Data Stewardship Wizard

Release 4.5

DSW Team

Apr 05, 2024

ABOUT

The Data Stewardship Wizard is a tool that helps researchers and data stewards create data management plans (DMPs) easily, efficiently, and in a FAIR manner.

Data stewards can easily capture the knowledge, including required project data and decisions in knowledge models that are then turned into per-project questionnaires to be filled by researchers. The questionnaires guide researchers through the process using recommendations, FAIR metrics indications, and only showing relevant questions based on previous answers.

Once the questionnaire is completed, a DMP can be easily generated using a selected template and output format. The document is then stored in DSW for easy access and future reference. This is especially helpful because many funding agencies now require a DMP for their application process.

But the benefits of using DSW go beyond just creating a DMP. Researchers also learn how to handle data correctly, make it FAIR, maintain it throughout the project, and curate it long-term. This intelligent, guided, and efficient approach to composing DMPs is useful for ELIXIR nodes, research institutions, and individual researchers alike.

https://youtu.be/gcSPG_dyVUQ

STRUCTURE OF THE GUIDE

The guide sections are organized into three categories:

- **About** contains an introduction to the Data Stewardship Wizard and its content to gain quick insight into how it works at a high level.

- **Application** is structured the same way as DSW's main menu to quickly find the relevant sections about how to use a specific part of the application.

- **More** contains all additional information related to DSW, such as development roadmap, how to develop own content, or how to run own instance.

Here are some recommended sections where to start based on the role:

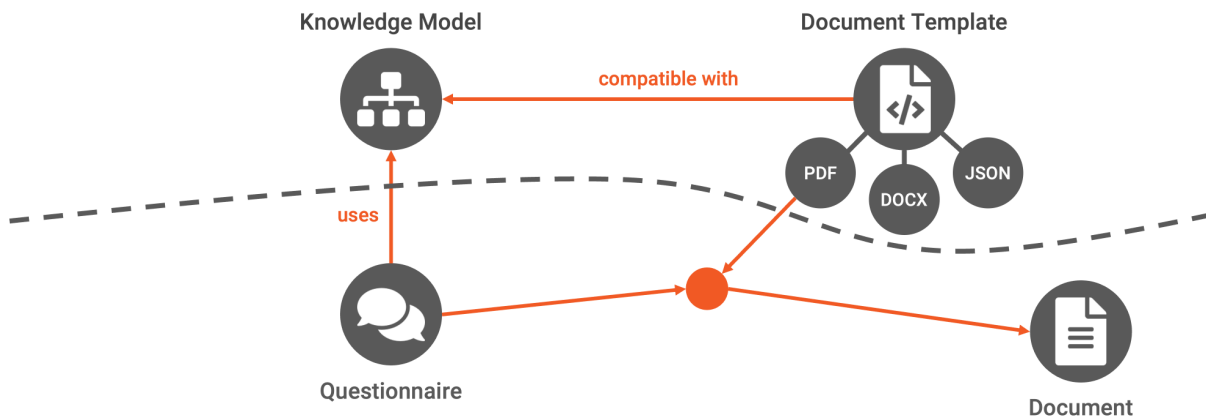| Researcher	Data Steward	Admin
Introduction	Introduction	Introduction
Overview	*Overview*	*Overview*
Project	*Knowledge Model*	
	Document Template	
Application	Application	Application
Projects	*Knowledge Models*	*Documents*
	Document Templates	*Administration*
	More	More
	Document Template Development	*Submission Service*
	Integration Questions	
	Project Importers	

1.1 Introduction

This section will cover the essentials of the Data Stewardship Wizard, including its key components and how they operate together to facilitate effective data management for researchers and data stewards.

1.1.1 Overview

Different components in the Data Stewardship Wizard are connected to create a data management plan and help with data management in general. Different components are typically made and used by different user roles. Data stewards work on preparing content (such as **knowledge models** or **document templates**) for researchers that they can use to work on their data management plans while filling in the **questionnaires** and exporting **documents**.

Fig. 1: Different components of the Data Stewardship Wizard and roles interacting with them.

Knowledge Model

The knowledge model refers to something like a template for the questionnaire. However, it is not linear but has a tree-like structure with different branches based on the previous answer. Therefore, even if it can be complex overall, only the specific questions are used in the questionnaire.

Learn more about knowledge models →

Document Template

While the knowledge model defines the structure of the questionnaire, it does not specify how the resulting document (such as the DMP) will look. We use document templates for that. They transform the answers into documents such as PDF, MS Word, or machine-actionable RDF. This way, we can only answer once and produce different documents.

Learn more about document templates →

Questionnaire

A questionnaire is part of a *project* where researchers fill in their answers regarding their particular research. It uses a specific knowledge model that defines its structure.

Learn more about questionnaires →

Document

Documents are produced from the questionnaire answers and a document template. The document template understands the knowledge model structure and knows how to transform the questionnaire answers into a specific document in the selected format. The documents are saved within a project where they were created from the questionnaire.

Learn more about documents →

1.1.2 Knowledge Model

The knowledge model is a tree-like structure of chapters, questions, answers, and other entities that servers as a template for the questionnaire. All the different questions, their possible answers and follow-up questions, metrics and more is defined there.

While all the possibilities are defined in the knowledge model, when researchers use it to create their project, they don't see everything, but only the top-level questions and more detailed questions are only asked if relevant to their use case.

Knowledge models are created by data stewards in the *knowledge model editor*.

Knowledge Model Structure

Knowledge model consists of several entities connected together. You can see how they are connected in the following diagram and read more details about them below.

Knowlege Model

At the top level, the knowledge model contains *chapters*, and entities refered to elsewhere from the knowledge model: *metrics*, *phases*, *question tags*, and *integrations*.

Chapter

The knowledge model consists of chapters at the top level. Each chapter has a **name**, a **description** and a list of *questions*. Usually, chapters are used to group the questions on the same topic together.

Question

Questions are used to collect the answers from users. Each question has a **title** (the actual question), a **description**, a **phase** when it becomes desirable, a list of *references* and *experts*, and a selection of *question tags*.

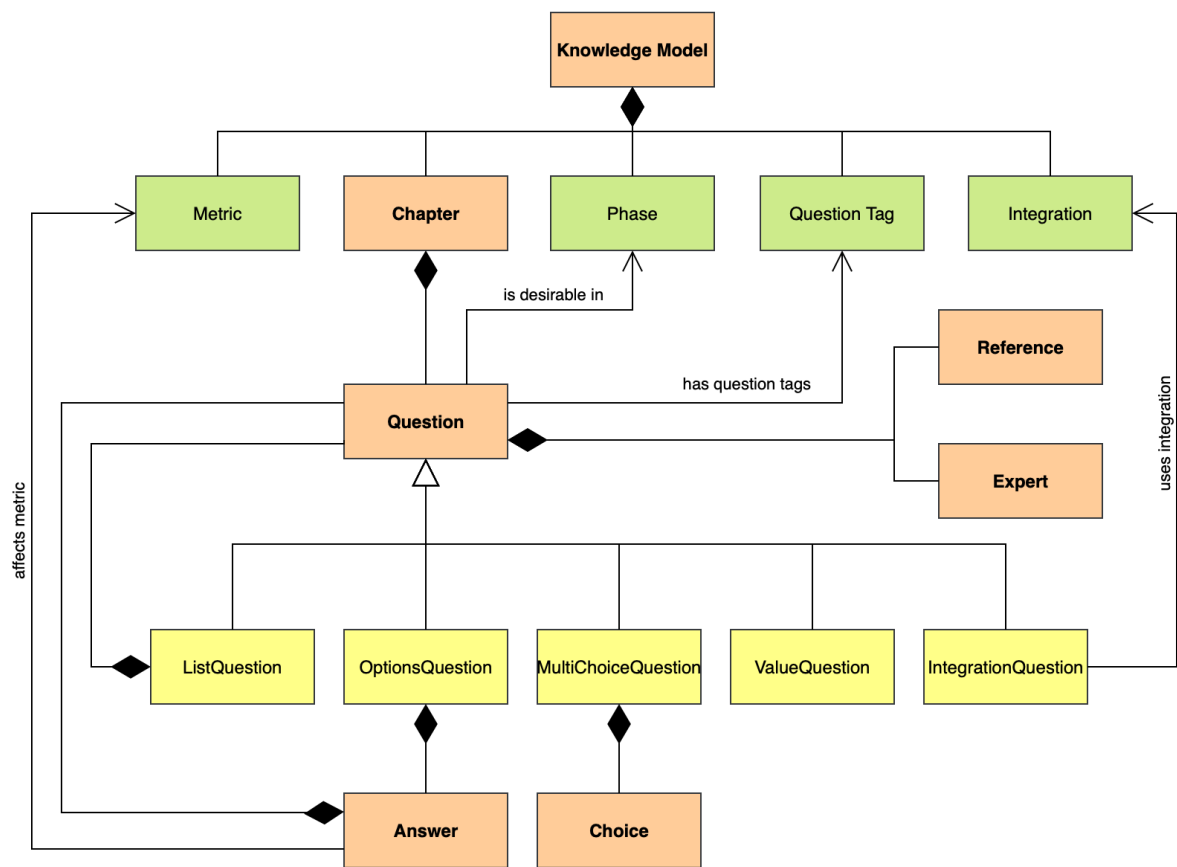Then there are some additional settings based on the **question type**.

Fig. 2: Knowledge model schema

Options Question

The options question contains a closed list of *answers* where users can pick one. Answers can have some follow-up questions that are only presented to the user when they pick the answer. So the options question can be used for questionnaire branching.

List Question

The list question is used when there are multiple answers and we want to ask more details about those. For example, we can ask about different datasets that will be produced – users will have multiple datasets but we want to ask the same questions for each of those. For that, we configure the **item template**, which defines the questions for each item.

Value Question

The value question asks for a single value that users type in. There are many different types of the value question that can be used:

- String
- Number
- Date
- Date Time
- Time
- Text
- Email
- URL
- Color

The input field differs based on the value type (simple input for string, date picker for date, etc.). Some of these have a check whether the entered value is valid (such as valid email or URL) and displays a warning if not.

Integration Question

The integration question is connected to an external resource where the users can pick the answer from. We need to select an *integration* that the question uses and sometimes additional properties, based on the integration configuration.

Users can then search the external resource through the questionnaire and choose the answer. The advantage is that the answer is not only the text but also a link or PID of the selected item making it more FAIR.

If the desired answer is not present in the external resource, users can still fill in a text answer themselves.

Multi-Choice Question

The multi-choice question has a list of *choices*. Users can then pick as many of those choices as they wish. There are, however, no follow-up questions available for this question type.

Answer

An answer is used with *options questions*. It contains a **label** which is the answer itself. Then an **advice** which is visible only if the answer is selected. We can use this when users pick answer that is not great to provide them further guidance on how to improve.

Answers can have **follow-up questions** that are only visible if the answer is selected. We can use this to ask only relevant questions based on the previous answers.

If there are some *metrics* created in the knowledge model, we can configure how each answer affects them. The result for each metric is eventually calculated as a weighted average of all answers affecting that metric. Therefore, we need to configure:

- **weight** [0..1] - how important the answer is (0 = not important at all, 1 = very important)
- **measure** [0..1] - how it affects the metric (0 = bad, 1 = good)

Choice

A choice is used with *multi-choice questions*. It only contains a **label** which is presented to the user.

Reference

We can provide some additional references for *questions* to help users better understand it or learn more details. There are more types of references.

URL Reference

A URL reference is a simple link to any website. It has **URL** which is the actual link and a **label** that describes what the reference is about.

Book Reference

> **Warning:** Book references are deprecated.

Resource Page Reference

> **Warning:** Resource page references are not yet implemented.

Expert

We can provide a contact information to an expert for some *questions*. An expert has a **name** and an **email**. We can use this, for example, if there is an expert for a specific topics in our institution and we want to make it easy to find out in our customized knowledge model.

Metric

We can define metrics for each knowledge model based on our needs. Each metric has a **title**, an **abbreviation**, and a **description**. Once the metric is defined, we can configure which *answers* affect it and how.

This can be use, for example, to define the FAIR metrics:

- **F** - Findability
- **A** - Accessibility
- **I** - Interoperability
- **R** - Reusability

And then define which answers affect which FAIR metrics to provide more feedback to the researchers.

How answer affect certain metric is then set up in the *answer*.

Phase

We can create phases to reflect the workflow. Such as: *Before submitting the proposal*, *Before submitting the DMP*, etc. Each phase has a **title** and a **description**.

Once we have phases defined, we can assign them to *questions* to indicate where each question become desirable. The phases implicitly follow the order in which they are in the knowledge model and the question is considered desirable from the defined phase and on. So for example, if a question is desirable in *Phase 2*, it is implicitly desirable in *Phase 3*, *Phase 4*, etc.

Question Tag

We can define question tags on the knowledge model and then assign them to different *questions*. This can be used to group together questions on the same topic or for the same purpose.

When researchers create a new project from the knowledge model, they can only choose the question groups they are interested in for their research. So we can use this to create a very rich knowledge model but researchers will be able to use only the parts relevant to them.

Integration

Integrations define a connection to an external service or resource where we can get the answers from. They are used with *integration questions*. For each integration we configure some basic information, such as **ID**, **Name**, or **Logo URL**. Other configuration varies based on the integration type. More information about how to configure integration is available under the *integration questions documentation*.

API Integration

API integration connects to an external service API to search for the answers. We need to provide some **request** and **response** configuration, so DSW can use the API.

Widget Integration

Widget integration doesn't use an API but a widget implemented using the DSW Integration Widget SDK. Then we need to configure the **widget URL** where the widget is deployed.

Annotations

Annotations are arbitrary key value pairs that can be assigned to any entity in the knowledge model. These can provide some additional information for the document templates.

Knowledge Model Customizations

A knowledge model doesn't have to be created from scratch. Instead, it can be created as a customization of an existing knowledge model.

We can choose any existing knowledge model and customize it to our needs. We can add, modify, or remove any entities. If there are newer changes in the parent knowledge model, it is possible to get them into our child knowledge model using the knowledge model migration.

1.1.3 Document Template

Document templates transform the answers from a questionnaire to a document of a specific format. The document template usually follow some standard template, such as Horizon Europe DMP, and can support different formats, such as PDF or MS Word. The formats can be basically any text-based format, so it can also be for example a JSON or XML. It follows that we can easily use them to crate a machine-actionable output, too.
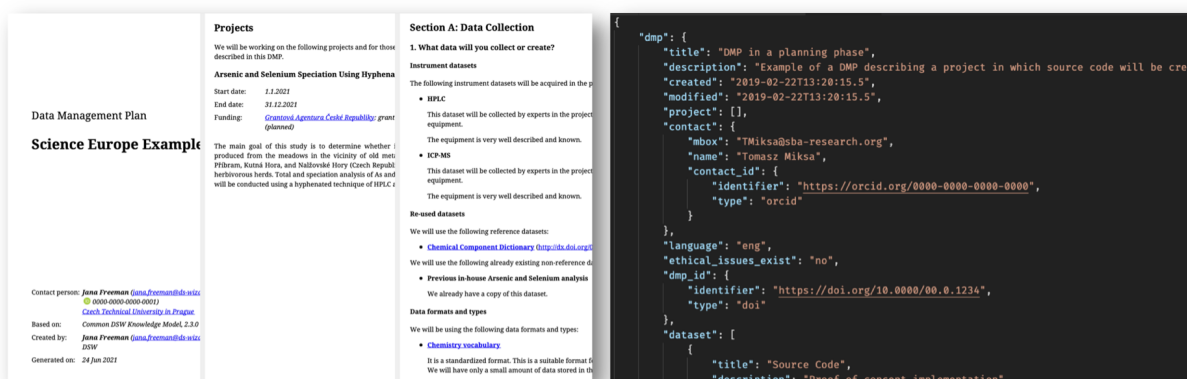

Fig. 3: Document templates can transform the answers from questionnaires to different formats, such as human-readable PDF or machine-actionable JSON.

It is important to note that the document template always define what *knowledge models* it is compatibly with because in order to transform the answers to the document it needs to understand what the questions are and how the answers should be composed into a document.

We can get existing document templates in the *DSW Registry*, or create our own using *document template editor* or *DSW Template Development Kit (TDK)*.

1.1.4 Project

A project is a central part of where we work on our data management plans. It is based on a *knowledge model* and uses one or more *document template* for generating documents.

Projects are fully collaborative, so we can share them with other researchers and work together.

Questionnaire

The questionnaire is part of the project where we fill in our answers. It is generated based on the selected knowledge model. It shows only the questions that are relevant based on our previous answers. There are many more features to provide guidance and embrace collaboration, such as FAIR metrics, comments, TODOs, or version history.

Learn how to work with the questionnaire →

Documents

The outcome of our endeavors in DSW should be the data management plan, the document. Once we have enough answers to the questionnaire, we can generate a document using a document template. These documents are saved within the project. We can create as many as we wish with different document templates and formats. We can also set up a default document template for the project and quickly see a preview of the document.

Learn how to work with the documents →

1.2 Knowledge Models

This section covers how to manage existing *knowledge models* and how to create new ones and make them available for researchers.

1.2.1 Knowledge Model List

As **data stewards** and **admins**, we can manage the knowledge models that are in our DSW instance. Then, **researchers** are using and browsing the knowledge models. We can access the list of knowledge model from the main menu via *Knowledge Model*. The list can be filtered and sorted by name or creation date.

For each knowledge model (KM), we can see the latest version in the list. If we want to read more about a specific KM or see the older versions, we need to access the *Knowledge Model Detail* by clicking the name of KM or clicking *Open* from the right item menu (three dots). There are also other options for each item:

- *Preview* to see how *Projects* generated using this KM would look like.
- *Export* for exporting the latest version of the KM as a file.
- *Create KM editor* is a shortcut for *Create Knowledge Model Editor* for creating a new version.
- *Fork KM* is again a shortcut for *Create Knowledge Model Editor* for to create a fork (some more specific KM based on this one).
- *Create project* is a shortcut to *Create Project* with this KM.

- *Set deprecated* or *Restore* for setting a KM deprecated when we no longer want the **researchers** to use it.

- *Delete* for all versions of the KM (possible only if is not used in any projects or linked in other KMs and editors).

Note: The options in the item menu are based on the role of a current user, e.g. a **researcher** cannot create KM editor.

For **data stewards** and **admins**, *update available* may appear if there is a newer version of the knowledge model in the DSW Registry (and if configured).

Finally, there is an option to *Knowledge Model Import* by click the *Import* button in the top right part of the screen.

Fig. 4: List of all knowledge models with actions.

Knowledge Model Import

We can import an existing knowledge model by navigating to *Knowledge Model List* (*Knowledge Models*) in the main menu and then clicking on *Import* button.

From DSW Registry

If the DSW instance is connected to the DSW Registry, it is possible to import knowledge models from it by entering the **knowledge model ID** of desired template (e.g. `dsw:lifesciences:2.4.0`) and pressing the *Import* button.

Note: In case of knowledge model present in the DSW Registry, we will be notified about the available upgrades.

Import Knowledge Model

☁ From registry   ⬆ From file

dsw:root:2.6.4   **Import**

You can find Knowledge Models in DSW Registry.

Fig. 5: Input for importing a knowledge model from DSW Registry.

From file

We can import a knowledge model as a KM file. Such a file can be created as an export from DSW (from *Knowledge Model List* or *Knowledge Model Detail*).

Import Knowledge Model

☁ From registry   ⬆ From file

Choose a file

Or just drop it here

Fig. 6: Input for importing a knowledge model using a KM package.

Knowledge Model Detail

We can visit a knowledge model detail by clicking on a desired KM in the *Knowledge Model List* (or selecting *View detail* from the right item menu). The detail shows basic information about the knowledge model such as its name, ID, version, license, metamodel version, or (if applicable) what is the parent knowledge model).

The main part of the detail is the README of the KM that should contain basic information and changelog. In the right panel under the basic information, we can navigate to other versions of the KM or navigate to the DSW Registry (if the KM is present there).

In the top bar, we can *Export* the knowledge model as a KM file or *Delete* this version of the knowledge model (only if it is not already used for some projects or other KMs and editors).

In the top pane, we can see the options based on our role:

- *Preview* can be used to check the content of the KM via the *Knowledge Model Preview* feature.

- *Export* for exporting the latest version of the KM as a file.

- *Create KM editor* is a shortcut for *Create Knowledge Model Editor* for creating a new version.

- *Fork KM* is again a shortcut for *Create Knowledge Model Editor* for to create a fork (some more specific KM based on this one).

- *Create project* is a shortcut to *Create Project* with this KM.

- *Set deprecated* or *Restore* for setting a KM deprecated when we no longer want the **researchers** to use it.

- *Delete* the specific version of the KM (possible only if is not used in any projects or linked in other KMs and editors).

If we are not seeing the latest version of the KM, a warning message is shown in the top. Similarly, we will see a notification that update is available if there is a newer version in the DSW Registry (if configured).

Knowledge Model Preview

The preview feature for knowledge models allows us to check the content, i.e. how the questionnaire looks like. It can be used to navigate just through the top-level questions. However, with a know UUID of the question, we can create a direct link to a certain question:

```
https://researchers.ds-wizard.org/knowledge-models/dsw:root:2.4.4/preview?
→questionUuid=0b12fb8c-ee0f-40c0-9c53-b6826b786a0c
```

For a nested question, all needed question above in the tree will be automatically prefilled. The KM preview may be also available to non-logged-in users, if configured by administrators in *Knowledge Models Settings*.

A user can directly create a new project from the KM preview. Again, this can be available also for non-logged-in users if anonymous projects are enabled by administrations in *Projects Settings*.

Fig. 7: Detail of a knowledge model.

1.2.2 Knowledge Model Editors

Here, we can see a list of all knoweldge model editors. Everyone with the data steward role assigned can see all the knowledge model editors.

Fig. 8: List of knowledge model editors.

We can use the search field to search for a specific KM editor. The editors are sorted by when they were last updated but we can change that.

We can *create a new knowledge model editor* by clicking the *Create* button.

By clicking the triple dots on each of the item in the list we can access some actions:

- **Open Editor** - simply open the *editor detail*

- **Upgrade** - if there is a newer version of parent knowledge model, we can use upgrade action to start a *knowledge model migration*, otherwise the action is not visible
- **Publish** - to *publish* a new version of the knowledge model
- **Delete** - to delete the knowledge model editor (cannot be undone)

If there is an ongoing *knowledge model migration*, there are different actions:

- **Continue migration**
- **Cancel migration**

Create Knowledge Model Editor

We can create a new knowledge model editor by navigating to *Knowledge Models → Editors* in the main menu and then clicking the *Create* button.

Every knowledge model needs to have a **name**, a **knowledge model ID** and **version**. The name should be something descriptive to help users understand what the knowledge model is about. The knowledge model ID is used for the identification together with the *organization ID* and knowledge model version after it is published. So the identifier of the knowledge model is:

```
<organizationId>:<knowledgeModelId>:<version>
```

We can create a new project either from scratch, i.e. the new knowledge model will be empty and we will build it all ourselves, or based on an existing knowledge models, which means that everything from the chosen knowledge model will be copied to ours. We can start from there and add, delete, or modify the existing entities in there. We just need to choose the original knowledge model in the **based on** field. Alternatively, we can open the *knowledge model detail* and click on *Fork KM* there.

We can only have one knowledge model editor with the same knowledge model ID. If we deleted the editor but want to continue working on that knowledge model, we can create a new editor with the same knowledge model ID. Or we can open the *knowledge model detail* and click on *Create KM editor* there to have the editor create form prefilled.

