...	-	-	-
Caleb Carr ...	-	-	New York University
Caleb Whitefoord	-	-	Edinburgh University
Callie Khouri ...	-	Film producer	-
Callimachus ...	Aetiology ...	-	-
Calvert Watkins ...	-	-	-
Camerina Pavón y Oviedo	-	-	-

Visualization tabs

This section will cover adding new tabs to your perspective. For the purpose of this tutorial, we'll be adding a pie/bar chart visualization for the occupations of writers to our Writers perspective.

Let's first start with the tab configuration in our writers.json file. Like previously mentioned, the configuration for new tabs is added to the 'resultClasses' object after our 'writers' object:

```
…
"resultClasses": {
        …
        "writersByProperty": {
                "tabID": 1,
                "component": "ApexCharts",
                "doNotRenderOnMount": true,
                "tabPath": "pie_chart",
                "tabIcon": "PieChart",
                "facetClass": "writers",
                "dropdownForResultClasses": true,
                "defaultResultClass": "writersByOccupation",
                "resultClasses": {
                        "writersByOccupation": {
                                "sparqlQuery": "writersByOccupationQuery",
                                "filterTarget": "writer",
                                "resultMapper": "mapPieChart",
                                "sliceVisibilityThreshold": 0.01,
                                "dropdownForChartTypes": true,
                                "chartTypes": [
                                        {
                                                "id": "pie",
                                                "createChartData": "createApexPieChartData"
                                        },
                                        {
                                                "id": "bar",
                                                "createChartData": "createApexBarChartData"
                                        }
                                ]
                        }
                }
        }
},
…
```

The pie/bar chart is built with the ApexCharts library, so the correct component name is ‘ApexCharts’. The same chart can be used to visualize the ratios of different values that can be switched between with a dropdown menu, so the configuration has a ‘resultClasses’ object that contains all the different groupings used. Here we’ll only have one result class, but still show the dropdown and use this result class configuration, since it’s easier to build onto. If you want to hide the dropdown for result classes, the configuration has to be adjusted slightly as the component will then only expect one result class.

Since we only have one result class (‘writersByOccupation’), we’ll also set that result class as the default. You can define the available chart types (pie, bar) for each result class by listing either one or both options inside the ‘chartTypes’ list.

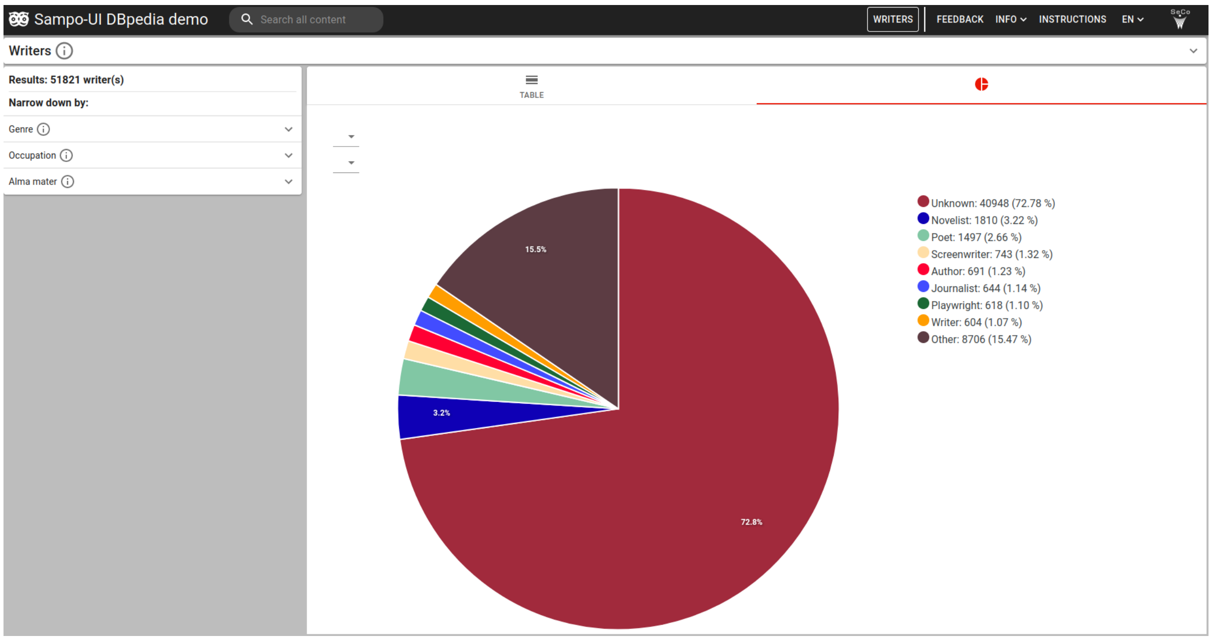
The query used for each of the result classes is defined with the ‘sparqlQuery’ attribute. This should refer to the variable in the SPARQL queries file that contains the appropriate query. We’ll be writing this in a bit for our pie/bar chart. The pie/bar chart visualizations can be affected by the facet choices we make, so we’ll also need to define what variable we’re going to be filtering in our pie/bar chart query. In this case we’ll be using the variable name ‘writer’ to refer to writer entities and setting that as the value to the ‘filterTarget’ attribute.