Knowledge Model Editor

Knowledge model editor is where we build knowledge models. In this section, we will see what entities we can add there, how they are connected, how to work with the editor and how to publish the knowledge model.

Knowledge Model

The *Knowledge Model* tab is where we work on the entities (such as questions or answers) that appear in the knowledge model. We define the structure here.

Navigation

In the top row, we can see the breadcrumbs that show us where we are within the knowledge model. This allows us to quickly jump back up in the hierarchy.

On the lefthand side of the editor, there is a navigation tree reflecting the structure of the knowledge model. We can click on the arrows to expand or collapse individual entities, or click on *Expand all* or *Collapse all*. By clicking on the entity name, we open its editor.

Create Knowledge Model

Name

Knowledge Model ID

Knowledge Model ID can contain alphanumeric characters and dashes but cannot start or end with a dash.

New version

. .

Suggestions: 1.0.0   0.1.0   0.0.1

The version number is in format X.Y.Z. Increasing number Z indicates only some fixes, number Y minor changes, and number X indicates a major change.

Based on

--

You can create a new Knowledge Model based on the existing one or start from scratch.

Cancel   Create
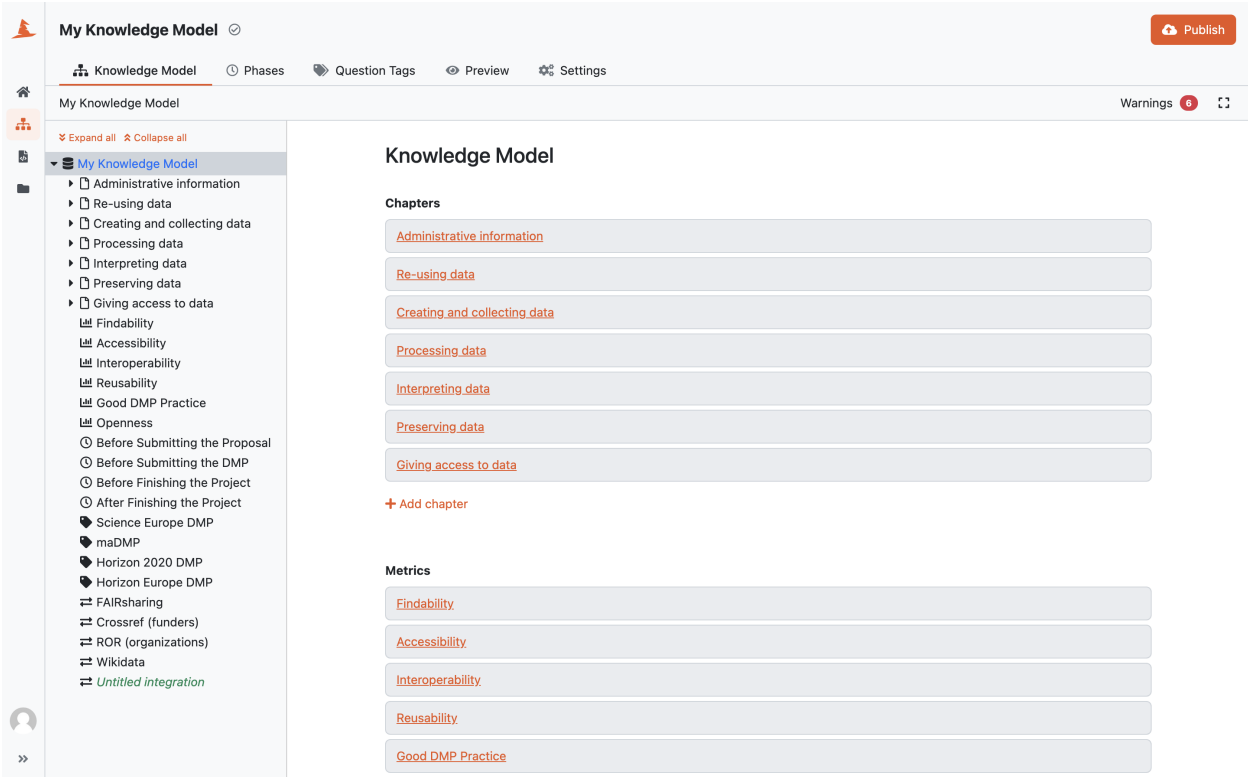
Fig. 9: Form for creating a new knowledge model.

Fig. 10: Knowledge model editor.

Tip: There is no search, however, if we want to quickly find an entity by name we can click on *Expand all* in the navigation tree and use search functionality of our web browser.

Editors

The main area of the *Knowledge Model* tab is the actual editor. The form fields change based on the entity we edit, but there are some shared actions:

- **Copy UUID** - every entity has a generated UUID, we can use this button to copy it. We usually need it for *document template development*.

- **Move** - we can move entities around the knowledge model. However, not every entity can be put under everything. We can open the move modal window and see where the current entity could be move to.

- **Delete** - delete is simply used for deleting the entities. This action cannot be undone, so we need to be careful what we delete.

There are different entities we can edit in the knowledge model, the editor shows different fields based on what we edit:

- *Knowledge Model*
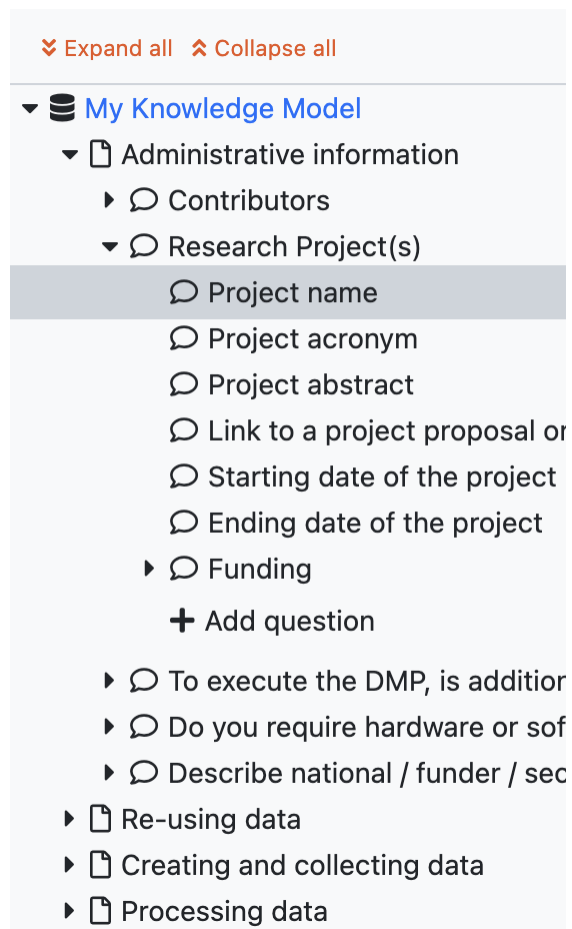
- *Chapter*

- *Question*

- *Answer*

Fig. 11: Knowledge model editor navigation.

Fig. 12: Editor action buttons.

- *Choice*
- *Reference*
- *Expert*
- *Metric*
- *Phase*
- *Question Tag*
- *Integration*

Question

f0ef08fd　Move　Delete

Type

Value

Title

Project name

Text

Editor　Preview

You can use Markdown and see the result in the preview tab.

When does this question become desirable?

Before Submitting the Proposal

Fig. 13: Example of question editor form.

Besides their own fields, each entity has so called **Annotations**. They are arbitrary key value pairs that can be assigned to the entity and used later, when *developing a document template*.

Warnings

The editor checks for some possible problems, such as empty title for a chapter or no answers for an options question. If there are any, the *Warnings* tab appear and we can quickly navigate to those problems and fix them.
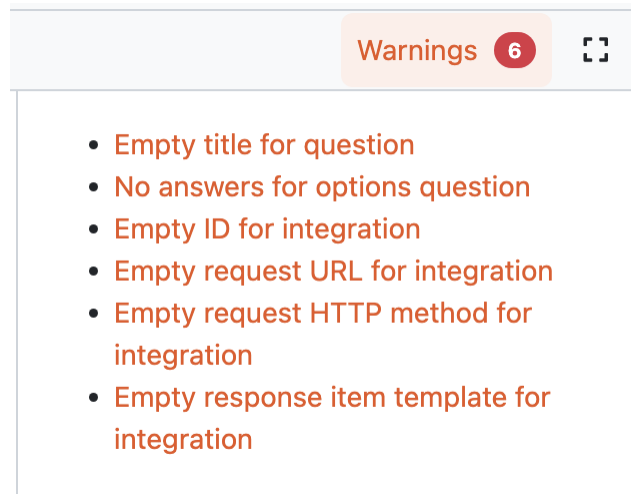
Warnings **6**

- Empty title for question
- No answers for options question
- Empty ID for integration
- Empty request URL for integration
- Empty request HTTP method for integration
- Empty response item template for integration

Fig. 14: Warnings in the knowledge model editor.

Phases

If there are any *project phases* set up in the knowledge model, we can use the *Phases* tab to see an overview of which questions were assigned to be completed in each phase and to change the phase assignments.

We can simply click to the checkbox in the corresponding phase - question table cell to assign/unassign the phase to that particular question.

Question Tags

If there are any *question tags* set up in the knowledge model, we can use the *Question Tags* tab to see the overview of what questions were these tags assigned to, and to assign the question tags to the questions.

We can simply click to the checkbox in the corresponding question tag - question table cell to assign/unassign the question tag to that particular question.

Preview

We can use *Preview* tab to quickly see how the resulting questionnaire would look like. If there are any *question tags* set up, we can also try different combinations of selected tags and observe the changes in the questionnaire.

My Knowledge Model ⊘

🔲 Publish

⊹ Knowledge Model ⏱ Phases 🏷 Question Tags 👁 Preview ⚙ Settings

	Before Submitting the Proposal	Before Submitting the DMP	Before Finishing the Project	After Finishing the Project
🗋 **Administrative information**				
⌾ Contributors	☐	☑	☐	☐
🗋 Item Template				
⌾ Name	☐	☑	☐	☐
⌾ E-mail address	☐	☐	☐	☐
⌾ ORCID Identifier	☐	☐	☐	☐
⌾ Affiliation	☐	☐	☐	☐
⌾ Role	☐	☑	☐	☐
⌾ Research Project(s)	☑	☐	☐	☐
🗋 Item Template				
⌾ Project name	☑	☐	☐	☐
⌾ Project acronym	☐	☐	☐	☐
⌾ Project abstract	☑	☐	☐	☐
⌾ Link to a project proposal or another description of the methods used in the project	☐	☐	☐	☐
⌾ Starting date of the project	☐	☑	☐	☐
⌾ Ending date of the project	☐	☑	☐	☐
⌾ Funding	☑	☐	☐	☐
🗋 Item Template				
⌾ Funder	☐	☐	☐	☐
⌾ Funding status	☐	☐	☑	☐
⌾ Grant number	☐	☐	☐	☐
⌾ To execute the DMP, is additional specialist expertise required?	☐	☑	☐	☐
⊙ Yes, we will be training existing staff				

Fig. 15: Phases editor where we can assign questions to phases.

	Horizon 2020 DMP	Horizon Europe DMP	Science Europe DMP	maDMP
📄 **Administrative information**				
💬 Contributors	☑	☑	☑	☑
📄 Item Template				
💬 Name	☑	☑	☑	☑
💬 E-mail address	☑	☑	☑	☑
💬 ORCID Identifier	☑	☑	☑	☑
💬 Affiliation	☑	☑	☑	☐
💬 Role	☑	☑	☑	☑
💬 Research Project(s)	☑	☑	☑	☑
📄 Item Template				
💬 Project name	☑	☑	☑	☑
💬 Project acronym	☑	☑	☐	☐
💬 Project abstract	☑	☑	☑	☑
💬 Link to a project proposal or another description of the methods used in the project	☐	☐	☐	☐
💬 Starting date of the project	☑	☑	☑	☑
💬 Ending date of the project	☑	☑	☑	☑
💬 Funding	☑	☑	☑	☑
📄 Item Template				
💬 Funder	☑	☑	☑	☑
💬 Funding status	☑	☑	☑	☑
💬 Grant number	☑	☑	☑	☑
💬 To execute the DMP, is additional specialist expertise required?	☑	☑	☑	☐
⊙ Yes, we will be training existing staff				
💬 What kind of training?	☑	☑	☑	☐

Fig. 16: Question tag editor where we can assign question tags to questions.

Fig. 17: Preview of the resulting questionnaire in the knowledge model editor.

Settings

The *Settings* tab, allows us to adjust atributes of the the knowledge model:

- **Name** - should be short name of the Knowledge Model.

- **Description** - this should be really short and descriptive.

- **Knowledge Model ID** - is an unique Knowledge Model identifier.

- **Version** - this is a number indicating which is the latest version of the Knowledge Model, because it can change over time.

- **License** - this is used when we want to share the knowledge model with other people so they know how they can do that. We recommend using a license identifier from SPDX Licens List.

- **Readme** - this is where we can describe everything we need about the knowledge model. We can, for example, include a changelog of what changed in which version, etc. We can use *Markdown* in this field to provide some nice formatting.

Note: **Name**, **Description** and **Version** are all visible to the researcher, when they select a Knowledge Model for their project. So the **Name** and **Description** should provide them with enough information to select a correct one.

Knowledge Model ID together with the *organization ID* and knowledge model version after it is published are used for the identification. So the identifier of the knowledge model is:

```
<organizationId>:<knowledgeModelId>:<version>
```

For the version number we recommended using similar approach as in semantic versioning. So when we have a version `<major>.<minor>.<patch>`, change in the major version number indicates some breaking changes (deleting questions, significant changes in the questionnaire structure, etc.), change in minor version number indicates some new changes that are backwards compatible (i.e., adding a new question), and change in the patch version number indicate some fixes (such as fixing some typos).

If the knowledge model was based on another knowledge model, we can also see the **Parent Knowledge Model** in the settings.

In the **Danger Zone** we can delete the knowledge model. Once it is deleted it can **no longer be recovered**.

Publish

Before the knowledge model can be used by researchers in their projects, we need to publish it. We can do that by clicking the *Publish* button in the top right corner of the *Knowledge Model Editor*.

If we click the button, we are prompted with the metadata details to check them before publishing. We cannot change anything here, so if we want to change it, we have to press Cancel and edit the details on the *Settings* tab of the *Knowledge Model Editor*.

If we confirm the publishing of the Knowledge Model by clicking *Publish* in the modal window, the Knowledge Model becomes available to all users and is accessible in *Knowledge Model List*.

The Knowledge Model Editor will remain in the *Knowledge Model Editors* list and will be available for any future changes.

Fig. 18: Knowledge Model settings.

Fig. 19: Publish dialog where we can confirm or cancel publishing of the Knowledge Model.

Knowledge Model Migration

Knowledge model can be created either from scratch or based on existing one and further modified. If that happened, we can call the original **Parent KM** and the modification **Child KM**. We can create the Child KM any *published version* of the Parent KM, make some modification and publish a version of the Child KM. However, the Parent KM can evolve, and at some point we might want to have those changes in our Child KM, too.

That is what the knowledge model migration is for. Once a new version of the Parent KM is published, we can start the KM migration where we go through these changes. We can choose whether we want to apply or reject these modification to our Child KM during migration. At the end, we publish a new version of the Child KM with all the selected changes.
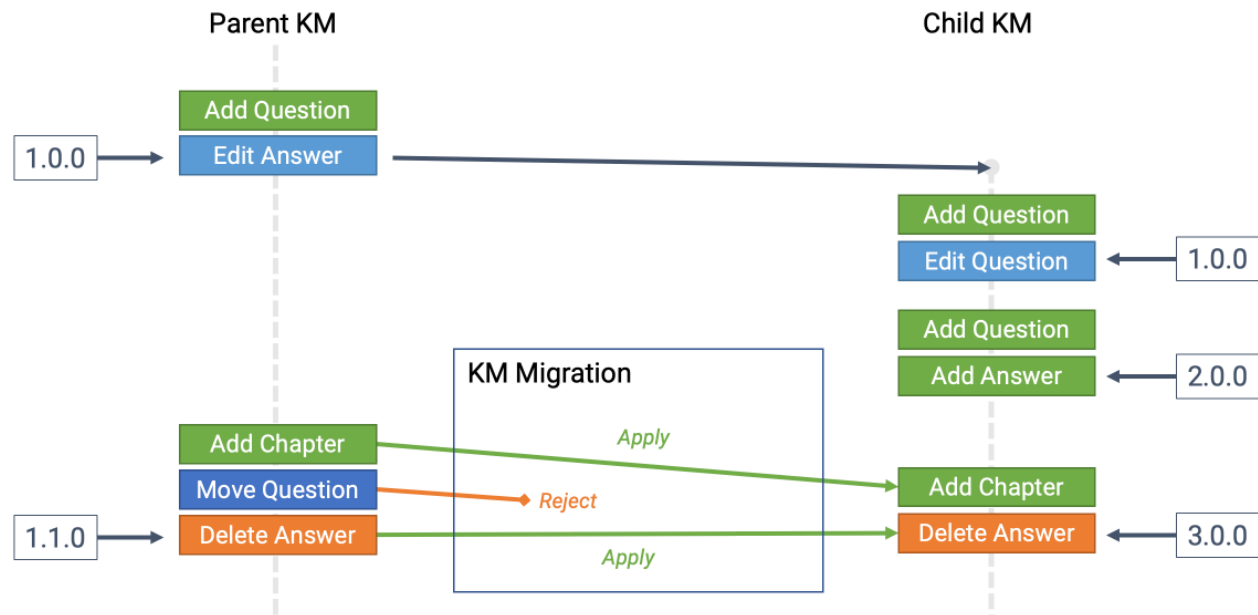
Fig. 20: Schematic representation of KM migration.
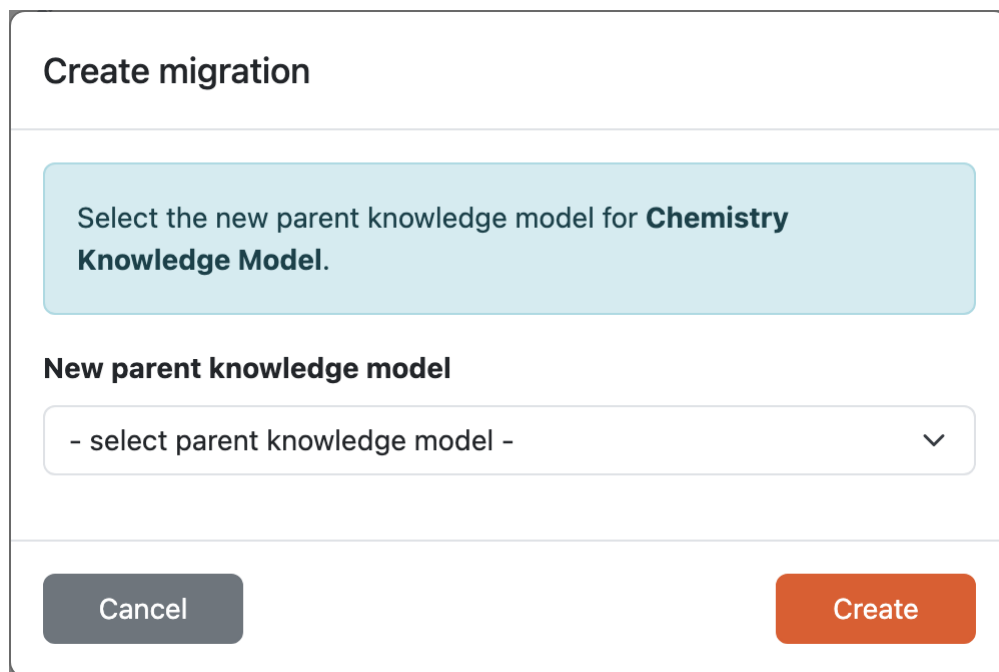
Creating a Knowledge Model Migration

We can start the knowledge model migration from the *list of knowledge model editors*. If there is a newer version of the Parent KM available for particular knowledge model editor, we can see the *update available* badge next to the name of the editor.

Fig. 21: Badge indicating that there is a newer version of the Parent KM.

We can either click on the badge directly, or choose *Upgrade* option from the dropdown menu. It will open a modal window where we can choose a new version of the Parent KM that we want to migrate our Child KM to. Usually, we want to pick the latest.

Fig. 22: Modal window to create a new knowledge model migration.

Knowledge Model Migration

During the migration, we can see all the changes one by one and can choose whether we want to *Apply* or *Reject* the change. We can also choose to *Apply all* if we simply want everything.

Cancelling a Knowledge Model Migration

We can cancell the knowledge model migration at any point before we publish the new version of the Child KM. We need to navigate to the *list of knowledge model editors* and choose *Cancel migration* from the dropdown menu for the desired KM editor.

Finishing a Knowledge Model Migration

After we resolve all the changes, we are ready to publish the new version of the Child KM. To do that, we need to click on the *Publish* → button. This will open the Publish new version screen where we need to provide additional information for the new version of the Knowledge Model.

The publish screen shows us some information about the knowledge model, such as it's **Knowledge Model Name** and **Knowledge Model ID**. We need to choose the new **version number**.

Note: We recommended using similar approach as in semantic versioning. So when we have a version `<major>`. `<minor>`.`<patch>`, change in the major version number indicates some breaking changes (deleting questions, significant changes in the questionnaire structure, etc.), change in minor version number indicates some new changes that are backwards compatible (i.e., adding a new question), and change in the patch version number indicate some fixes (such as fixing some typos).

Then we need to add some additional metadata (these fields are pre-filled if there was a previous version published):
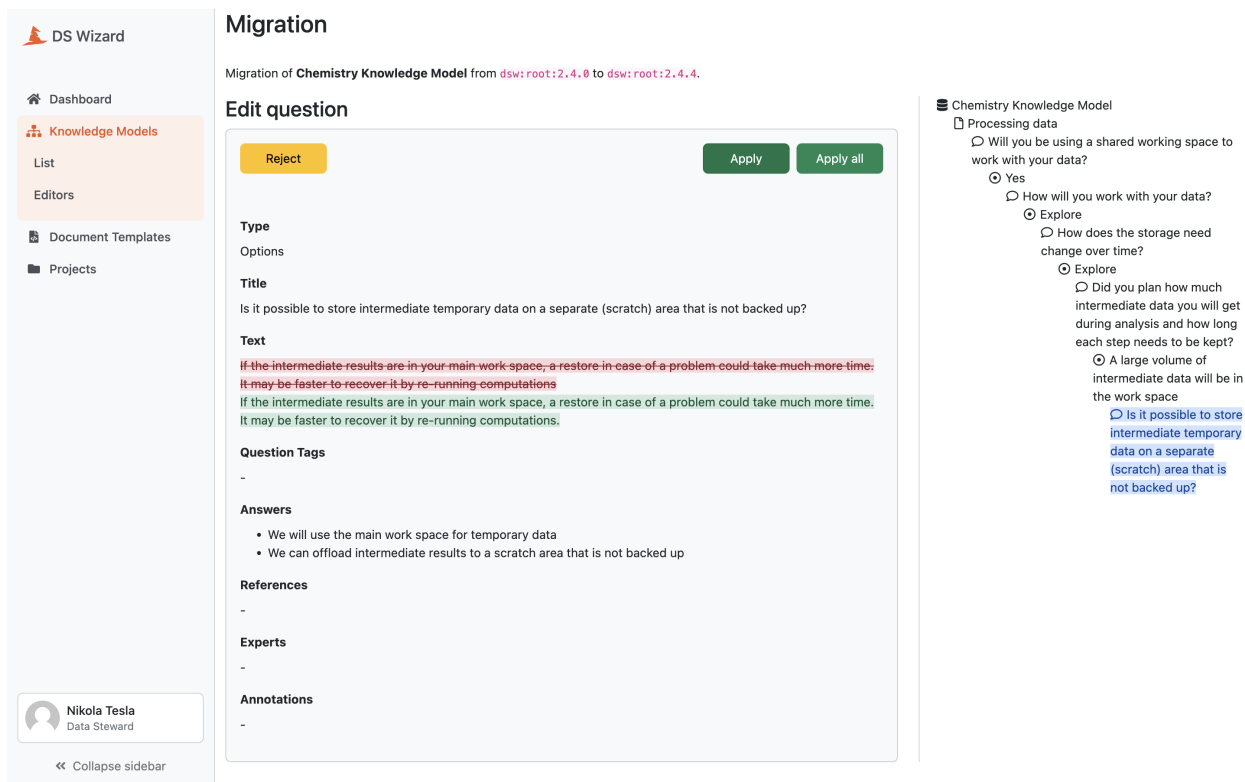
Fig. 23: During the migration we can apply or reject the changes form the Parent KM.

- **License** - this is used when we want to share the knowledge model with other people so they know how they can do that. We recommend using a license identifier from SPDX Licens List.

- **Description** - this should be really short and descriptive. It is used, for example, in select boxes when creating a new project to help researchers choose the best knowledge model for their use case.

- **Readme** - this is where we can describe everything we need about the knowledge model. We can, for example, include a changelog of what changed in what version, etc. We can use *Markdown* in this field to provide some nice formatting.

1.3 Document Templates

This section is about managing the document templates and document template editors. Similarly to Knowledge Models, we can manage *Document Template List* or work on new or customise existing templates using *Document Template Editors*.

1.3.1 Document Template List

As **data stewards** and **admins**, we can check and manage all document templates from the list accessible from the main menu via *Document Templates*. The list can be filtered and sorted by name.

For each document template, we can see the latest version present; however, we can see all the versions by accessing the *Document Template Detail* by clicking the name of the template or via *View detail* from the right dropdown item menu. The dropdown menu also offers *Export* of the latest document template or *Delete* of entire template (all its versions).

Note: A document template can be deleted only if it is not used already for documents, projects, or settings.

Each item may be marked with *unsupported metamodel* when the document template is not compatible with the version of DSW. *Metamodel Schemas* are used to define structures that developers can interact with. If the template originates from the DSW Registry and the registry is configured, *update available* will appear.

To see how to configure the registry, read the *FAQ and Deployment Notes*. If your template is not from the registry, you will have to *Publish* a new version via template editor, which will increase the metamodel version automatically.

Finally, we can use *Import* new document templates by clicking the top right button (see *Document Template Import* for details).

Fig. 24: List of all document templates with actions.

Document Template Import

We can import an existing document template by navigating to *Document Template List* (*Document Templates*) in the main menu and then clicking on *Import* button on the list of document templates.

From DSW Registry

If the DSW instance is connected to the DSW Registry, it is possible to import document templates from it by entering the **document template ID** of desired template (e.g. `dsw:questionnaire-report:2.7.1`) and pressing the *Import* button.

Note: In case of document templates present in the DSW Registry, we will be notified about the available upgrades.

Import Document Template

☁ From registry   ⬆ From file

| dsw:questionnaire.report:2.11.0 | Import |

You can find document templates in DSW Registry.

Fig. 25: Input for importing a document template from DSW Registry.

From file

We can import a document template as a ZIP package. Such a package can be created as an export from DSW or using the Template Development Kit (see *Document Template Development*).

Import Document Template

Fig. 26: Input for importing a document template using a ZIP package.

Document Template Detail

We can check a document template detail by clicking on a desired template in the *Document Template List* (or selecting *View detail* from the right dropdown). The detail shows basic information about the template such as its name, ID, version, license, metamodel version, or supported document formats.

The main part of the detail is the README of the template that should contain basic information and changelog for the template. In the right panel under the basic information, we can navigate to other versions of the document template, navigate to the DSW Registry (if the template is present there), or check compatible knowledge model with the template.

In the top bar, we can *Export* the template as a ZIP package or *Delete* this version of the template (only if it is not already used for some documents). We can also quickly navigate to *Create Document Template Editor* by clicking *Create editor*; it will prepare editor creation for a new version of this document template. Finally, there is the possibility *Set deprecated* which will change the state of the document template so it is no longer usable by researchers in their projects (it becomes unavailable).

If we are not seeing the latest version of the template, a warning message is shown in the top. Similarly, we will see a notification that update is available if there is a newer version in the DSW Registry (if configured).

Fig. 27: Detail of a document template.

1.3.2 Document Template Editors

On this page, we can see a list of all document template editors. Everyone with the data steward role assigned can see all the document template editors and use them.

We can use the search field to search for a specific document template editor. The editors are sorted by when they were last updated but we can change that.

We can *Create Document Template Editor* by clicking the *Create* button.

By clicking the triple dots on each of the item in the list we can access some actions:

- **Open Editor** - simply open the *Document Template Editor* (we can also click the name of the editor)
- **Delete** - to delete the document template editor (cannot be undone)
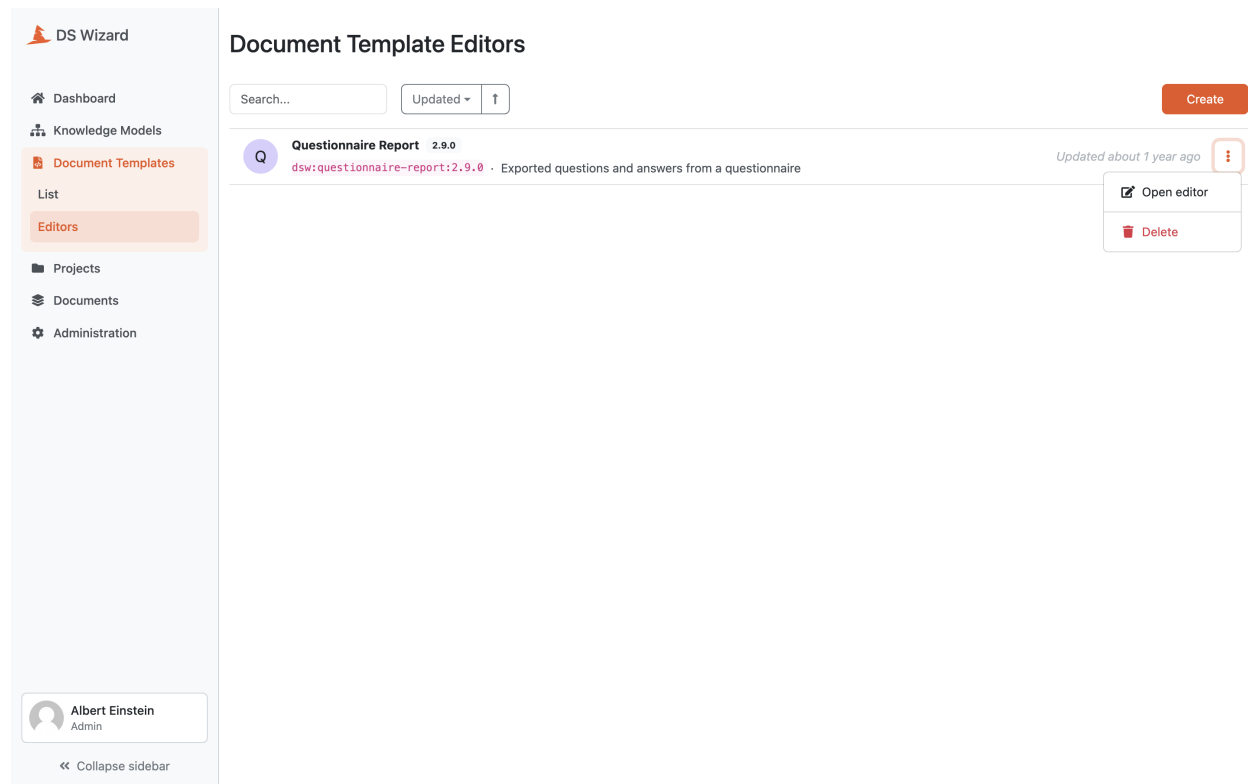
Fig. 28: List of document template editors with actions.

Create Document Template Editor

We can create a new document template editor by navigating to *Document Templates → Editors* in the main menu and then clicking the *Create* button.

Every document template needs to have a **Name**, a **Document Template ID** and **version**. The name should be something descriptive to help users understand what the document template is about. The Document Template ID is used for the identification together with the *organization ID* and document template version that we have to fill as a **New Version**. So the identifier of the document template is:

```
<organizationId>:<documentTemplateId>:<version>
```

Create Document Template

Name

Document Template ID

Document template ID can contain alphanumeric characters and dashes but cannot start or end with a dash.

New version

.   .

Suggestions: 1.0.0   0.1.0   0.0.1

The version number is in format X.Y.Z. Increasing number Z indicates only some fixes, number Y minor changes, and number X indicates a major change.

Based on

--

You can create a new document template based on the existing one or start from scratch.

Cancel   Create

Fig. 29: Form for creating a new document template editor.

We can create a new editor either from scratch, i.e. the new document template will be empty and we will build it all ourselves, or based on an existing document template, which means that everything from the chosen document template (files as well as configuration) will be copied to ours. We just need to choose the original document template in the **Based on** field. Alternatively, we can click on *Create editor* from *Document Template Detail*.

We can only have one document template editor with the same document template ID. If we deleted the editor but want to continue working on that document template, we can create a new editor with the same document template ID. Alternatively, we would have to use a different document template ID.

Document Template Editor

A document template editor allows us to edit both configuration and all files of a document template. We can manage the configuration (such as template metadata, formats, and steps) on the *Settings* tab. The files including directories can be managed (created, edited, deleted, or uploaded) on the *Files* tab. Finally, the *Preview* tab allows us to quickly check how the document template works for a certain project and format and how the resulting document looks like.

There is also option to *Publish* the document template via *Publish* buttom in the top right corner.

Fig. 30: Document template editor.

Files

The *Files* tab in *Document Template Editor* allows us to manage the files and directories (or folders) of the document template. There is a file tree on the left side whereas the main part is containing the built-in text editor. We can create a new folder or text file by clicking *Add* on the of top the file tree and then *File* or *Folder*. Once we enter a desired name, the folder or file is created in the active directory. Alternatively, we can upload a file by clicking *Add* and then *Upload*.

If we select a text file in the file tree, it is opened in the built-in text editor. We can also open multiple files (the editor supports tabs). Moreover, if we have more files opened, we can see a *split view* icon on the top of the file tree. When we click it, it will split the view and move the opened file to the other group. If there are already two groups, it switches the file between the two groups. We can close the file by clicking cross icon in its tab.

We can easily delete a file or a folder; when it is selected in the file tree, we can simply click the red trash icon above the file tree. The deletion must be confirmed in the prompt so we will not delete something by accident as it is not reversible operation.

If we make some changes in a file, the asterisk (or star) symbol will appear by the name in the tree view as well as in the tab (if opened). The changes must be then saved using *Save* button or discarded using *Discard* button in the top right corner. Those buttons will appear instead of *Publish* as it is not possible to publish a document template with unsaved changes. After saving the changes and switching to *Preview* (or refreshing it), the document will be re-generated using the newly changed document template.
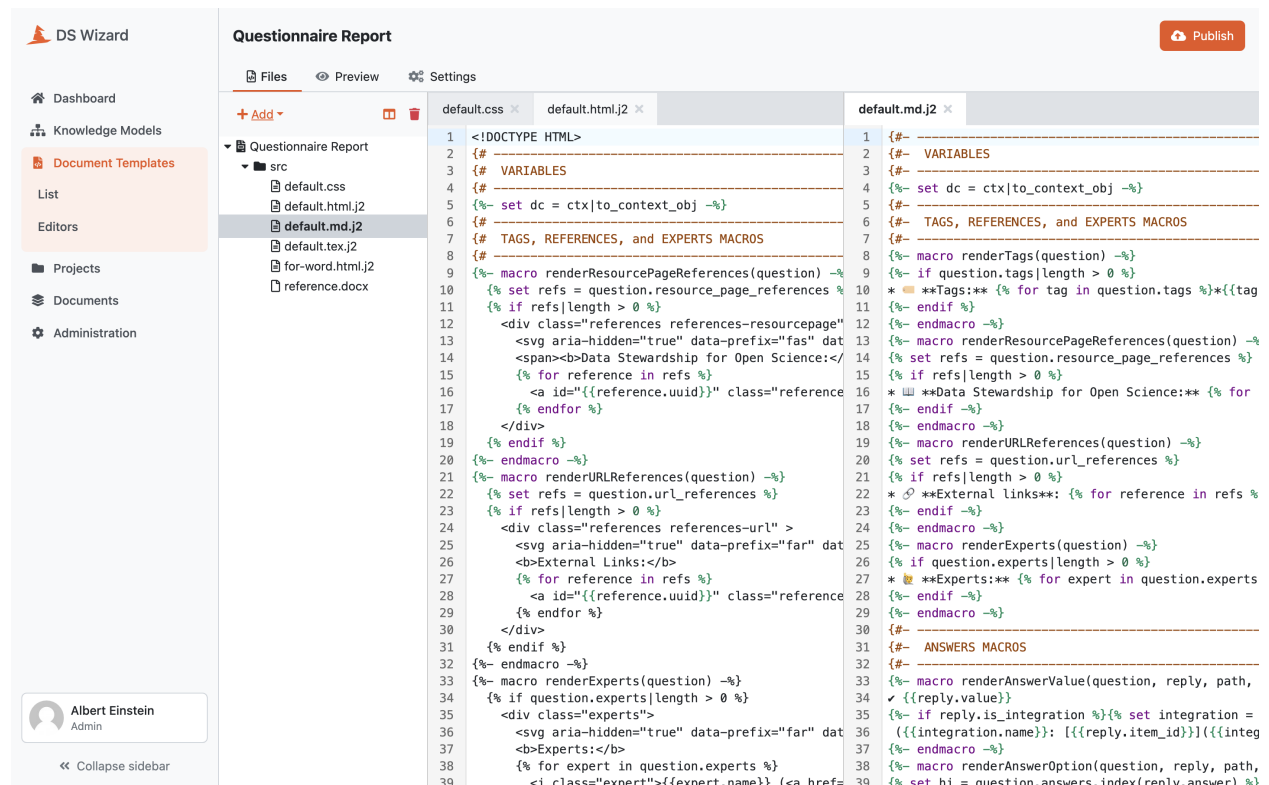
Fig. 31: Files editor with tabs and split view.

Preview

The *Preview* tab of *Document Template Editor* allows us to quickly check how our document template works and how the resulting document looks like for a selected project and document format. We can select any **Project** that we are allowed to view. Then, we select the desired **Format** (one of those specified on the *Settings* tab of the editor). Any change in the template configuration and its files will trigger re-generation of the preview (we can simply click the *Preview* tab again).

Questionnaire Report   ☁ Publish

　📄 Files　　👁 Preview　　⚙ Settings

Project:　--　　　　　　　　　　　　　Format:　--　　　　　　　　⌄

Preview not set

Select project and format you want to preview.
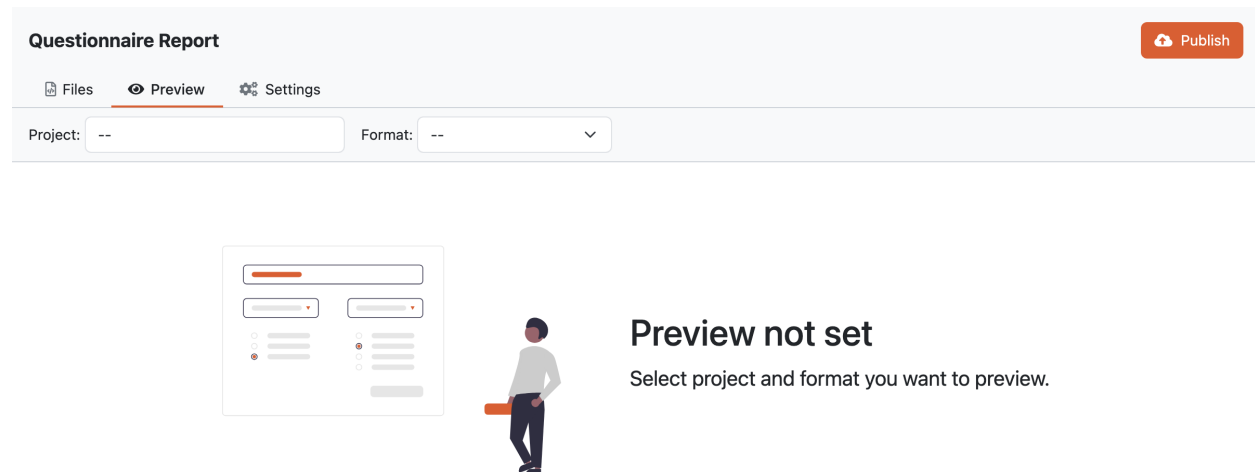
Fig. 32: Preview in a document template editor (without selected **Project** and **Format**).

Note: A good practice is to open the document template editor in a different browser tab than editor opened on the *Files* or *Settings* tab. Moreover, we can also open the project that we use for preview.

Settings

The *Settings* tab of *Document Template Editor* allows us to adjust the configuration and metadata of the document template. It is split to the following three parts:

General

This part allows us to change the metadata about the document template:

- **Name** should be short name of the template.

- **Description** should be short and descriptive (users will see it while selecting a document template).

- **Template ID** is the document template ID (as explained for *Create Document Template Editor*).

- **Version** of the document template.

- **License** should contain a name of used license (e.g. *Apache-2.0* or *unlicensed*).

- **Readme** can contain a longer description, acknowledgements, notes how to use the template, links to resources, and a changelog.

Knowledge Models

We can specify here what knowledge models are compatible with our document template. This is useful to capture if our template is usable only with a certain knowledge model(s) which will guarantee us some content (specific chapters, questions, answers, etc.). For each entry, we are prompted to specify the **Organization ID**, **Knowledge Model ID**, **Min Version**, and **Max Version**. Any of these value can be left empty which means *any value*.

For example, if we set **Organization ID** to *dsw*, **Knowledge Model ID** to *root*, **Min Version** to *2.4.0* and leave **Max Version** empty, it will mean knowledge model *dsw:root:2.4.0* and any higher version. So that example would work for *dsw:root:2.4.4* or *dsw:root:2.5.0* but not for *dsw:root:2.3.0* nor *acme:base-km:2.6.0*

Allowed Knowledge Models

Organization ID	Knowledge Model ID	Min Version	Max Version	
dsw	root	2.4.0		🗑
dsw	lifesciences	2.3.0	2.4.0	🗑

+ Add knowledge model

Fig. 33: Allowed knowledge models specification for a document template.

Formats

Each document template can support multiple file formats and users will be able to select which one they want to use to generate (or preview a document). We can add a new format by clicking *Add* button. Then, each format must have a **Name** and **Icon** (by using Font Awesome).

Each format has some steps which capture how a file for that format is produced. There are different steps defined such as `json`, `jinja2`, `pandoc`, or `wkhtmltopdf` which is used as its **Name**. Then, the step may have certain configuration **Options**. For example, `jinja2` must have `content-type`, `extension`, and `template` specified. All the possible steps and their options are further described in the *Document Template Development*.

Publish

Once we are ready with our document template in the editor, we can publish a new version by using the *Publish* button located in the top right corner of *Document Template Editor*. If we click the button, we can check the metadata details and confirm the publishing. We cannot change anything here, so if we want to make some changes, we have to press *Cancel* and edit the details on the *Settings* tab of the *Document Template Editor*.

If we confirm the publishing of the document template by clicking *Publish* in the modal window, the document template becomes available to all users and is accessible in *Document Template List*. Moreover, the document template editor disappears (as the state of the document template changed). If we want to directly continue in developing a new version, we have to *Create Document Template Editor*.

Fig. 34: Markdown format specification with `jinja` step.

Fig. 35: Confirmation of document template publishing.

1.4 Projects

This section is about *projects* – how to create them, how to work and collaborate on them, how to generate documents, how to upgrade projects, what are the project templates and more.

Projects are mainly used by the researchers. We have the introduction video about how to create project, fill the questionnaire and get the documents.

https://youtu.be/XrI8qYtWSBw

1.4.1 Project List

In the project list, we can see a list of all projects we have access to. Those are the projects where we were assigned to with any role or that are visible by all other logged-in users in the *project sharing settings*. The projects are filtered to those we are explicitly assigned to by default.
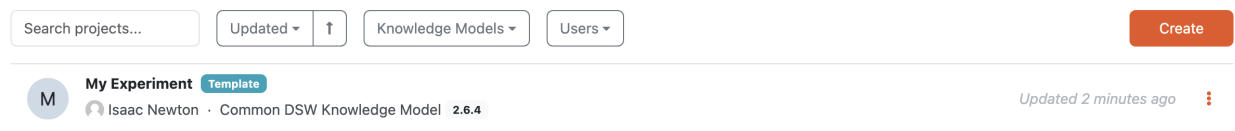
Fig. 36: Project list with filters.

We can search for specific projects using the search field or filter them using additonal filters:

- **Project Template** - if we want to see only *project templates* or regular projects
- **Project Tags** - show only the projects that has specific tags assigned
- **Knowledge Models** - show only the projects created from a specific knowledge model
- **Users** - filter only the projects with specifc users assigned to them

Note: Note that some of the filters can be disabled based on the DSW instance settings or user role.

We can *create a new project* by clicking the *Create* button.

By clicking the triple dots on each of the item in the list we can access some actions:

- **Open project** - will simply open the project
- **Create project from this template** - will create a new project from the selected project template (this is only available if the project is also a project template)
- **Clone** - will create the exact copy of the project
- **Create migration** - will start a *project migration*
- **Delete** - will delete the project (this action cannot be undone)

If there is an ongoing *project migration*, there are different actions:

- **Continue migration** - to come back to the ongoing project migration

- **Cancel migration** - to cancel the ongoing project migration

Create Project

We can create a new project by navigating to *Projects* in the main menu and then clicking on *Create* button on the project list.

Note: If we have **Researcher** role we also have a **Create Project** widget on our dashboard right after logging in. We can click on *Create* button there, too.

Based on our instance configuration, we can create the new project from a project template, custom, or both.

Fig. 37: Different options how to create a project.

From Project Template

There are many options how to create and configure a project, such as what *knowledge model* or *document template* to use. *Project templates* are prepared projects by data stewards with possibly pre-selected knowledge models and document templates, they can have some pre-filled answers, comments and TODOs as well.

So if there are any project templates created in our instance, we can use them to have a smoother start of our project. We just need to give our project a **name** and choose the **project template** from offered options.

Custom

If there are no project templates available or we don't want to use them, we can choose to create a custom project. We need to give our project a **name** again, but this time we choose a *knowledge model* from offered options.

If the knowledge model has *question tags*, we can either choose to create questionnaire with all available questions or filter them by the question tags.

This will create an empty project with only the selected knowledge model and we need to configure everything (such as a document template) ourselves.

Project Detail

Project detail is where we work on our data management plan. In this section, we explore different features from filling in the answers to collaborating with other people or generating documents.

Questionnaire

Questionnaire is the first tab of the project detail. This is the most imporant part where we fill in all the details about our project.

Current Phase

If the *knowledge model* we use for the *project* has *phases* defined, we can see a phase selection in the questionnaire detail. Different questions become desirable based on the selected phase. For example, some should be answered before submitting the proposal, while others can be filled later.