Some additional attributes you should pay attention to are the ‘facetClass’ and ‘sliceVisibilityThreshold’ attributes. The ‘facetClass’ attribute should match the perspective ID in this case (‘writers’). The ‘sliceVisibilityThreshold’ attribute is used for setting the lowest percentage slice that is shown in the pie chart. Any slices that make up a smaller percentage than this will be aggregated under a slice called ‘Other’. So, for example, if you want to only show slices that make up at least 1% of all the values, you should set the ‘sliceVisibilityThreshold’ attribute to 0.01.

Now we can get to writing the query itself. The query is written in the same file used for other things in the perspective, so ‘SparqlQueriesWriters.js’ in our case. The query should be stored in a variable named in the way as specified in the perspective configuration file:

```
...
export const writersByOccupationQuery = `
        SELECT ?category ?prefLabel (COUNT(DISTINCT ?writer) as ?instanceCount)
        WHERE {
                <FILTER>
                {
                        ?writer a dbo:Writer ;
                                dbo:occupation ?category .
                        ?category rdfs:label ?prefLabel .
                        FILTER(LANG(?prefLabel) = 'en')
                }
                UNION
                {
                        ?writer a dbo:Writer .
                        FILTER NOT EXISTS {
                                ?writer dbo:occupation [] .
                        }
                        BIND("Unknown" as ?category)
                        BIND("Unknown" as ?prefLabel)
                }
        }
        GROUP BY ?category ?prefLabel
        ORDER BY DESC(?instanceCount)
`
```

The pie/bar chart visualization expects your query to return three different columns: ‘category’, ‘prefLabel’ and ‘instanceCount’. The ‘category’ variable should hold the ID of the category, e.g., the ID of the occupation entities here. The ‘prefLabel’ variable should have the label that you want to be shown in the pie/bar chart for that particular slice. Lastly, the ‘instanceCount’ variable should have the count of instances to be included in that slice.