By clicking the phase selection we open the modal window where we can choose the current phase.

We can see the desirability of questions based on the phase we are currently in. We can also see the number of questions that still need to be answered in this phase for each chapter in the chapter list.

There are three desirability states the question can be in:

- **red, with a pen icon** - this question must be answered in the current phase

- **light grey, with an hourglass icon** - this question will have to be answered in some later phase

- **green, with a checkmark icon** - this question has already been answered

Note: If there is no phase defined on the knowledge model, the current phase selection is not visible in the questionnaire detail.

Chapters

Below the current phase selection, we have a list of *chapters*. We can see the number of questions that are yet to be answered in current phase (or overall if there are no phases defined in the knowledge model). We can use this list to navigate freely between chapters.

For the opened chapter, we can see a navigation tree for the chapter structure, showing the questions, follow-up questions, items, etc. We can use this tree to quickly navigate to a specific question in the chapter.

Create Project

Name

| From project template | From knowledge model |

Knowledge models are templates for questionnaires. You can select question tags now and configure the document template and other settings later.

Knowledge Model

> C **Common DSW Knowledge Model** 2.6.4
> DSW Knowledge Model originating from mindmap made by Rob ✕
> Hooft

Question Tags

You can either use all questions from the knowledge model or filter them by question tags.

○ **Use all questions**

◉ **Filter by question tags**

☐ ELSI

☐ Horizon 2020 DMP

☑ Horizon Europe DMP

☐ Science Europe DMP

☑ maDMP

Cancel   Create

Fig. 38: Creating custom project with question tag selection.

Fig. 39: Project detail opened on the questionnaire page.

Fig. 40: Phase selection widget.

Select phase ✕

Before Submitting the Proposal

Some data stewardship issues already need to be addressed during the creation of a proposal for a research project. For example these could be questions that can have a significant impact on the budget or timeline of a project. Only such questions are labeled as "desirable" in this phase.

Before Submitting the DMP

In many research projects, the finished data management plan is one of the early deliverables. Questions that should be addressed in those early months of the project are highlighted as desirable when this phase is selected.

Before Finishing the Project

When a project rounds up, this means that all data has found its final spots. In this phase, questions that document the final state of affairs will be labeled as desirable.

After Finishing the Project

After a project is finished, the team conducts a comprehensive project review to evaluate its overall success and identify areas for future improvement and followup. Relevant feedback is gathered from stakeholders, and the team is finalizing reports and archiving project documentation. Finally, lessons learned are being documented for future reference. Questions relevant for these processes will be labeled as desirable in this phase.

Fig. 41: Phase selection modal window.

Fig. 42: Chapter list showing the also the questions for the opened chapter.

Questionnaire Area

The questionnaire area fills the most space in the questionnaire screen. It displays the questions and answers from the opened chapter.

Each question has an identifier which indicates the chapter it belongs to, as well as its order and nesting within the chapter. For example, **I.1.a.5**, where the Roman numeral represents the number of the chapter, and the remaining numbers indicate the order and nesting of the question. Then there is also the question name.

Some additional information can also be part of the question:

- **Question tags** - can indicate some additional groupping of questions, for example what DMP templates is this question used for
- **Description** - additional information explaining the question
- **Desirability** - what phase this question become desirable in
- **List of references** - links to additional external resources related to the question
- **List of experts** - whom to contact when help is needed with answering the question

Based on our role in the project and specific instance settings there are some additional actions besides answering the question:

- *Add TODO*
- *Add comment*
- Provide feedback for the question

We can get more information on how various collaboration tools work and can be used in *Sharing*.

The most important part is, however, answering the question. The way of how to answer the question differs based on the question type.

The following video tutorial explains questions and different question types in more detail.

https://youtu.be/buG5BjWPs70

Options Question

Options question has a list of pre-set answers and we can choose one from those. There can be some follow-up questions (indicated by the icon by the answer). These questions are displayed only if we select that answer.

If there are some merics set for the answers, we can see labels with the metrics by the answer as well. The color of the label indicates how good or bad the answer is (red means bad, green good, yellow something in between).

List Question

List question doesn't have a simple answer but a list of items. Each of the items has the same set of subquestions. For example, a list question asking about the project contributors where each item represent one contributor with questions about their name, role, etc.

We can simply click on *Add* button under the question to add a new item. Then, we can answer the questions for the item. If the item has a lot of questions, we can use the arrow icon in the item's top left corner to **fold/unfold** the item.

There is a trash bin icon in the item's top right corner that we can use to **delete** the item. If there are more than one item, there are also arrow icons that we can use to **change the order** of the items.

✏ II.2 **Is there any pre-existing data?**   TODO ✕ 💬 🔗

ELSI   Horizon 2020 DMP   Horizon Europe DMP   Science Europe DMP   maDMP

Are there any data sets available in the world that are relevant to your planned research?

☑ Desirable: *Before Submitting the Proposal*
🗎 Data Stewardship for Open Science: *atq*
🔗 External links: *Google dataset search,   Datacite Search,   RDMkit on Reusing Data,   RDMkit on Existing Data*

○ **a. No**

○ **b. Yes** ☰

Fig. 43: Options question with a closed set of answers.

✔ V.1 **List the data formats you will be using for interpretation and describe their structure**   ＋ 💬 🔗

Give each type of data a name that you recognise.

If you have data in many different structures, integrating the data may be more challenging.

☑ Desirable: *Before Submitting the Proposal*
🔗 External links: *RDMkit on Machine Actionability,   RDMkit on Data Processing*

⌄   🗑

  ✔ V.1.a.1 **Data type**   ＋ 💬 🔗

  ☑ Desirable: *Before Submitting the Proposal*

  XML for evolutionary biology and comparative genomics. (phyloXML) type model and format
  PhyloXML is an XML language designed to describe phylogenetic trees (or networks) and associated data. PhyloXML provides elements for commonly used features, such as taxonomic information, gene names and identifiers, branch lengths, support values, and gene duplication and speciation events.

  FAIRsharing https://fairsharing.org/10.25504/FAIRsharing.3mprr8

  ↻ Clear answer

  Answered about 1 hour ago by Albert Einstein.

  ✏ V.1.a.2 **How is this data structured?**   ＋ 💬 🔗

  ☑ Desirable: *Before Submitting the Proposal*

    ○ **a. A structured domain specific file with data and metadata fields** ☰

    ○ **b. A table or set of tables (consisting of 'data records')** ☰

    ○ **c. Complex data, like a graph** ☰

Fig. 44: List question with a single item.

Value Question

Value question contains an input field for our answer. This can be a simple text field (such as asking for a project contributor's name), or some additional widget, for example a date picker.

Some of the value types contains a validation (e.g., email or URL). We can still type in an invalid answer, but it will display a warning and also show it in the list of *warnings*.

Fig. 45: Value question with a simple text input.

Integration Question

Integration question is connected to an external resource where it searches for the answers. The input field works as a search field, so when we start typing something, it will search the external resource and offers us a list of possible answers.

When we pick an answer from the list, we not only have the answer but also **a link to the selected item in the external service**. If the answer we searched for is not there, we can simply keep what we have written in the input field. We just won't have the link with this answer.

Fig. 46: Integration question with a response from FAIRsharing containing also a link.

Multi-Choice Question

Multi-choice question is similar to the options question, however we can choose more there one answer and there are no follow-up questions.

Fig. 47: Multi-choice question with many choices.

View settings

In the questionnaire tab, there is a menu bar with various options. The first one is *View*, where we can show or hide some question details:

- *Answered by* - show/hide who and when answered questions
- *Phases* - show/hide what phase the questions are desirable in
- *Question tags* - show/hide the question tags
- *Non-desirable questions* - show/hide questions that are not desired to be filled in current phase
- *Metric values* - show/hide value of metrics for accessibility purpose
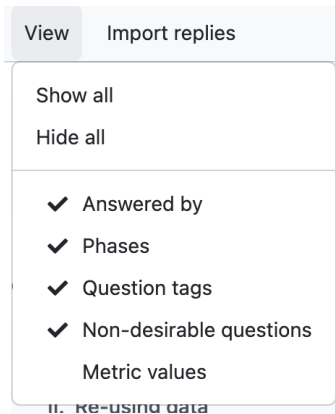
Fig. 48: Different view options accessible form the questionnaire toolbar.

Import answers

Questionnaire answers can be imported for various sources using *project importers*.

If there are some project importers available for the project, there is the *Import answers* button in the questionnaire menu bar. We can choose one of the available importers there and then follow the instructions in the importer window.

Warnings

Some value questions (such as email or URL) validates the answer written there. If it is an invalid value, we will see *Warnings* tab in the questionnaire menu bar with a badge showing the number of warnings. If we click on it, we can see a list of all questions that has a warning and we can click on it to navigate quickly to that question.
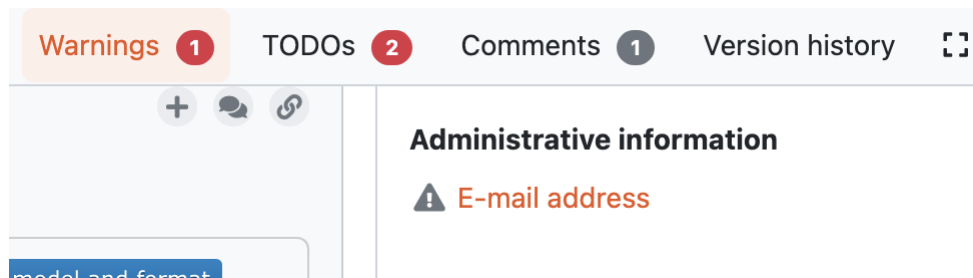
Fig. 49: Warnings refering to questions with invalid values.

Comments

We sometimes want to leave comments to discuss things with our team or just reminders for ourselves. We can write comments to each question in the questionnaire.

Fig. 50: Example of a comment.

Add Comment

To add a comment, we need to click on the comments icon by the question which opens a side panel with all the comments related to that question. Then, we can write the comment into the text box and submit.

View Comments

When there are any comments for a question, the comments icon is changed. It has a yellow color and shows the number of unresolved comments for that question.

In the questionnaire menu bar, there is also a *Comments* tab, showing a badge with the number of comments everywhere in the questionnaire. If we open the tab, we can see a list of questions for each chapter where there are some comments. Clicking on the question there will bring us to that question and open the comments side panel.

Comment Threads

Comments are organized into comment threads for better clarity. We can either start a new thread or reply in an existing thread if our comment is on the same topic.

When the thread is resolved, we can click on the ✓ icon in order to resolve it. Resolved threads can be later viewed by selecting *View resolved comments*. They can also be reopened if needed.

Editor Notes

Besides comments there are also editor notes which work the same way as comments but they are visible only to project editors and owners. We can use editor notes to internal communication with our team while working on the DMP and then comments to gather the feedback from supervisor or reviewer.

TODOs

When we are filling in the questionnaire, we can stumble upon a question that we don't know how to answer yet, but we don't want to forget to come back to that question. We can click on + *Add TODO* to add a TODO to the question.

We can then open the *TODOs* tab from the questionnaire menu to see the list of all questions with assigned TODO in the questionnaire. By clicking on a question there, we can quickly jump back to that question and fill it.
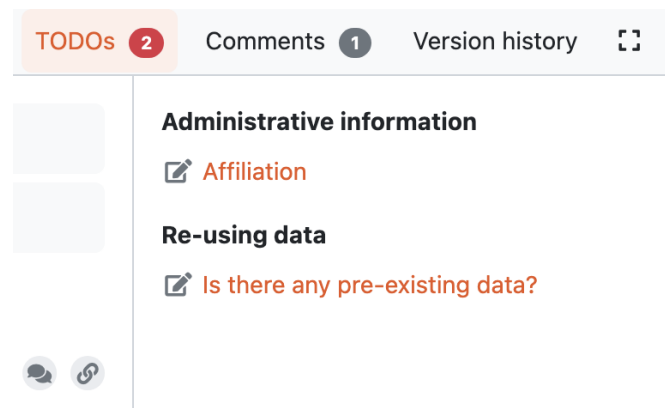
Fig. 51: List of TODOs.

Version History

When we open the *Version history* tab from the questionnaire menu bar we can see the list of all the changes that happened in that questionnaire. We can see who and when made what changes groupped by months and days.

Name a Version

At any point (also retroactively) we can name a version. Click on the triple dots on any event we want to name and choose *Name this version*. Then we just fill in name and description of that version. If the event already has a named version, we can choose *Rename this version* instead.

When we have some named versions, we can choose *Named versions only*. Then, we don't see every single change but only the important versions we gave a name to.

View Questionnaire in a Version

Thanks to the version history, we can see how the questionnaire was filled at any point in the past. We simply find the event in the version history and choose *View questionnaire* from the event menu.

Create Document from an Older Version

Sometimes, we might want to create a document from an older version. For example, we created only a PDF document, but later we find out that we also needed a Word document. To do that, we simply find that version in the version history and select *Create document*. Then, we just fill in the details in the form and create the document.

Revert to an Older Version

We can also revert a questionnaire to an older version. We can simply find the desired version in the version history and choose *Revert to this version* from the event menu.

> **Warning:** Reverting to an older version cannot be undone. It is therefore recommended to create a copy of the project before reverting.

Metrics

In the *Metrics* tab in the project detail we can see a **Summary Report** for the whole questionnaire and then the same details for each individual chapter.

The report shows how many answered questions out of how many are there for the current phase (if there are phases in the knowledge model) and overal.

If there are any metrics in the knowledge model, the report also shows the score for each metric. The score is calculated as a weighted average of all the answers affecting that metric and is always between 0 and 1. If there are at least 3 metrics present, a spider chart is also displayed.

There is also a metrics description at the bottom of the page to better understand what exactly each metric means.

Fig. 52: Version history shows all events changing the project.
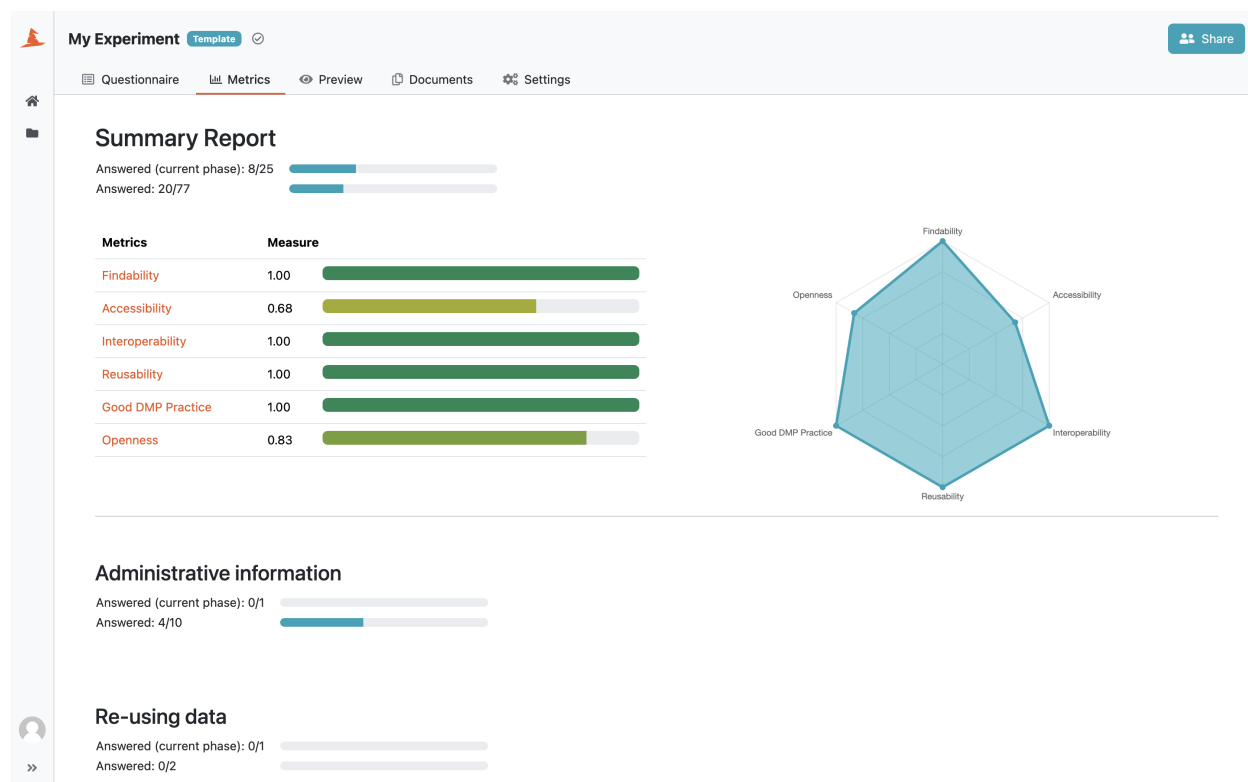
Fig. 53: Summary report of project metrics.

Preview

We can quickly see how the resulting document would look like in the *Preview* tab.

Default Document Template Not Set

The preview uses the default document template. If there is no default document template set in the *project settings*, we need to set it first.

Download Preview

Not all formats can be displayed in the web browser, for example a MS Word document. In that case, we'll be able to download the document instead of previewing it.
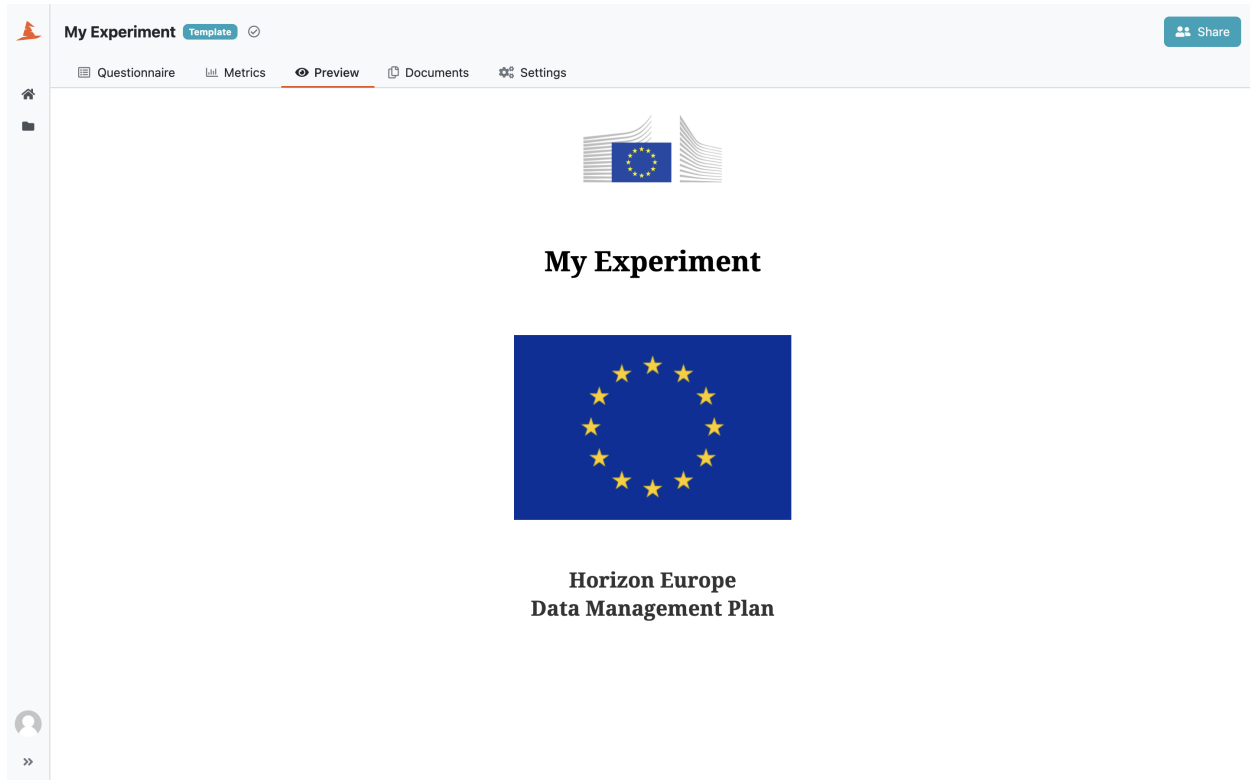
Fig. 54: Document preview in the project detail.
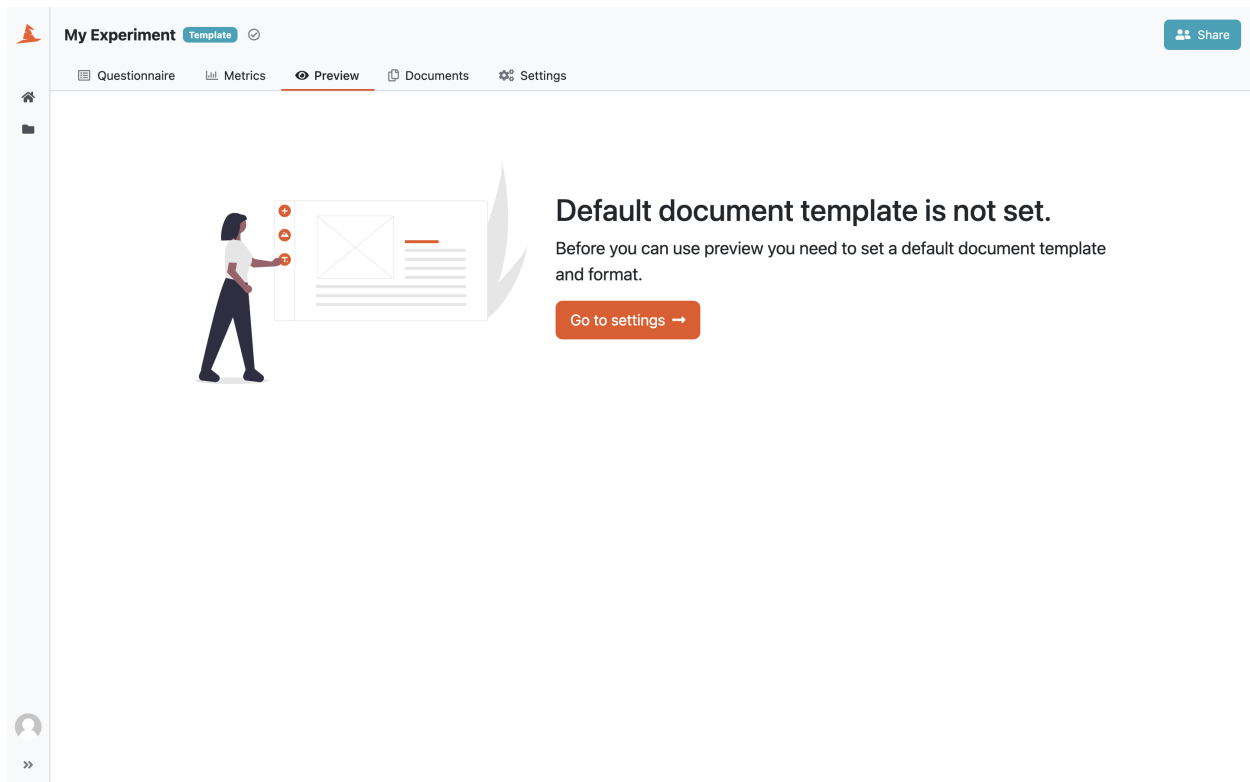
Fig. 55: Preview only works when the default document template is set.
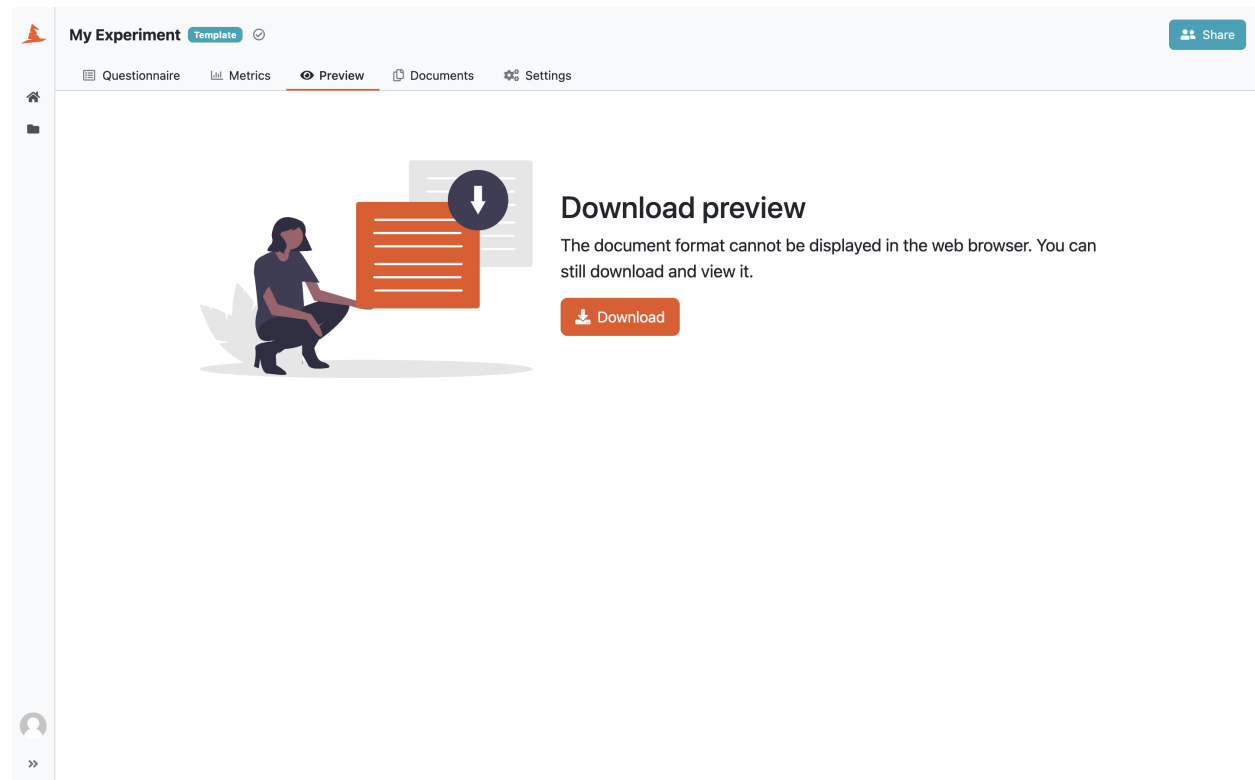
Fig. 56: Some formats cannot be displayed in the web browser.

Documents

In the *Documents* tab in the project detail we can manage the documents related to the project. We can see a list of all the documents with their **format**, **file size**, and **document template** used to create that document.

While we can quickly see how the current state of the questionnaire looks like in the document in the *preview* tab, the documents created here are persistent. That means that the once the document is created, it is immutable and you can always download it later, after you have changed the questionnaire, and it will still be the same.

New document

We can click on *New document* when we want to create a new document. We need to give a name to our new document (project name is prefilled) and choose the **document template** and **format**. If there is a *default document template and format* set for the project, they are prefilled in this form. However, we can change them to whatever we want before creating the document. Once we hit *Create*, we are taken back to the document list and we'll see the new document there (it might take while before it is generated though).
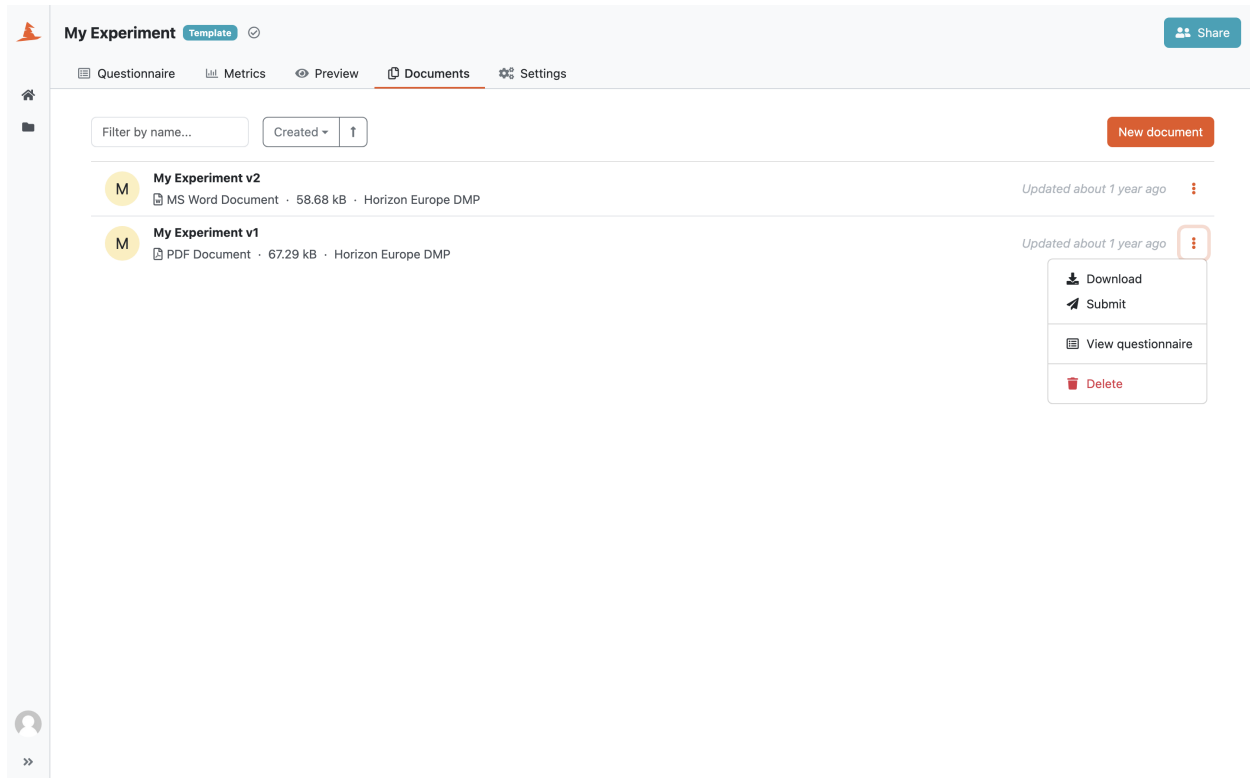
Fig. 57: List of the documents on the project.

Document Submission

Submission services can be used to quickly submit the document directly from DSW to some external service.

If there are any document submission services configured for our DSW instance, we should be able to see *Submit* option when we open document menu in the document list by clicking on the triple dots. Then we choose the desired from the list and click on *Submit*.

We can see all the submissions for each document in the document list as well.

Settings

In the *Settings* tab, we can configure some details about the project. First we have a **name** and a **description** to identify the project.

Next, we have **Project Tags**. These can be used for providing some metadata or categorization of the project. In the project list, we can filter the projects by these tags. We can write any text we want as a project tag and DSW will suggest us the tags that are there already in use so we can keep them consistent.

Note: Project tags might be disabled in some DSW instances.

New Document

Name

My Experiment

Answered (current phase): 8/25
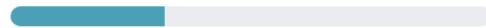
Answered: 20/77
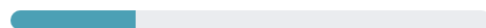
Document Template

H **Horizon Europe DMP** 1.7.0
Data Management Plan according to the Horizon Europe template

Format

○ ⌑ HTML   ◉ ⌑ PDF Document

○ ⌑ MS Word Document

Cancel   Create

Fig. 58: We can choose any compatible document template and format when creating a new document on the project.

Fig. 59: Submission service selection for a document.

Fig. 60: Project settings.

Default Document Template

We can set a **default document template** and a **default document format**. These are then used in the *preview* tab and also pre-selected when creating a *new document*.

Project Template

We can use the project as a *project template*. If we enable this option, researchers can use it when creating a new project *from project template*. Note that we also need to make it visible for other logged-in users in *sharing* settings.

Note: Project template options are visible only for Data Stewards or Admins.

Unsupported metamodel can appear, when the document template is not compatible with the version of DSW. Researchers should contact their Data Steward or Administrator in this case.

Knowledge Model

We can see the *knowledge model* and its tags used for creating the project. If we want to change that, we can simply create a *project migration*.

Danger Zone

In the danger zone, we can delete the project. Once the project is deleted it can **no longer be recovered**.

Sharing

We can share project with other DSW users or even external collaborators. We can access all the sharing option by clicking the *Share* button in the top right.

There are different roles how we can share the project that can access different project features:

Note: Some of the following options might be globally disabled in the application settings for the whole DSW instance, therefore not visible on the project level.

The following video tutorial explains and showcases sharing options and tools that can be used while collaborating with others. Some features mentioned in the video are also explained in the *project questionnaire*.

https://youtu.be/ZN0VTbpLrHk

Fig. 61: Project sharing settings.

Users

We can choose specific users from the DSW instance and their role on the project to grant them access to project features based on the table above. This is a good way to add other collaborators that work together with us on the project. Also, this is the only way to add other project owners.

Visible by all other logged-in users

We can enable the *Visible by other logged-in users* toggle to grant access to the project for all other users without the need to explicitly list them. Then, we can also choose what the users can do – **view**, **comment**, or **edit** the project. We cannot grant **owner** access this way though.

This can be useful, when we want to have an example project accessible by everybody. We can simply enable this and choose that other users can **view** the project. We also want to set this up when we create a *project template*.

Public link

We can enable the *Public link* toggle to grant access to the project to anyone who has the link. We can again choose what they can do – **view**, **comment**, or **edit** the project. Also, the public link is visible there, so we can simply copy it and send to whoever we want to collaborate with. Then, they don't need a DSW account and still be able to access our project.

Project Migration

Every project is based on a specific *knowledge model*, its version, and selected tags. Sometimes, we might want to change the knowledge model to a different version (for example, when a new version is released), change the knowledge model (for example, when a new customization is created), or just change the tag selection. Project migration is a process where we can do this.

The following video explains all aspects of Project Migration.

https://youtu.be/E5bXIWvhNKw

Creating a Project Migration

We can start a project migration either from the *project list*, or from the *project settings*. Sometimes, when there is a newer version of the knowledge model available, we can see a tag *update available* next to the project name. We can click on the tag to start the migration as well.

We can see the **original knowledge model**, its **version**, and selected **question tags** on the left side. On the right side we can choose new values for all of these. After we are satisfied with our selection we can click on *Create* button.

Note that the original project will remain unchanged until the migration is finished. So we can cancel it anytime before it is finsihed without affecting the project.

Fig. 62: Choosing a new knowledge model for the project when creating a project migration.

Project Migration

The next screen is the project migration itself. We can go through all the changes between the orignal and the new knowledge model that affects our answers. During this process, we can also add *todos* if we want to come back to a specific question later, after the migration.

It is possible that there are no changes to review. This can happen when we don't have all the answers in the questionnaire yet and those we have are not affected by the changes, i.e., all of the questions that we answered are in the original and in the new knowledge model.

We can leave the migration at any point now and come back to it later. We will see the project twice in the *project list*, one of them tagged as *migrating*. If we open the migrating one, we can come back to the project migration. If we open the other one, we can access the original project, however, only in the read-only mode until the migration is finished or cancelled.

Fig. 63: Reviewing changes during the project migration.

Cancelling a Project Migration

At any point before we finalize the migration, we can decide that we actually don't want to do the migration. We can simply navigate to the *project list* and choose the *Cancel migration* action on the project tagged as *migrating*. This will cancel the migration and the original project will remain unaffected.

Finishing a Project Migration

After we resolve all the changes (or if there are no changes to review), we can click on *Finalize migration*. This will complete the project migration applying all the knowledge model changes, and unlocking the project from the read-only mode.

Project Templates

Note: Only the data stewards or admins can create project templates.

When creating a new project, we need to choose a knowledge model and optionally select some question tags. After the project is created, we should also choose a *default document template* and format to enable *preview*. It can be overwhelming for new researchers to set up everything when they are new to all this.

Project templates are special type of projects where we can set up everything – choose a knowledge model and question tags, set up default document template, pre-fill some answers, add TODOs, comments or editor notes. Researchers can then pick from these project templates when *creating a new project*. The new project will be the exact copy of the

project template so they don't have to set those things themselves and they have an easier start to their data management planning.

When we want to turn a project into a project template we need to go to the *project settings* and check the *Project Template* checkbox.

Sharing the Project Templates

Project templates follow the same rules for sharing as regular projects. Therefore, to make it available for other users, we need to set up proper *sharing settings*.

We can either share them with specific users only, or we can simply enable that the project template is **visible by all other logged-in users** and that they can only **view** it (as we do not want them to change it).

External Resources

- Project Templates in FAIR Wizard

1.4.2 Project Importers

Warning: Project importers are an experimental feature.

We can use project importers to import data from different DSW instances or even different applications to DSW. Each has a set of supported knowledge models defined. This is because each knowledge model has a different structure and the importer needs to understand it so it can import the answers to the correct questions.

Note: Only data stewards or admins can access project importers.

If we navigate to *Projects → Importers*, we can see the list of all available importers. We can enable or disable them by clicking on the triple dots icon and choosing the appropriate action.

Fig. 64: List of project importers where we can enable or disable them.

More information about how to develop project importers is available on the *project importers development* page.

1.5 Documents

As admins, we can quickly browse all documents stored in the DSW instance by navigating to *Documents* from the main menu. It is possible to search for a document by name or sort them using the name or creation timestamp.

Each document has name, format, and certain size (if the generation is finished). It can be directly downloaded or deleted from the list. Moreover, we can quickly navigate to the project from which the document is created.

In case that there is a document that was not generated due to an error, we can check the error message. Also, if there are some documents got stuck in *Queued* or *In Progress* status, we should check the deployment (especially of the document worker component).
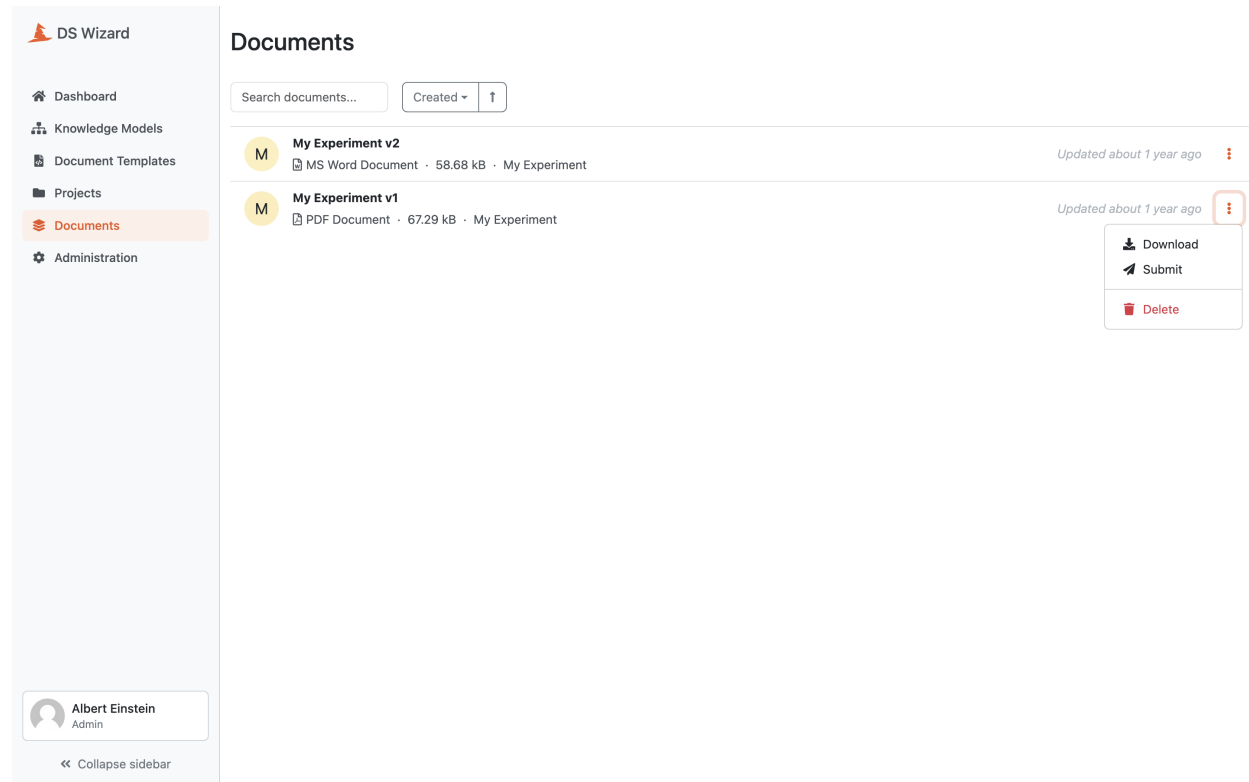
Fig. 65: List of all documents across projects.

1.6 Administration

Administration section serves as the name suggests to admins with managing the DSW instance. By navigating via *Administration* item from the main menu, we can manage the things listed below.

1.6.1 Settings

This section covers different settings available to admin users. The settings are categorized as listed below.

Note: Do not forget press *Save* button to save the changes.

System Settings

System settings allow us to configure basics about the organization running the DSW instance, how can user register and login, and finally the privacy and support information for the users.

Organization Settings

On this page, we can configure **Name**, **Description**, and **Organization ID** for our DSW instance. The organization ID can contain alphanumeric characters and dot symbol (but cannot start or end with dot).

Note: It is recommended to use period as for domain name or to use it for capturing organizational structure in the ID, for example, `faculty.university` or `group.faculty.university`.

Another part of settings on this page are **Affiliations**. It servers to pre-define a list of possible affiliations (on per line) that will be suggested to users while they register or update their profiles.

Authentication Settings

The **Default Role** settings option allows us to define which role is assigned to new users (see *user roles* for details about permissions).

> **Warning:** It is recommended to set this to the lowest role possible, i.e. **Researchers**. Otherwise, new users will be able to change the content for all other users in the DSW instance.

Internal

For internal authentication, we can set whether the **Registration** is enabled or not. If enabled, any user who can visit the DSW instance may sign up (and obtain the default role).

Note: In case we are using OpenID or creating user accounts manually, registrations should be disabled.

Another option is whether the **Two-Factor Authentication** (2FA) is enabled. If enabled, once users try to log in using credentials, they receive an email message with one-time code to confirm the login. Moreover, we can configure **Code Length** (how many character the code has) and **Expiration** period in seconds.

External

Using these settings we can add **OpenID Services** to allow logging into the DSW instance via external identity provider. First, press *Add* and fill **ID** of the service (use only lowercase alphanumeric characters or dash symbols). Then, we should prepare the client application on the side of OpenID service:

- Use **Callback URL** (and optionally **Logout URL**) to create the client

- Obtain **Client ID** and **Client Secret**

- Obtain OpenID endpoint **URL** (we may get one ending with `/.well-known/openid-configuration`, if so we just use the part before this suffix)

- Configure the client to have the following claims: `openid`, `profile`, `email`

- Configure the client to provide the following details in ID tokens: `email`, `given_name`, `family_name`

Back in the DSW settings, we can fill **Client ID**, **Client Secret**, and **URL** from our OpenID client together with optional **Parameters** (usually not needed). Finally, we can configure how the log-in button will look like by setting **Icon** (by using Font Awesome), **Name**, **Background**, and text/icon **Color**.

Note: After setting a new OpenID service, we should directly test it and verify that the configuration works well. For that, we can simply open our DSW instance in a new anonymous window of the web browser.

Privacy & Support Settings

To request users to agree with **Privacy Policy** and/or **Terms of Service** documents, we can configure URLs to locate such documents. Then, when new users register to the service, they will be prompted to agree with the linked documents. Note that usually you should inform already registered users in case you change such documents.

Support

These settings also allow us to configure **Support Email** that users can use to request help or report issues. Similarly **Support Site Name** and **Support Site URL** can be used in case we want users to create tickets in issue tracker of some repository, e.g., on GitHub. These support links are then shown in *Report issue* modal window.

User Interface Settings

User Interface (UI) settings allow us to manage how the DSW instance looks like: styling, titles, or the dashboard shown when user logs information.

External

OpenID Services

ID   🗑 Remove

institutional-idp

Callback URL

https://mediakit.ds-wizard.org/wizard/auth/institutional-idp/callback

Logout URL

https://mediakit.ds-wizard.org/wizard-api/auth/institutional-idp/logout

Client ID   **Client Secret**

4bd87155-42b9-4a7a-b48b-cf2880b0b763   ·············· 👁

URL

https://idp.example.com/4bd87155-42b9-4a7a-b48b-cf2880b0b763/v2.0

Parameters

➕ Add parameter

Icon   **Name**   **Button Preview**

fas fa-key   Example SSO   🔑 Example SSO

Background Color   **Text Color**

#3f58ab   #ffffff

➕ Add service

Fig. 66: Example configuration of OpenID service.

Dashboard & Login Screen Settings

The dashboard settings allows us to adjust what users will see after they log in, i.e. on the application initial page called the dashboard.

Dashboard Style

We can select the **Dashboard Style** whether the user should see a standard **welcome** screen which just greets the user in the application, or a **role-based** dashboard which contains widgets based on current user's role (see *User Roles*):

- **Researcher**
  - **Recent Projects Widget** contains a list of recent projects of the user for a quick navigation.
  - **Create Project Widget** lets the user quickly start a new project.
- **Data Steward**
  - **Create KM / Project Template Widgets** let the user to quickly start a new knowledge model editor or project template.
  - **Outdated KM / Document Templates Widgets** allow to quickly see outdated packages and document templates in case the DSW Registry connection is configured.
  - **Import KM / Document Template Widgets** allow to proceed easily to import of a knowledge model or a document template in case the DSW Registry connection is configured.
- **Administrator**
  - **Outdated KM / Document Templates Widgets** allow to quickly see outdated packages and document templates in case the DSW Registry connection is configured.
  - **Usage Widget** summarises the usage just as is also possible to see in the *Usage*.
  - **Configure Organization Widget** quickly navigates to *Organization Settings* if it is not yet done.
  - **Configure Look and Feel Widget** quickly navigates to *Look & Feel Settings* to adjust style of the DSW instance.
  - **Connect DSW Registry Widget** quickly navigates to *DSW Registry Settings* to configure the connection (if not yet been done).
  - **Add OpenID Widget** quickly navigates to *Authentication Settings* to configure the identity provider services (if not yet been done).

Login Info

It is possible to write a message that users will see before logging in the DSW instance, using HTML or Markdown. The Login info is placed in the center of the login screen. We can use it to explain users in what cases they can/should use our DSW instance, how they should log in (e.g. if we have more *authentication services* configured), or if there is any news regarding our DSW instance.

> **Warning:** Defining HTML classes in the login info can overwrite DSW application classes. It is recommended to use prefixes for classes, if they are used, to avoid conflicts.

Sidebar Login Info

It is also possible to write another message that users will see on the login screen. The Sidebar Login info is placed underneath the login form. We can also use HTML or Markdown as in the Login Info.