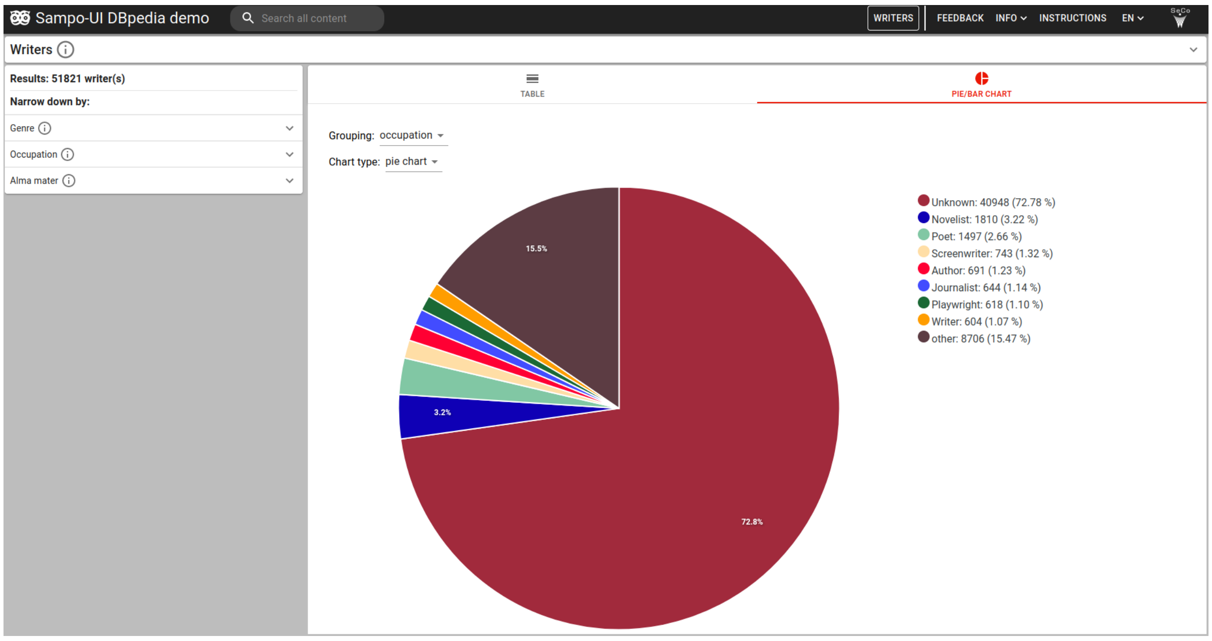
Next we want to add labels for the tab as well as the interface for the pie/bar chart. The label for the tab is added in the 'tabs' object in the locale files. You should name the attribute according to the tab path you defined for the tab:

```
…
"tabs": {
        "table": "table",
        …
        "download": "download",
        "recommendations": "recommendations",
        "pie_chart": "pie/bar chart"
},
…
```

General labels for the pie chart and its result classes are added by adding a object called 'apexCharts' in the locale file:

```
…
"table": {
        …
},
"apexCharts": {
        "grouping": "Grouping:",
        "by": "by",
        "property": "X-axis:",
        "chartType": "Chart type:",
        "pie": "pie chart",
        "bar": "bar chart",
        "other": "other",
        "resultClasses": {
                "writersByOccupation": "occupation"
        }
},
…
```

The general labels will now work for all future pie/bar charts. The only thing you'll need to add for all pie/bar charts individually is the label for the result class. For this you should always use the name of the attribute holding that result class object.



Results: 51821 writer(s)
Narrow down by:

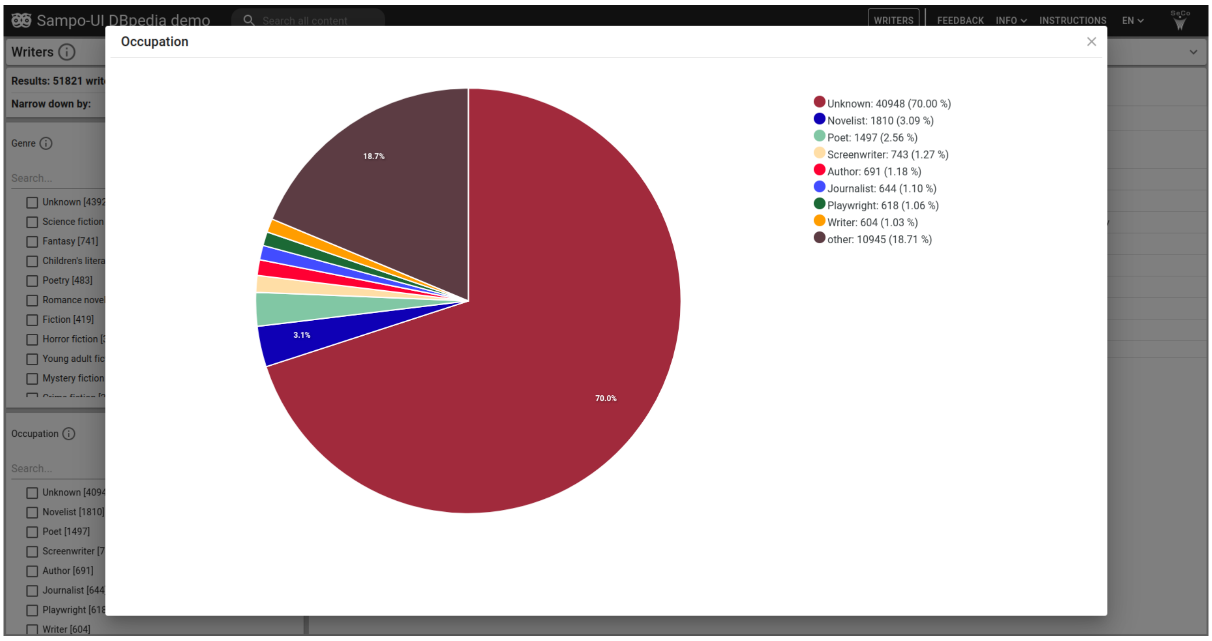
Grouping: occupation
Chart type: pie chart

● Unknown: 40948 (72.78 %)
● Novelist: 1810 (3.22 %)
● Poet: 1497 (2.66 %)
● Screenwriter: 743 (1.32 %)
● Author: 691 (1.23 %)
● Journalist: 644 (1.14 %)
● Playwright: 618 (1.10 %)
● Writer: 604 (1.07 %)
● other: 8706 (15.47 %)

We can also easily add pie charts into the facets themselves. This is done by setting the ‘pieChartButton’ attribute to true inside the facet objects for each facet:

```
…
"genre": {
        …
        "sortDirection": "desc",
        "pieChartButton": true
},
"occupation": {
        …
        "sortDirection": "desc",
        "pieChartButton": true
},
"almaMater": {
        …
        "sortDirection": "desc",
        "pieChartButton": true
}
…
```

Now if you open any of the facets, there should be a pie chart button in the top right corner of that facet. Clicking on that button opens up a pie chart with values from that particular facet:



These pie charts don't offer the same configuration options like slice visibility thresholds that a proper pie chart tab would offer, but give the user an easy way to quickly check the ratio of values inside a facet.

Sampo-UI also offers other types of visualizations. This tutorial doesn't cover their configuration in detail, but you can look at how they are configured in some example cases:

1. **Leaflet map**
   You can see this visualization in action in Sampo-UI's Perspective 2 'Production places' tab. The configuration is here and the SPARQL query here, here and here.
2. **Heatmap**
   You can see this visualization in action in Sampo-UI's Perspective 2 'Production heatmap' tab. The configuration is here and the SPARQL query here.
3. **Line chart**
   You can see this visualization in action in Sampo-UI's Perspective 2 'Production dates' and 'Event dates' tabs. The configuration for the 'Production dates' tab is here and the SPARQL query here. The configuration for the Event dates' tab is here and the SPARQL query here.
4. **Migrations**
   You can see this visualization in action in Sampo-UI's Perspective 2 'Migrations' tab. The configuration is here and the SPARQL query here.
5. **Network**
   You can see this visualization in action in Sampo-UI's Perspective 2 'Network' tab. The configuration is here and the SPARQL query here.