Announcements

Another option to adjust the dashboard and/or the login screen is to add Announcements. Announcements are displayed above the main content in the login screen. In dashboard, they are also displayed above the main content for both **welcome** and **role-based** dashboard style. There are three levels of Announcements:

- **Info** - light blue color for sending informations to the users.
- **Warning** - yellow to warn the users about something.
- **Critical** - red to signalize the Announcement is critical and it needs attention.

The content of the Announcement can be edited using Markdown. There are also two additional switches which determine, where the Announcement is displayed. The Announcement can be set up to be displayed either on the dashboard after users log in or on the login screen before the users log in. It is also possible to display the same Announcement in both places. Number of Announcements is not limited.

Look & Feel Settings

This part of settings allows us to adjust how the DSW instance looks like.

Application Titles

There are two titles that we can set. First, **Application Title** is the full title that should identify the DSW instance, for example, in browser's tab. Second is **Short Application Title** which is visibile, for example, at the top of the main (left) menu next to the icon. We should keep **Short Application Title** really short (about 10 characters at maximum) so it fits well.

Custom Menu Links

We can easily add custom links to the main (left) menu by clicking *Add* under **Custom Menu Links**. For each link, we can set **Icon** (from Font Awesome), **Title** and the target **URL**. We can also set whether the link should open in **New Window** (if not, it will nagivate user directly in the same window/tab from DSW instance). Once the links are there, we can manage them or delete them at this place.

Content Settings

This part of settings allows us to configure various content-related things such as the connection with DSW Registry for easy imports, Knowledge Models, Projects, and Document Submissions as listed below.

Look & Feel

Application Title

DS Wizard

Full name of the DSW instance (displayed, for example, in the browser tab title or before login).

Short Application Title

DS Wizard

Short name of the DSW instance (displayed, for example, on top of the navigation bar). Short title can be the same as the application title if it is short enough.
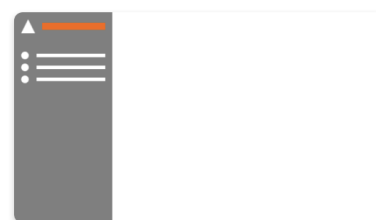
Custom Menu Links

Configure additional links in the menu. Choose any free icon from the Font Awesome, e.g. *fas fa-magic*. Check *New window* if you want to open the link in a new window.

Icon	Title	URL	New window	
fas fa-book	User Guide	https://guide.ds-wizard.org		
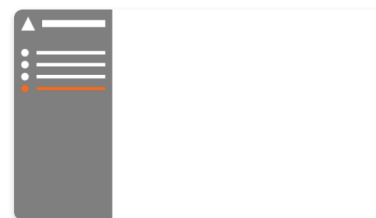
+ Add link

Fig. 67: Example configuration of a custom menu link.

DSW Registry Settings

In this settings, we can configure a connection to DSW Registry that will allow import of various content (Knowledge Models, Document Templates, and Locales) to our DSW instance.

Upon enabling the DSW Registry option, we are prompted to enter a **Token**. It can be obtained either by direct registration in the *DSW Registry<https://registry.ds-wizard.org>* or through clicking *Sign up* button. After clicking the button, we will need to enter details about organization (it is prefilled from *Organization Settings*) and email address to which the confirmation will be sent (prefilled by email of the current user). Then, after clicking a link in the confirmation email, the token will be prefilled automatically. After having the token filled in either way, we can *Save* the settings.

After successfully setting the DSW Registry, we will see option to import from it for *Knowledge Models*, *Document Templates*, and *Locales*.

Note: The **Token** value is encrypted in the database.

Knowledge Models Settings

Public Knowledge Models

If we want to let users to see and browse certain Knowledge Models (specifically, visit the *KM detail* and the *KM preview*) even if not logged in, we can enable **Public Knowledge Models**. Then, we need to specify **Allowed Packages**, e.g. which ranges of verions of a certain knowledge model will be publicly available. A blank value serves as *any value*, for example, if we fill the **Organization ID** and **Knowledge Model ID** but leave **Min Version** and **Max Version**, it will result in all version of that knowledge model to be public.

Integration Config

The integrations specified in Knowledge Models can use configuration values (typicaly secrets such as API keys or tokens) from YAML configuration specified in *integration.yml file* or the content specified here under **Integration Config**. The value here can be for example:

```
dbase:
    apiKey: topSecretDBaseApiKey
    apiUrl: https://api.dbase.example:10666
```

Note: This configuration value is encrypted in the database.

Projects Settings

Project Visibility

If we want to let users select visibility of their projects within the DSW instance, we can enable **Project Visibility** feature. If it is disabled, the new projects will have the **Default Project Visibility** which is used when creating a new project:

- **Private** = the project is visible only to the users with explicit access to the project.

- **Visible - View** = the project is visible in view-only mode to all logged-in users, i.e. all users will be able to see the project in their *projects list* and access it (but not edit or comment anything unless they are invited with different permissions).

- **Visible - Comment** = the project is visible in comment mode to all logged-in users, i.e. all users will be able to see the project in their *projects list*, access it and comment it (but not edit anything unless they are invited with different permissions).

- **Visible - Edit** = the project is visible in edit mode to all logged-in users, i.e. all users will be able to see the project in their *projects list*, access it, comment it, and also edit it (e.g. answer questions or editor notes).

Project Sharing

If we want to let users select sharing option of their projects within the DSW instance, we can enable **Project Sharing** feature. If it is disabled, the new projects will have the **Default Project Sharing** which is used when creating a new project:

- **Restricted** = only logged-in users can access the project depending on the project visibility (no public access for anonymous users).

- **View with the link** = anyone with the link to the project may open it in view mode and browse it.

- **Comment with the link** = anyone with the link to the project may open it in comment mode, i.e. browse it and comment on questions.

- **Edit with the link** = anyone with the link to the project may open it in edit mode, i.e. browse it, comment on questions, and also edit it (e.g. answer questions or editor notes).

Anonymous Projects

If we have enabled *Public Knowledge Models*, we can also allow anonymous users to create projects where they will be able to fill questionnaires by enabling **Anonymous Projects**. These anonymous project then work as any other projects with public link set to edit permissions. However, if a logged-in user accesses such a project, then such a user may claim the ownership by clicking *Add to my projects* button. Anonymous users cannot create new documents, for that they must register and open the project as a logged-in user.

Feedback

In case we want to allow users to provide feedback specific to questions directly from *questionnaires*, we can enable **Feedback** and configure a GitHub repository which will be used as an issue tracker. We should have a bot/service account created with access to the GitHub repository and obtain Personal Access Token. This account will be used to create the GitHub issues in the repository. Then, we need to simply fill **GitHub Repository Owner**, **GitHub Repository Name**, and the **Access Token**.

Note: The **Access Token** value is encrypted in the database.

Project Tagging

If enabled, users will be able to tag their projects (using so-called **Project Tags**) and then use those tags to filter the *projects list*. The users will be always able to write their own tags but we can provide a list of pre-defined **Default Project Tags** (one per line).

Document Submission Settings

If we enable the **Document Submission** feature, users will be able to see *Submit* option for their documents. After selecting it, they will be prompted to select a service compatible with the document where they want to submit it.

Each service must have its own **ID** (recommended is to use lowecase alphanumeric symbols and dash symbols). Then, we can set human-readable **Name** and **Description** (Markdown-enabled) to clarify for users what is this service and in what cases they should use it. We also need to specify the **Supported Formats**, for each we need to select a document template, its version, and the desired format. Finally, we configure how the document is sent to the external service, the request may contain some **User Properties** (users will be able to set values for them in their user profiles) and it is a HTTP request with a specific **Method**, sent to the **URL**, possibly with HTTP **Headers**. The very last option is to check whether the file should be sent as **Multipart** (with its own **File Name**) instead of plainly in the request body. Most of this configuration should be specified by the external submission service.

Note: In case we will update the document template used in **Supported Formats**, we should verify that it is still suitable for the submission service and if yes, then add it as a new entry under **Supported Formats**.

Info

The info section of settings simply serves to let us check various information and statistics about the DSW instance.

Usage

Usage allows us quickly see numbers of entities in the DSW instance such as number of users, active users, knowledge models, KM editors, templates, projects, or documents. Moverover, we can see storage usage, i.e. how much capacity is being used by documents and templates (their files). We cannot perform any actions on this page.

1.6.2 Users

Users list allows administrators to see and manage all users in a DSW instance. The list can be filtered using role, searched based using name or email fragment, and sorted via various properties of users. The list shows the role of a user next to its name and also indicates in case the user is inactive. Next to the email, we can quickly see what authentication services the user uses to log-in (*internal* is the internal authentication with email-password credentials, other are based on configured *OpenID services*).

A *user detail* can be opened by clicking the name of a user or by selecting *Edit* in the right dropdown menu for the desired row. There, a user can be also deleted via the *Delete* action. Finally, administrator can *create a new user* by clicking *Create*.
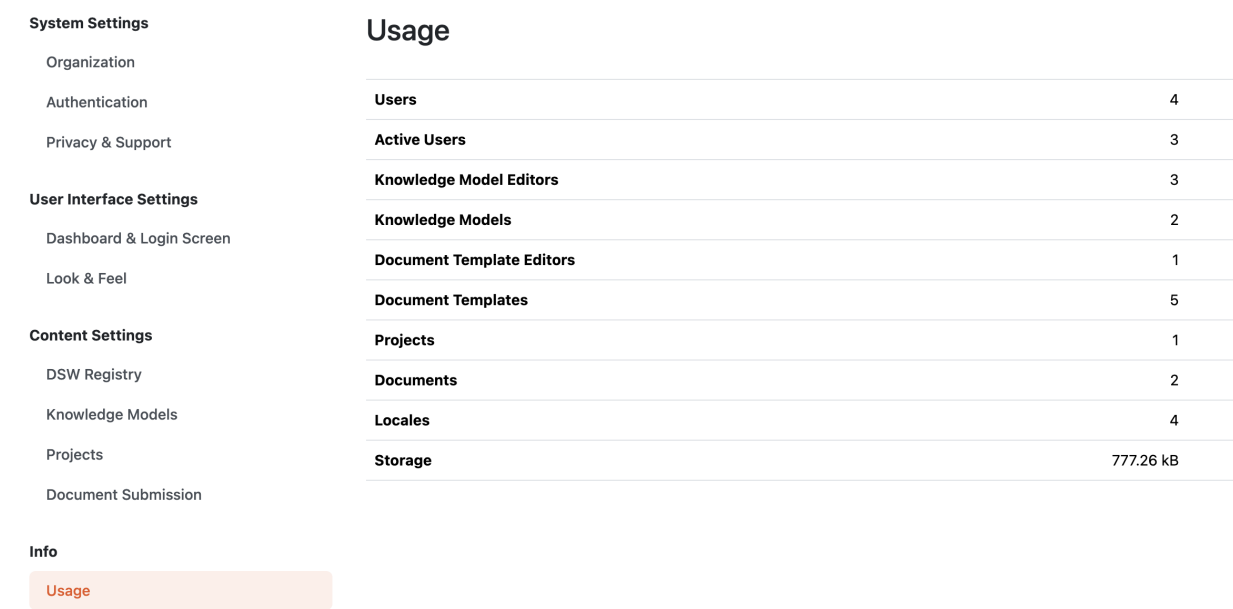
Fig. 68: Usage statistics.
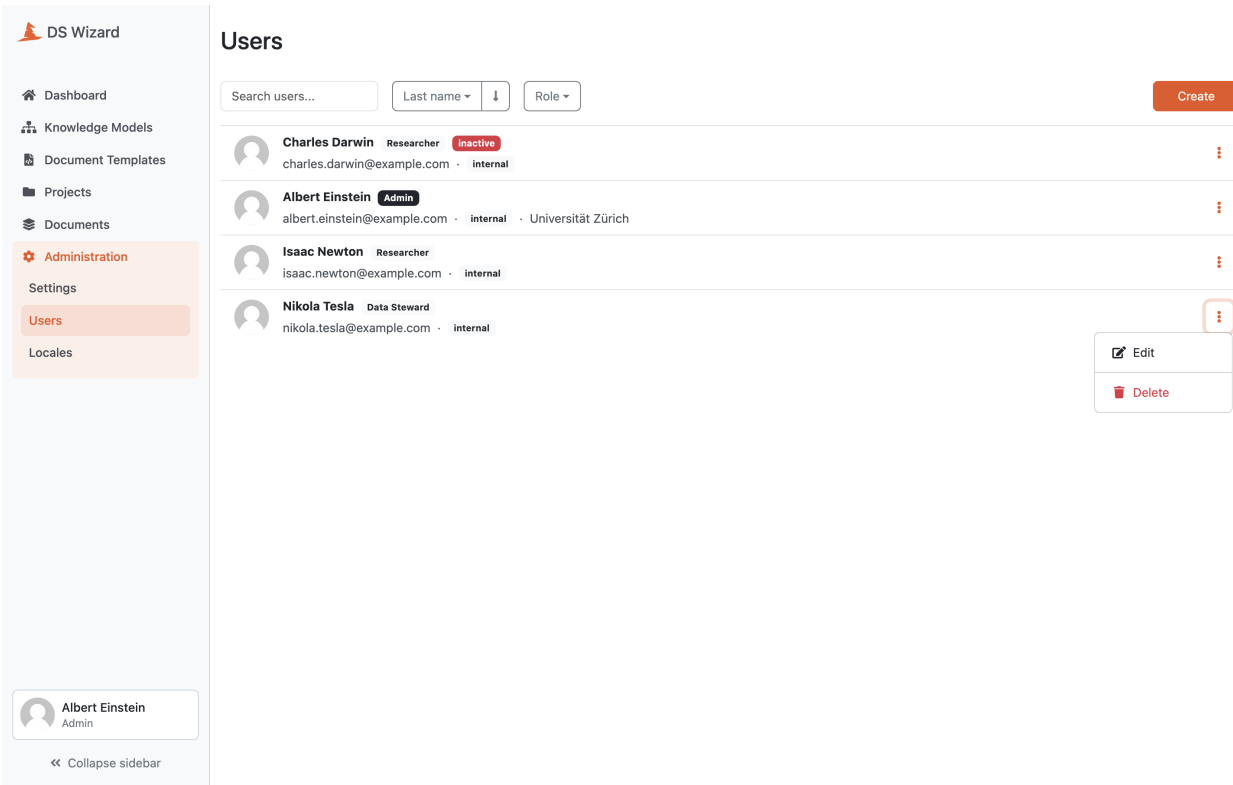
Fig. 69: List of users.

Create User

As administrators, we can create new users manually by clicking *Create* on the *users list* and submitting the form. Each user must have a unique email address, first name and last name, assigned *role*, and a password. Optionally, a user can have affiliation specified.

Note: If the user is created by administrator, the user is activated by default and no email is sent to the user.

User Detail

As administrators, we can edit existing users manually on the detail (selected user from the *users list*). It is possible to change all properties of the user, including possibility to change whether the user account is active or inactive.

User Settings

Profile	
Password	
API Keys	
Active Sessions	
Submission Settings	

Profile

Email
albert.einstein@example.com
internal

First name
Albert

Last name
Einstein

Affiliation
Universität Zürich

Role
Admin

Active

Save

User Image

Image is taken from OpenID profile or Gravatar.

Fig. 70: Detail of a user profile.

The password can be also changed (after selecting *Password* from the left navigation of user settings).

User Roles

There are three user roles available: researchers, data steward, and admin. Permissions are associated with the roles, basically they affect what the users can do in a DSW instance:

User Settings

Profile

Password

API Keys

Active Sessions

Submission Settings

Password

New password

············

New password again

············

Save

Fig. 71: Form for changing password of a user.

1.6.3 Locales

After navigating to *Locales* (under *Administration*), we can browse and manage a list of locales in the DSW instance. Similarly to knowledge models and document templates, each locale has its unique identifier and version. Moreover, each locale has a language code specified. In the list we see the latest version and can quickly navigate to *Locale Detail* (which includes also older versions) by clicking the locale name or selecting *View detail* from the right item menu of the desired row.

There is always the **English** locale (`wizard:default:1.0.0`) which is embedded and cannot be deleted. For others, we can use *Export* and *Delete* options from the right item menu.

Another option is to switch other locale to be the default one using *Set default* action. The default locale will be used in case no available locale tgat matching user's preferences (explicit or implicit from the web browser). We can *Disable* or *Enable* locales except the default one (which must be enabled).

If there is a locale with newer version available in the DSW Registry (and if configured), *update available* clickable badge may appear. Finally, we can use *Import* to *Import Locale* and *Create* to *Create Locale*.

Note: We support community of DSW translators by managing the repository ds-wizard/wizard-client-locales and service for translating using web browser localize.ds-wizard.org.

Import Locale

We can import an existing locale by navigating to *Settings → Locales* in the main menu and then clicking on *Import* button on the list of locales.
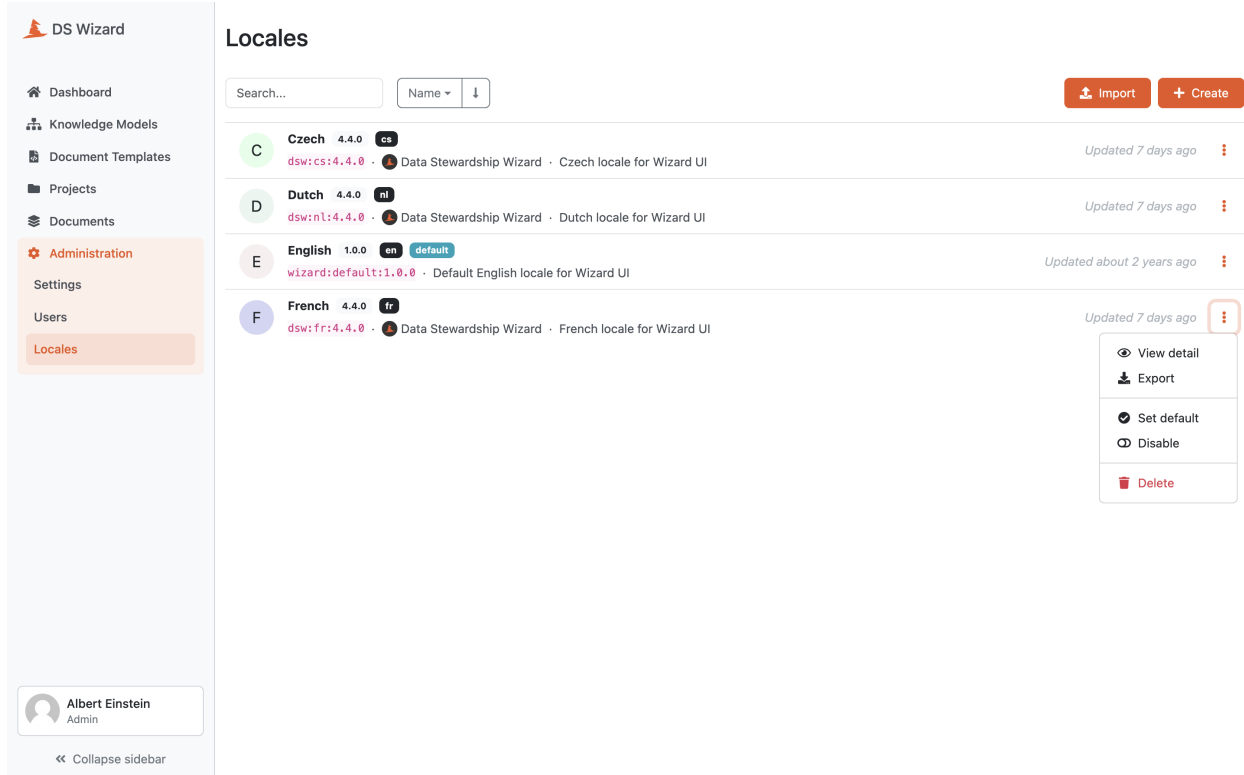
Fig. 72: List of locales.

From DSW Registry

If the DSW instance is connected to the [DSW Registry](#), it is possible to import locales from it by entering the **locale ID** of desired template (e.g. `dsw:cs:0.2.0`) and pressing the *Import* button.

Note: In case of locales present in the [DSW Registry](#), we will be notified about the available upgrades.

From file

We can import a locale as a ZIP package. Such a package can be created as an export from DSW.

Create Locale

We can create a new locale directly in DSW by pressing *Create* from *[Locales](#)*. We need to fill the details about the new locale such as name, description, language code ([RFC5646](#), e.g. `en` or `en-GB`), locale ID, locale version, license, README (with Markdown syntax), and recommended app version. The recommended app version captures for which version of the DSW is this locale intended and compatible with (it can be used in other versions as well but may have some untranslated texts).

Finally, a PO file is requested from us. We can create such PO file in a standard ([gettext](#)-based) way. The needed POT file is always part of the release attachments of the [wizard client application](#) (select the desired release and there is `wizard.pot` asset).

Import Locale

From registry   From file

dsw:nl:4.4.0   Import

You can find locales in DSW Registry.

Fig. 73: Input for importing a locale from DSW Registry.

Import Locale

From registry   From file

Choose a file

Or just drop it here

Fig. 74: Input for importing a locale using a ZIP package.

Locale Detail

The detail of a locale provides us information about a locale after navigating to it from *Locales*. The detail shows basic information about the locale such as its name, ID, language code (RFC5646, e.g. `en` or `en-GB`), version, recommended compatible DSW version, license, and indication whether the locale is enabled or not.

The main part of the detail is the README of the locale that should contain basic information and changelog. In the right panel under the basic information, we can navigate to other versions of the locale.

In the top bar, we can *Export* the locale as a ZIP package, set or unset it as default, *Enable* or *Disable* it, or *Delete* this version of the locale.

Note: The default locale must be always enabled as it serves for users that did not have any preference or do not request matching locale directly using the browser configuration.

Fig. 75: Detail of a locale.

1.7 Profile

As every logged-in user, we can manage our own profile. We can see the options by hovering over the box with our name and avatar in the lower part of the left sidebar with main menu (only avatar is shown in case of collapsed sidebar). From there, we can navigate to *Edit Profile*. If locales are configured, we can also *Change Language*.

Moreover, there are also options to *Log out*, see basic information about the DSW instance using *About*, or in case of problems we can use *Report issue* to know how to proceed.
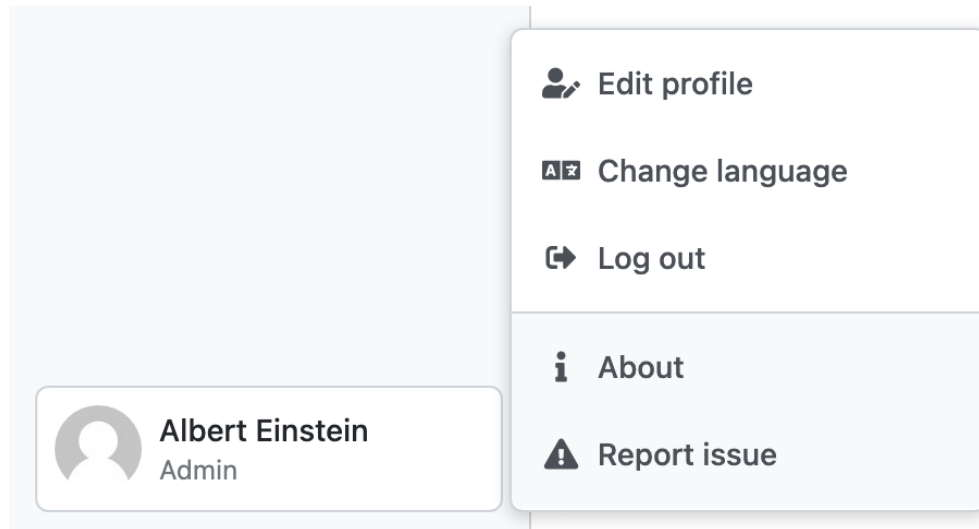
Fig. 76: Profile menu.

1.7.1 Edit Profile

After navigating to *Edit profile* from the *Profile* menu, we can change the information of our profile: **Email** address, **First name**, **Last name**, and **Affiliation**. We need to submit the changes by clicking *Save* button.

In case of configured submission services, there might be additional inputs under **Submission Settings** such as API tokens or other information used for the document submission.

Fig. 77: Form for editting profile with example submission settings.

Note: The values of **Submission Settings** are treated as potentially sensitive information; thus are stored encrypted.

If we want to *Change Password*, we need to switch to *Password* from the left menu titled **User settings**.

Change Password

The password be can changed after navigating using *Password* from the *Edit Profile*. Here we can simply enter new password (it must be strong enough), repeat it again and press *Save* button.

Fig. 78: Form for changing password.

API Keys

When we want to access the DSW through DSW API we have to set up an API Key. The API Key has an *API Key Name* so we can remember for what purpose this Key is used and *Expiration* which is a date from when the API Key will no longer be valid.

Fig. 79: Form for creating an API Key.

After filling out the *API Key Name* and *Expiration*, an **API Key** is generated. In this step we have to copy the API Key, because after seeing it once, it is no longer possible to access it again.

After we click on Done button, the new API Key is hidden and the information about this key is added to the table below, that contains all Active API Keys.

Active Sessions

Here we can see list of active sessions that are logged into our account. In case we don't longer want any of them to have access to our instance, we can revoke the session.
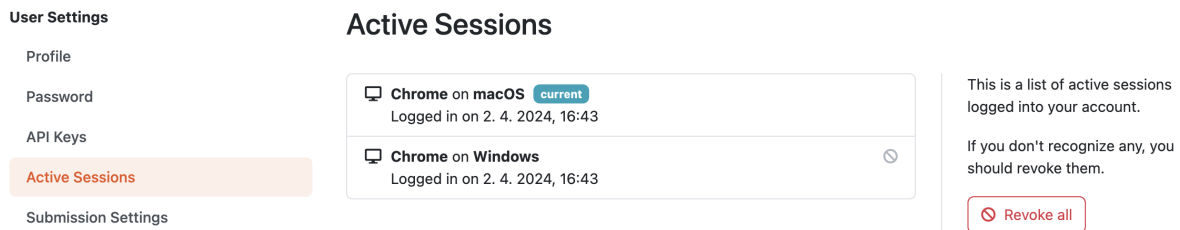
Fig. 80: List of active sessions.

1.7.2 Change Language

A user can explicitly select a desired language after clicking *Change language* from the *Profile* menu. In case the language becomes unavailable later after the selection, it will fall back to the one marked as **default**.
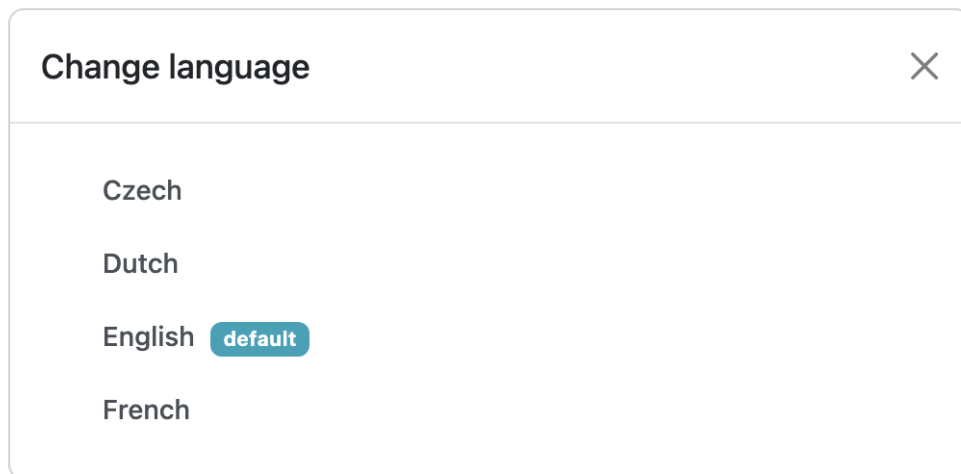
Fig. 81: Modal window with language selection.

Note: The selection of language is saved only locally in the browser (in local storage), so if we log in from different locations, we need to select the language there again.

1.8 Self-Hosted DSW

This section explains how to deploy, configure, and maintain DSW instance on your own.

Note: Before going for self-hosted options, it is a good idea to try out DSW using existing providers.

1.8.1 Deployment

Own DSW Instance

> **Warning:** For production use, we should consult deployment and other operations with the sysadmin or contact the DSW Team for professional services.
>
> The following instructions are intended **only for local deployment** and trying it out. It does not include any safety measures that are required for production deployment and are handled differently based on specific requirements and deployment sites (e.g., using HTTPS, enabled websockets, backups, metrics-gathering, and many others).

Deployment with Docker

The simplest way is to use Docker Compose. Requirements are just to have Docker installed, privileges for current users, and the Docker daemon started. At least basic knowledge of Docker is required.

1. Clone the DSW Deployment Example repository
2. Check config files (described in *Configuration*)
3. Run the DSW with Docker compose `docker-compose up -d`
4. After starting up, we will be able to open the Wizard in our browser on http://localhost:8080
5. We can use `docker-compose logs` to see the logs and `docker-compose down` to stop all the services

> **Warning:** It is important to use latest patch version, we should monitor the documentation or join the community Slack if new hotfix releases are available.

Deployment without Docker

It is highly recommended using Docker. However, we can compile and run everything directly on our machine. Nevertheless, an additional expertise in programming and building/running Elm, Haskell, Python, and others is required.

The related code and instructions are available:

- https://github.com/ds-wizard/engine-backend
- https://github.com/ds-wizard/engine-frontend
- https://github.com/ds-wizard/engine-tools

Default Users

Initially, migrations will fill the database with predefined data needed including three users, all with password - `password`:

- albert.einstein@example.com (Administrator)
- nikola.tesla@example.com (Data Steward)
- isaac.newton@example.com (Researcher)

We can use those accounts for testing or initially make our own admin account and then delete them.

> **Warning:** Having a public instance with default accounts is a **security risk**. We should delete or change default accounts (mainly Albert Einstein) if our DSW instance is public as soon as possible.

Note: Do not use UIDs below 10000 as that might introduce a security risk.

DSW Registry

When we have our own self-hosted instance, it is essential to register within the DSW Registry. It is a source of shared knowledge models, document templates, and locales that can support our deployment. The registry is also integrated inside the DSW. Therefore, we can easily pull new versions from the DSW. The registration can be done either directly in our DSW instance in Settings or via the DSW Registry website.

Initial Knowledge Model, Document Templates, and Locales

When we have a fresh installation, there are just the default users and no knowledge models or document templates. We are free to create them from scratch if we want. Another option is to import existing KM and document templates from the DSW Registry.

In the DSW Registry, we can find the core knowledge model for general data stewardship and a couple of document templates, such as Horizon Europe and locales! Another option is to import it from a file if we have any (according to usage).

Database Backups

If we want to regularly backup our database (and we should!), all we need to do is to set up a cronjob that backups PostgreSQL database (e.g., using pg_dump utility) as well S3 storage.

Deployment Requirements

The following requirements were estimated using limiting Docker resources provided to containers.

Component	Minimal	Recommended
Server	128 MB	512 MB
Client	16 MB	64 MB
Document Worker	240 MB	1024 MB
Mailer	240 MB	448 MB
Total	**624 MB**	**2048 MB**

As for the CPU, there are no long-running tasks that would require excessive CPU consumption. Limiting CPU resources can only make some operations slightly longer (e.g., importing a knowledge model or generating a document). A number of CPUs/cores will then affect performance for concurrent users/actions. Memory consumption is affected by the size of the content (as some content is being cached for speed optimizations).

Memory used by document workers might be affected by the size of the document template (and assets) for generating a document. Recommended memory in the table above is approximated for a long-term run (without restarts) with a significant amount of content. It also minimizes the need for garbage collection techniques that may slow down the server component.

Note: Real requirements should be aligned with the intended use (number of concurrent users, number of users in total, size of document templates, etc.). The minimal requirements are sufficient for single-user deployment and recommended requirements should handle tens of concurrent users.

1.8.2 Configuration

Settings

Most of the configuration is done through *Settings* (accessible by Administrator).

Configuration Files

Configuration files are used for setting the server-side configuration. Since the 3.0 release, the configuration for server and document worker components has overlapped. Therefore, we can have a single configuration file for both.

Server Configuration

For reference, see the configuration of the DSW Deployment example.

General

This configuration section is used only by **Server** and covers basic configuration of the application.

serverPort
> **Type**
>> Int
>
> **Default**
>> 3000
>
>> Port that will be the web server listening on.

clientUrl
> **Type**
>> URI
>
>> Address of client application (e.g. https://localhost:8080).

secret
> **Type**
>> String
>
>> Secret string of 32 characters for encrypting configuration in the database.

rsaPrivateKey
> **Type**
>> String
>
> RSA private key for signing JWT tokens according to RS256 algorithm in PEM format (e.g. use `ssh-keygen -t rsa -b 4096 -m PEM -f jwtRS256.key` without passphrase and paste the contents to this configuration item).

Warning: We should keep our `secret` and `rsaPrivateKey` secured! Changing `secret` will require re-configuration of secrets stored in the database, e.g., token for Registry.

If we need to change our `secret`, we need also replace all values encrypted by the secret that is stored in the database as follows:

1. Note somewhere values from Settings: Client ID and Client Secret of OpenID configurations, Registry token, and GitHub token for Feedback functionality, etc. Adjust the settings that the values are not there (recommended; e.g., remove OpenID configuration), and save it.

2. Change the `secret` in the configuration file and restart the DSW server (re-create the container if using Docker).

3. Adjust the settings back to our previous values.

4. If we also use some "user properties" (for the Document Submission feature), let our users know to change the values in their profiles.

Database

Information for connection to PostgreSQL database.

database.connectionString
> **Type**
>> String
>
>> PostgreSQL database connection string (typically: `postgresql://{username}:{password}@{hostname}:{port}/{dbname}`, for example, `postgresql://postgres:postgres@localhost:5432/postgres`).

S3

Information for connection to S3 storage (used for document and document template assets).

`s3.url`
> **Type**
>> URI
>
>> Endpoint of S3 storage, e.g., `http://minio:9000`

`s3.username` Noindex
> **Type**
>> String
>
> Username (or `Access Key ID`) for authentication

`s3.password`
> **Type**
>> String
>
>> Password (or `Secret Access Key`) for authentication

`s3.bucket`
> **Type**
>> String
>
> **Default**
>> `engine-wizard`
>
>> Bucket name used by DSW

Warning: S3 service must be publicly accessible (so users can download documents and export templates or locales). Also, bucket must be created otherwise documents cannot be created and documet tempates / locales imported.

Mail

This configuration section is used only by **Mailer**. It must be filled with SMTP connection information to allow sending emails (registration verification, password recovery, project invitation, etc.).

`mail.enabled`
> **Type**
>> String
>
>> It should be set to `true` unless used for local testing only.

`mail.name`
> **Type**
>> String
>
>> Name of the DSW instance that will be used as "senders name" in email headers.

`mail.email`
> **Type**
>> String
>
>> Email address from which the emails will be sent.

`mail.host`
> **Type**
>> String
>
>> Hostname or IP address of SMTP server.

`mail.port` **Type**

Int

Port that is used for SMTP on the server (usually `25` for plain or `465` for SSL).

`mail.ssl` **Type**

Boolean

Default

`false`

If SMTP connection is encrypted via SSL (we highly recommend this).

`mail.authEnabled` **Type**

Boolean

If authentication using username and password should be used for SMTP.

`mail.username` **Type**

String

Username for the SMTP connection.

`mail.password` **Type**

String

Password for the SMTP connection.

Externals

This configuration section is used only by **Document Worker**. We can affect steps for templates that use external tools (`pandoc`). It is usually sufficient to keep the defaults. Each of them has configuration options:

`executable` **Type**

String

Command or path to run the external tool.

`args` **Type**

String

Command line arguments used to run the tool.

`timeout` **Type**

Int

Optional for limiting time given to run the tool.

Integrations Configuration

Integrations in the DSW use external APIs. Sometimes, we might need some configured variables, such as API keys or endpoints. For example, integration with ID `dbase` might use the following configuration.

```
dbase:
    apiKey: topSecretDBaseApiKey
    apiUrl: https://api.dbase.example:10666
    someConfig: someValue4Integration
```

There can be multiple integrations configured in a single file. These can be used then when setting up the integration in the Editor as ${apiKey}, ${apiUrl}, etc. More about integrations can be found in separate *integration questions documentation*.

Note: Different knowledge models may use different variable naming. Please read the information in README to find out what is required. We recommend authors to stick with `apiKey` and `apiUrl` variables as our convention.

Client Configuration

If we are running the client app using "With Docker", the all we need is to specify `API_URL` environment variable inside `docker-compose.yml`. In case we want to run the client locally, we need to create a `config.js` file in the project root:

```
window.dsw = {
    apiUrl: 'http://localhost:3000'
}
```

Custom Logo

We can use our own custom logo by mounting it to the client container. The logo must be square and in SVG format.

```
dsw-client:
    volumes:
    - /path/to/logo.svg:/usr/share/nginx/html/wizard/img/logo.svg
```

Favicon

If we changed the logo, we might also want to change the favicon. First, we need to generate the necessary files using, for example, this Favicon Generator. The wizard uses the following files:

- android-chrome-192x192.png
- android-chrome-512x512.png
- apple-touch-icon.png
- browserconfig.xml
- favicon-16x16.png
- favicon-32x32.png
- favicon.ico
- mstile-144x144.png
- mstile-150x150.png
- mstile-310x150.png
- mstile-310x310.png
- mstile-70x70.png
- safari-pinned-tab.svg

- site.webmanifest

They are all in the `/usr/share/nginx/html/wizard/img/favicon` folder, so we can mount our generated favicon files from the generator there, or we can mount the whole folder:

```
dsw-client:
    volumes:
    - /path/to/favicon:/usr/share/nginx/html/wizard/img/favicon
```

Style Customizations

We can mount a file called *head-extra.html* to the wizard client image to attach extra code to the `<head>` tag. This can be used to override some styles or CSS variables. For example, to change a color theme, we only need to override a few Bootstrap variables:

```html
<style>
    :root {
        --bs-bg-primary-color: rgb(255, 255, 255);
        --bs-btn-primary-active-bg: rgb(18, 128, 106);
        --bs-btn-primary-color: rgb(255, 255, 255);
        --bs-btn-primary-active-color: rgb(255, 255, 255);
        --bs-btn-primary-disabled-color: rgb(255, 255, 255);
        --bs-btn-primary-hover-bg: rgb(19, 136, 113);
        --bs-btn-primary-hover-color: rgb(255, 255, 255);
        --bs-focus-ring-color: 57, 174, 151;
        --bs-input-focus-border-color: rgb(139, 208, 194);
        --bs-link-color: rgb(22, 160, 133);
        --bs-link-color-rgb: 22, 160, 133;
        --bs-link-hover-color: rgb(18, 128, 106);
        --bs-link-hover-color-rgb: 18, 128, 106;
        --bs-primary: rgb(22, 160, 133);
        --bs-primary-bg: rgb(232, 246, 243);
        --bs-primary-bg2: rgb(208, 236, 231);
        --bs-primary-rgb: 22, 160, 133;
        --illustrations-color: rgb(241, 196, 15);
    }
</style>
```

For more information about what variables can be overridden, see the CSS variables in Bootstrap documentation.

Once we have the file ready, we need to mount it into the container:

```
dsw-client:
    volumes:
    - /path/to/head-extra.html:/src/head-extra.html
```

Document Templates

We can freely customize and style templates of documents (DMPs). HTML and CSS knowledge is required, and for doing more complex templates that use some conditions, loops, or macros, knowledge of Jinja templating language (pure Python implementation) is useful. For more information, please read *the following section*.

Email Templates

Similarly to document templates, we can customize templates for emails sent by the Wizard located in `templates` folder. It also uses Jinja templating language. And we can create HTML template, Plain Text template, add attachments, and add inline images (which can be used inside the HTML using Content-ID equal to the filename). We can learn more about the template structure and contents directly from the mailer GitHub repository.

Including our own email templates while using dockerized Wizard is practically the same as for DMP templates. We can also bind whole `templates` folders. (or even `templates` if we want to change both):

```yaml
mailer:
    image: datastewardshipwizard/mailer
    restart: always
    depends_on:
    - postgres
    - dsw-server
    volumes:
    - ./config/application.yml:/app/config/application.yml:ro
    - ./templates:/home/user/templates:ro
# ... (continued)
```

1.8.3 Upgrade Guidelines

Upgrading DSW

> **Warning:** Backup database and other important data (e.g., configuration) before upgrade!

Using Docker

In case of using Docker, just use the tag in `docker-compose.yml` or pull the new Docker image and restart using down/up:

```
$ docker pull datastewardshipwizard/wizard-server
$ docker pull datastewardshipwizard/wizard-client
$ docker pull datastewardshipwizard/document-worker
$ docker pull datastewardshipwizard/mailer
$ docker-compose down
$ docker-compose up -d
```

Other setup (using Git)

All we need to do is download or checkout the new version from GitHub repositories and rebuild the application (according to the guidelines above):

```
$ git checkout vX.Y.Z
```

Upgrade process

Usually, nothing special is required for the upgrade. Internal structure changes are migrated automatically using DB migrations and Metamodel migrations *(since 1.8.0)*. See below the changes that need to be done by us *(since 1.10.0)*:

> **Warning:** Make sure to stop `document-worker` and `mailer` before upgrading to the next version. Run `wizard-server` first, then run the workers. Otherwise the database migrations might not work correctly.

4.4.X to 4.5.X

(nothing)

4.3.X to 4.4.X

(nothing)

4.2.X to 4.3.X

- The document template metamodel version is raised to 13.

4.1.X to 4.2.X

(nothing)

4.0.X to 4.1.X

- **(breaking)** DSW Integration Widget SDK for *Integration Question - Widget* and DSW Importer SDK for *Project Importers Development* are now deprecated. DSW Integration SDK should be used instead.
- The document template metamodel version is raised to 12.

3.28.X to 4.0.X

- **(breaking)** The client runs on the nested route `/wizard`, and the server runs on the nested route `/wizard-api`. These changes must be reflected in the deployment configuration (such as routing in a reverse proxy) and the `clientUrl` in the *Server Configuration*.

- **(breaking)** *Client Configuration* for a custom logo and theme no longer uses SASS and needs to be updated accordingly.

- **(breaking)** *API Keys* created in previous versions will no longer work.

- **(breaking)** The callback URL for *OpenID* (such as Google or Life Science Login) has changed due to the nested route, so it has to be set up, for example:

  - Before: `https://researchers.ds-wizard.org/auth/google/callback`

  - After: `https://researchers.ds-wizard.org/wizard/auth/google/callback`

3.27.X to 3.28.X

(nothing)

3.26.X to 3.27.X

(nothing)

3.25.X to 3.26.X

(nothing)

3.24.X to 3.25.X

(nothing)

3.23.X to 3.24.X

(nothing)

3.22.X to 3.23.X

- **(breaking)** The JWT signing has been changed to RS256 and you need to add `rsaPrivateKey` in configuration file (see *Server Configuration*).

- **(breaking)** The location of configuration files has been changed and unified across components, check the deployment example for details. The main configuration file is located in `/app/config/application.yml` path which can be adjusted `APPLICATION_CONFIG_PATH`.

3.21.X to 3.22.X

(nothing)

3.20.X to 3.21.X

- **(breaking)** The `wizard-client` container now exposes a different port (as all images are now root-less): 8080 instance of 80.
- **(breaking)** The S3 service must be now publicly available, thus the S3 URL configured via *Server Configuration* must be reachable by users to support download of documents or document preview.

3.19.X to 3.20.X

- The document template metamodel version is raised to **11**. All templates must be updated (changes are only minor in template.json files, see Template Development section for more information).

3.18.X to 3.19.X

(nothing)

3.17.X to 3.18.X

(nothing)

3.16.X to 3.17.X

- If we are upgrading from the older version then 3.16.X we need to first upgrade to version 3.16.X.

3.15.X to 3.16.X

(nothing)

3.14.X to 3.15.X

(nothing)

3.13.X to 3.14.1

(nothing)

3.13.X to 3.14.X

(nothing)

3.12.X to 3.13.X

(nothing)

3.11.X to 3.12.X

(nothing)

3.10.X to 3.11.X

- (optional) We can now use integration.yaml configuration in Settings instead of the file store on FS and mounted to the Docker container.

3.9.X to 3.10.X

- Standalone mailer component has been introduced. We need to adjust our deployment (e.g., *docker-compose.yml*) accordingly (see deployment-example).

3.8.X to 3.9.X

(nothing)

3.7.X to 3.8.X

- All KM migrations must be finished (completed or deleted); otherwise, the upgrade of the backend (database) will fail with the corresponding message in the logs.

3.6.X to 3.7.X

(nothing)

3.5.X to 3.6.X

(nothing)

3.4.X to 3.5.X

- The template metamodel version has been updated (to v5). Updating all document templates is needed (annotations were added, so we can safely change version 4 to version 5 without breaking anything).
- All KM migrations must be finished (completed or deleted); otherwise, the upgrade of the backend (database) will fail with the corresponding message in the logs.

3.3.X to 3.4.X

(nothing)

3.2.X to 3.3.X

(nothing)

3.1.X to 3.2.X

- The template metamodel version has been updated (to v4). Updating all document templates is needed.
- All knowledge models have (after the automatic data migration) the default metrics and phases that can be changed in KM Editor.

3.0.X to 3.1.X

- As an administrator, we should either disable the "Project Templates" feature (Settings - Projects - Project Creation, select "Custom only") or prepare some project templates for our users to avoid confusion.

2.14.X to 3.0.X

- All data must be migrated as we switched from MongoDB and RabbitMQ to PostgreSQL and S3. To support data migration, we provide dsw2to3 tool with step-by-step instructions.

2.13.X to 2.14.X

(nothing)

2.12.X to 2.13.X

(nothing)

2.11.X to 2.12.X

- The metamodel for templates has been upgraded, and accessing the reply values is changed due to additional metadata about each reply, see *Document Context*. But if we are using filters such as `reply_str_value`, it gets the reply object with value correctly. Moreover, for working with integration reply, the type values are renamed `IntegrationValue` -> `IntegrationType` and `PlainValue` -> `PlainType` for consistency.

2.10.X to 2.11.X

- If we are using the `questionnaire-report` template, it is recommended to upgrade it to version 1.2.0 (from Registry or GitHub Release) so it displays also new Multi-Choice questions. Otherwise the choices won't appear in the exported document if there are any.

2.9.X to 2.10.X

(nothing)

2.8.X to 2.9.X

(nothing)

2.7.X to 2.8.X

(nothing)

2.6.X to 2.7.X

(nothing)

2.5.X to 2.6.X

- The document templates including the default `questionnaire-report` must be updated from https://registry.ds-wizard.org/templates.
- Upgraded template metamodel version 2 requires manual migration of custom templates:
  - *questionnaireRepliesMap* (map path:Reply) is no longer present in the context
  - *questionnaireReplies* is now map with path:ReplyValue, provided filters (such as `reply_str_value`) are adjusted but wherever we used `reply.value.value` it should be `reply.value` with this change.
  - Reply for item question is no longer an integer (number of answers) but a list of UUIDs representing the answers instead of integers. We added `reply_items` to extract the list from a ReplyValue.

- Since 2.6.0, we are using WebSockets (for live collaboration). If we are using a proxy, we need to configure it accordingly. For example, in case of Nginx:

```
server {
    # ...

    location / {
        # ...

        # required for websockets
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_read_timeout 86400;
        proxy_send_timeout 86400;
    }
}
```

2.4.X to 2.5.X

- Document templates have been moved from FS to database. To simplify the transition for custom templates, we added to the Docker image a script that loads templates from FS to the database via DSW API. But there are several new information that we need to provide in `template.json` file: `id` (instead of `uuid`), `templateId`, `organizationId`, `version` (semver), `license`, `readme` (Markdown). The `id` should be in format `organizationId:templateId:version`. Please note that this applies only for custom templates, default template can be removed from FS as it is added to the database automatically. The script must be enabled by setting envvar `ENABLE_TEMPLATE_LOAD `` to ``1` and `SERVICE_TOKEN` according to the configuration.

- Cron is no longer needed for the feedback synchronization (environment variables in `docker-compose.yml`) as DSW schedules synchronization internally.

2.3.X to 2.4.X

- To unify configuration, document-worker now supports and prefers YAML configuration files.

- Local/custom `template.json` files must be updated (renamed `allowedKMs` to `allowedPackages`, and several new attributes: `description` for template and `shortName` + `color` for each format).

2.2.X to 2.3.X

(nothing)

2.1.X to 2.2.X

- Configuration of client and several features is now moved from `application.yml` file to in-app *Settings*; therefore, it must be reconfigured during upgrade process. Additional `secret` must be configured in `application.yml` for encryption and JWT tokens (*JWT.secret* section has been removed), see *Server Configuration* configuration. It is recommended to first add *general.secret* (32 chars secret), start DSW, migrate options from `application.yml` to *Settings* and then optionally clean up `application.yml` file.

- User fiels `name` and `surname` has been renamed to `firstName` and `lastName` - it needs be updated if used in **custom** mail or document templates.

- Recommended version of MongoDB is updated to 4.2.3.

2.0.X to 2.1.X

- There is a significant change related to new *Document Worker* that handles generation of documents from templates and filled questionnaires. We need to run RabbitMQ and document-worker with correct configuration according to server, see *Deployment with Docker* and *Configuration* for details.

1.10.X to 2.0.X

- Changing the major version actually does not mean any problem in migration, it has been made due to significant internal changes (restructuring, new repositories, etc.)

- If we are using Docker for running DSW, we need to change it according to new documentation of *Deployment with Docker* and *Configuration*.

- Crontab image is no longer needed.

- A DMP template configuration file must contain list of `allowedKMs` (see the default *root* template).

1.9.X to 1.10.X

- Custom DMP templates needs to be upgraded to a new structure (see the default *root* template).

Compatibility

Important: DSW components (server, client, document worker, mailer, registry) should always use the matching version (compatibility is assured)!

The DSW is compatible with all recent versions of web browsers Chrome, Opera, Firefox, and Edge. We do not recommend the use of Internet Explorer.

The following table shows the compatibility of the DSW with the metamodel versions of Knowledge models, Document Templates, Project Importers, and the Registry.

Wizard	KM Metamodel	Document Template Metamodel	Project Importer Metamodel	Registry
3.24.0	13	11	1	3.24.0
3.23.0	13	11	1	3.23.0
3.22.0	13	11	1	3.22.0
3.21.0	13	11	1	3.21.0
3.20.0	13	11	1	3.20.0
3.19.0	13	10	1	3.19.0
3.18.0	13	10	1	3.18.0
3.17.0	13	10	1	3.17.0
3.16.0	13	10	1	3.16.0
3.15.0	13	10	1	3.15.0
3.14.0	13	10	–	3.14.0
3.13.0	13	10	–	3.13.0
3.12.0	13	10	–	3.12.0
3.11.0	12	9	–	3.11.0
3.10.0	12	9	–	3.10.0
3.9.0	11	8	–	3.9.0
3.8.0	11	8	–	3.8.0
3.7.0	10	7	–	3.7.0
3.6.0	10	6	–	3.6.0
3.5.0	9	5	–	3.5.0
3.4.0	8	4	–	3.4.0
3.3.0	8	4	–	3.3.0
3.2.0	8	4	–	3.2.0
3.1.0	7	3	–	3.1.0
3.0.0	7	3	–	3.0.0
2.14.0	7	3	–	2.14.0
2.13.0	7	3	–	2.13.0
2.12.0	6	3	–	2.12.0
2.11.0	5	2	–	2.11.0
2.10.0	5	2	–	2.10.0
2.9.0	5	2	–	2.9.0
2.8.0	5	2	–	2.8.0
2.7.0	5	2	–	2.7.0
2.6.0	5	2	–	2.6.0
2.5.0	5	1	–	2.5.0
2.4.0	5	–	–	2.4.0
2.3.0	5	–	–	2.3.0
2.2.0	5	–	–	2.2.0
2.1.0	5	–	–	2.1.0
2.0.0	5	–	–	2.0.0
1.10.0	4	–	–	1.2.0
1.9.0	3	–	–	1.1.0
1.8.0	3	–	–	1.0.0
1.7.0	2	–	–	–
1.6.0	1	–	–	–
1.5.0	–	–	–	–
1.4.0	–	–	–	–
1.3.0	–	–	–	–
1.2.0	–	–	–	–
1.1.0	–	–	–	–
1.0.0	–	–	–	–

1.8.4 FAQ and Deployment Notes

Frequenty Asked Questions

This section tries to cover the common deployment issues people have and suggest what next steps should be done.

Why something is not running; what should I do?

Check what is not running using *docker-compose ps* and then use also *docker-compose logs <service>* to check what the issue is.

Why I cannot upload locales/templates and document generation fails?

You probably have some issue with S3 configuration or its deployment. Also, check whether you have S3 bucket present with correct name.

Why I cannot download files from DSW or generate document preview?

Your S3 is probably not accessible by users. The S3 URL configured in *Server Configuration* should be reachable so users can download something from the storage.

There is some issue with the PostgreSQL database; what should I do?

Please use the PostgreSQL documentation to check the cause, various things may have happened... especially if you tried to upgrade the database version.

There is some issue with the MinIO S3 storage; what should I do?

Please use the MinIO documentation to check the cause, various things may have happened... especially if you tried to upgrade the storage version.

I upgraded DSW and now it does not work properly, what should I do?

You should always check *Upgrade Guidelines* before upgrading, be sure that you followed all steps. In case you forgot and it is not possible to fix it now, you will have to rollback from you backup and do it again by following the guidelines this time. In case you encounter an issue even though you followed the guidelines, that might a bug and please report it.

Document templates show "Unsupported Metamodel", what should I do?

You need to update your document templates so those are compatible with your DSW version, e.g. from DSW Registry. If those are your own document templates, you need to update them according to *Upgrade Guidelines*.

You can follow this guide:

1. Go to Settings -> Content Settings -> DSW Registry

2. Click on Enabled

3. Click on Sign Up

4. Fill out your email

5. With a token you will get to login to https://registry.ds-wizard.org

6. Open Questionnaire Report

7. Copy the Template ID

8. Go back to Wizard -> Document Templates -> List

9. Click on Import

10. Paste the Template ID

11. Click on Import

Deployment Notes

- You should be knowledgeable with at least basics of server management, service operations, work with Docker, and debugging issues (e.g. accessing Docker logs).

- The deployment can vary significantly based on needs and available infrastructure, we cannot help with different kinds of deployments and technologies that we are not experts with.

- The deployment example serves for local testing purposes and should not be used as is for production. An expert should deploy DSW for production while considering local needs and capabilities.

- Never update production instance without backups and preferrably try the update procedure first on a testing environment.

- Running DSW locally is not "free", you need people, time, and infrastructure. With that in mind, consider what option is the most suitable for you.

1.9 Development

DSW can be extended in many ways and new components and ways of integrations can be developed to support our needs. Besides the API available for everything that can be done in DSW, new *integration questions* and *project importers* can be implementet to get data from outside to DSW, or new *document templates* and *submission services* can be created to get the data outside of DSW in the desired form.

This section provides information on how to develop custom content for DSW to fully tailor the tool to our specific requirements.

1.9.1 Metamodel Schemas

As Data Stewardship Wizard evolves, the internal structures may change during the time. To support migration under the hood, we use metamodel versioning for KM and templates.

KM Package (.km file)

File for import and export of Knowledge Models is a JSON file that contains all KM packages (lists of change events with additional metadata). The structure of events is versioned using the KM metamodel version number and migrations in DSW automatically update the KMs when needed. As said, files according to this schema can be exported from *Knowledge Model List* or *Knowledge Model Detail* and then used for *Knowledge Model Import*.

Metamodel Version	Schema file	Changes (brief)	Since
14	JSON Schema	Optional Integration fields	4.1.0
13	JSON Schema	New question value types	3.12.0
12	JSON Schema	Enhanced integration (e.g. widget type)	3.10.0
11	JSON Schema	Annotations and timestamps for events	3.8.0
10	JSON Schema	Integrations with item template	3.6.0
9	JSON Schema	Annotations	3.5.0
8	JSON Schema	Metrics and phases are part of KM	3.2.0
7	JSON Schema	KM name attribute removed	2.13.0
6	JSON Schema	Multi-choice question type added	2.11.0
5	JSON Schema	Move event	2.0.0
4	JSON Schema	Refactored KM, optional chapter text	1.10.0
3	JSON Schema	Changed integration question fields	1.8.0
2	JSON Schema	Changed phases representation	1.7.0
1	JSON Schema	Initial versioned metamodel	1.6.0

Document Context

Document Context is provided to the document templates by document worker. It contains all relevant data about project/questionnaire with replies, related knowledge model, author, and more. As KM evolves, the context may evolve as well. It is versioned using the Template metamodel version number. A document template must support the metamodel that is in the current DSW instance. It is needed to know how the document context looks like especially for *Document Template Development*.

Metamodel Version	Schema file	Changes (brief)	Since
13	JSON Schema	Removed states from templates	4.3.0
12	JSON Schema	Optional Integration fields	4.1.0
11	JSON Schema	Change template metadata	3.20.0
10	JSON Schema	New question value types	3.12.0
9	JSON Schema	Enhanced integration (e.g. widget type)	3.10.0
8	JSON Schema	Annotations change	3.8.0
7	JSON Schema	Project tags and description	3.7.0
6	JSON Schema	Integrations with item template	3.6.0
5	JSON Schema	Annotations	3.5.0
4	JSON Schema	Metrics and phases	3.2.0
3	JSON Schema	Project versions	2.12.0
2	JSON Schema	Reply provenance	2.6.0
1	JSON Schema	Initial versioned metamodel	2.5.0

Template (.json file)

Each template has its descriptor file `template.json` which contains all the information about the template, its format(s) and steps how to produce the document(s). It is also versioned by the Template metamodel version number. This file also contains the actual number of the supported version... With local *Document Template Development*, we will need to manage the file according to the schema; however, when *Document Template Editors* are used, we will define it using forms directly in DSW.

Note: Between versions 1 and 5, the structure of `template.json` is still the same. Only the document context has been changed.

Metamodel Version	Schema file	Changes (brief)	Since
13	JSON Schema	Removed states from templates	4.3.0
12	JSON Schema	Optional Integration fields	4.1.0
11	JSON Schema	Change template metadata	3.20.0
10	JSON Schema	New question value types	3.12.0
9	JSON Schema	Enhanced integration (e.g. widget type)	3.10.0
8	JSON Schema	Annotations change	3.8.0
7	JSON Schema	Project tags and description	3.7.0
6	JSON Schema	Integrations with item template	3.6.0
5	JSON Schema	Annotations	3.5.0
4	JSON Schema	Metrics and phases	3.2.0
3	JSON Schema	Project versions	2.12.0
2	JSON Schema	Reply provenance	2.6.0
1	JSON Schema	Initial versioned metamodel	2.5.0

1.9.2 Document Template Development

Document templates allows to specify how to export a questionnaire in form of a textual file. It is a highly flexible element of the tool; however, the development requires basic programming skills with Jinja2 templating language. We can develop the document templates either on our local computer (traditional development with text editor or IDE) with use of the *Template Development Kit* (TDK) or directly in DSW using *Document Template Editors*.

Every document template is based on the *template specification* and typically uses the *document context* to query information from a project (questionnaire replies, knowledge model, metadata, etc.) to create a document.

Examples

- ds-wizard/questionnaire-report-template

- ds-wizard/madmp-template

- ds-wizard/horizon-europe-dmp-template

Document Context

Note: To work efficiently with the Document Context, you want to use object instead of the JSON-like one. Please read through DocumentContext.md directly (select different version if needed).

Document context is an object that carries all information related to a DSW questionnaire in order to produce a document. To investigate it, it is the best to use *Questionnaire Report* template with `JSON` format. The core fields are:

- `config` = object with DSW configuration related to documents, e.g., `clientUrl` for referring to the DSW instance

- `createdAt` = timestamp when the document was created

- `createdBy` = object describing author of the document

- **`knowledgeModel` = object describing used KM for the questionnaire**

    - `chapterUuids` = list of UUIDs for chapters

    - `integrationUuids` = list of UUIDs for integrations

    - `tagUuids` = list of UUIDs for tags

    - `entities` = contains `questions`, `answers`, and other maps with UUID-entity pairs

    - `name` = name of the knowledge model

    - `uuid` = UUID of the knowledge model

- `level` = current desirability level selected for the questionnaire

- `levels` = list of desirability levels possible

- `metrics` = list of available metrics

- `organization` = object describing organization that runs the DSW instance

- `package` = object with metadata about the KM package such as `version`, `name`, or `description`

- `questionnaireName` = name of the questionnaire

- `questionnaireReplies` = map of replies with path-reply pairs, each reply has `type` and `value`
- `questionnaireUuid` = UUID of the questionnaire
- `report` = object that contains report for the questionnaire that contains computed information about number of answered questions as well as metric values
- `updatedAt` = timestamp when the document was last updated
- `uuid` = UUID of the document

This structure is provided to a Jinja template in *Step: jinja* and outputed from *Step: json*. We can use the JSON step to observe the actual content of the document context (structure as well as the values). Finally, we can also check *Metamodel Schemas* (the relevant JSON schema for document context).

Objectified Document Context

It is possible to easily turn the JSON-like / tree-structured document context into objects with additional helper relations, attributes, methods, and many more to ease up the template development:

```
{%- set dc = ctx|to_context_obj -%}
```

- All data types are using Python, e.g., `str` is textual string, `Optional[str]` is a string or `None`, `list[str]` is a list of strings.
- We use `snake_case` for naming of attributes and variables, `PascalCase` is used for class names.
- `datetime` is the standard datetime module.

Diagram

We provide the structure visualized on a class diagram (right-click and open in to tab to enlarge):
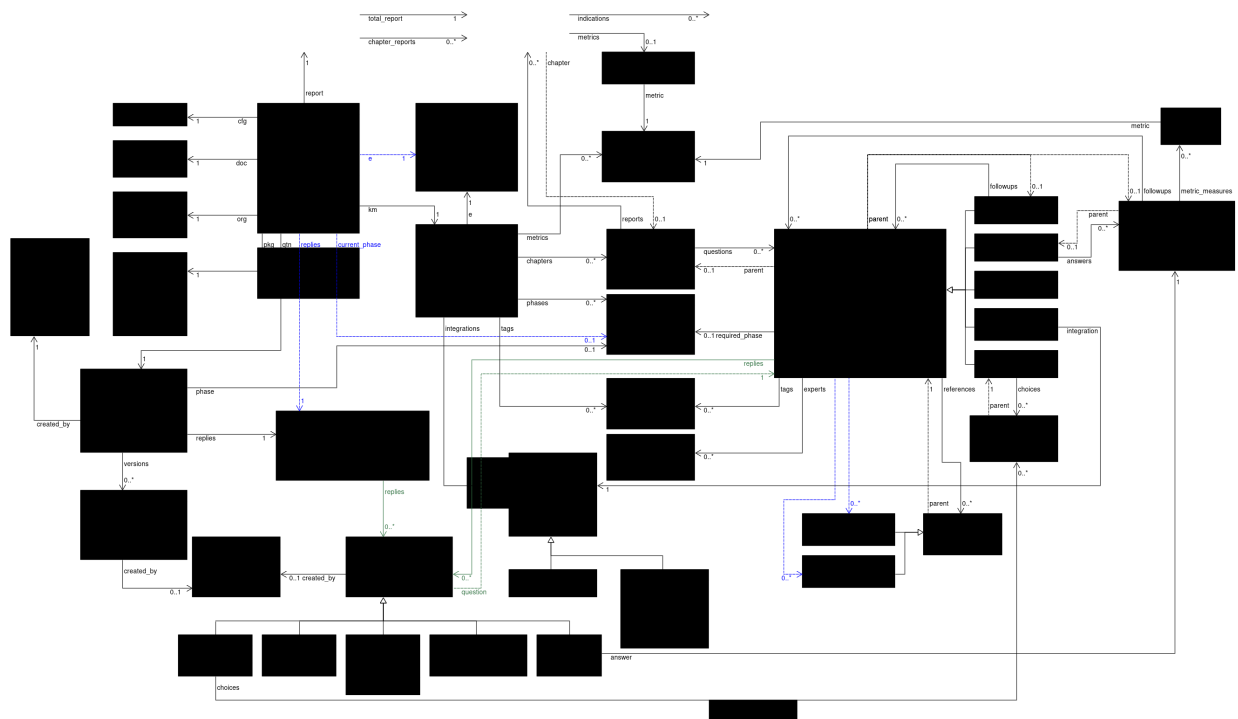
Entities

Here is an interlinked description of each entity and its attributes and links. There are also *aliases* that are convenient shorthands to make template more concise.

DocumentContext

- config (*ContextConfig*)
- current_phase (Optional[*Phase*])
- document (*Document*)
- km (*KnowledgeModel*)
- organization (*Organization*)
- package (*Package*)
- questionnaire (*Questionnaire*)
- report (*Report*)

Aliases:

- e (*KnowledgeModelEntities*) - same as km.entities
- doc (*Document*) - same as document
- org (*Organization*) - same as organization
- pkg (*Package*) - same as package
- qtn (*Questionnaire*) - same as questionnaire
- replies (*RepliesContainer*) - same as questionnaire.replies

ContextConfig

- client_url (str) - base URL of the DSW instance (client app)

Document

- uuid (str)
- created_at (datetime)
- updated_at (datetime)

Organization

- `id` (`str`)
- `name` (`str`)
- `description` (`Optional[str]`)
- `affiliations` (`list[str]`)

Package

- `id` (`str`) - full ID of KM Package
- `organization_id` (`str`)
- `km_id` (`str`)
- `version` (`str`)
- `versions` (`list[str]`)
- `name` (`str`)
- `description` (`Optional[str]`)
- `created_at` (`datetime`)

Questionnaire

- `uuid` (`str`)
- `name` (`str`)
- `version` (`Optional[`*QuestionnaireVersion*`]`)
- `versions` (`list[`*QuestionnaireVersion*`]`)
- `phase` (`Optional[`*Phase*`]`)
- `replies` (*RepliesContainer*)
- `created_by` (*User*)

QuestionnaireVersion

- `uuid` (`str`)
- `event_uuid` (`str`)
- `name` (`str`)
- `description` (`Optional[str]`)
- `created_by` (*SimpleAuthor*)
- `created_at` (`datetime`)
- `updated_at` (`datetime`)

User

- `uuid` (`str`)
- `first_name` (`str`)
- `last_name` (`str`)
- `email` (`str`)
- `role` (`str`) - one of: `admin`, `dataSteward`, `researcher`
- `image_url` (`Optional[str]`)
- `affiliation` (`Optional[str]`)
- `permissions` (`list[str]`)
- `sources` (`list[str]`)
- `created_at` (`datetime`)
- `updated_at` (`datetime`)

SimpleAuthor

- `uuid` (`str`)
- `first_name` (`str`)
- `last_name` (`str`)
- `image_url` (`Optional[str]`)
- `gravatar_hash` (`Optional[str]`)

Report

- `uuid` (`str`)
- `total_report` (*ReportItem*)
- `chapter_reports` (`list[`*ReportItem*`]`)
- `created_at` (`datetime`)
- `updated_at` (`datetime`)

ReportItem

- `indications` (`list[`*ReportIndication*`]`)
- `metrics` (`list[`*ReportMetric*`]`)
- `chapter` (`Optional[`*Chapter*`]`) - set if it is a chapter report

ReportIndication

- `indication_type` (`str`) - one of: `PhasesAnsweredIndication`, `AnsweredIndication` (use alias)
- `answered` (`int`) - number of answered questions
- `unanswered` (`int`) - number of unanswered questions

Aliases:

- `total` (`int`) - `answered` + `unanswered`
- `percentage` (`float`) - `answered` / `total` (handles zero division, number between `0.0` and `1.0`)
- `is_for_phase` (`bool`) - if it is a phase-related indication
- `is_overall` (`bool`) - if it is an overall indication (not phase-related)

ReportMetric

- `measure` (`float`) - number between `0.0` and `1.0`
- `metric` (*Metric*)

KnowledgeModel

- `uuid` (`str`)
- `annotations` (`dict[str,str]`)
- `entities` (*KnowledgeModelEntities*)
- `chapters` (`list[`*Chapter*`]`)
- `integrations` (`list[`*Integration*`]`)
- `metrics` (`list[`*Metric*`]`)
- `phases` (`list[`*Phase*`]`)
- `tags` (`list[`*Tag*`]`)

Aliases:

- `e` (*KnowledgeModelEntities*) - same as `entities`
- `a` (`dict[str,str]`) - same as `annotations`

Notes:

- Equality of all KM entities is being done using the `uuid` comparison under the hood.
- All KM entities that have `annotations` have also the `a` alias.

KnowledgeModelEntities

Container holding all types of Knowledge Model entities within UUID-key dictionaries:

- `answers` (dict[str, *Answer*])
- `chapter` (dict[str, *Chapter*])
- `choices` (dict[str, *Choice*])
- `experts` (dict[str, *Expert*])
- `integrations` (dict[str, *Integration*])
- `metrics` (dict[str, *Metric*])
- `phases` (dict[str, *Phase*])
- `questions` (dict[str, *Question*])
- `references` (dict[str, *Reference*])
- `tags` (dict[str, *Tag*])

Chapter

- `uuid` (str)
- `title` (str)
- `text` (Optional[str]) - possibly Markdown text
- `questions` (list[*Question*])
- `reports` (list[*ReportItem*])
- `annotations` (dict[str, str])

Question

Superclass with common attributes for all types of questions. You always get a more specific one and never just a `Question`.

- `uuid` (str)
- `type` (str)
- `title` (str)
- `text` (Optional[str])
- `required_phase` (Optional[*Phase*])
- `is_required` (bool) - if the question is required in the current phase
- `replies` (dict[str, *Reply*]) - path-key dictionary of replies to the question
- `experts` (list[*Expert*])
- `references` (list[*Reference*])
- `tags` (list[*Tag*])
- `parent` (Union[*Chapter*, *ListQuestion*, *Answer*])

- annotations (dict[str,str])

Aliases:

- url_references (list[*URLReference*])

- resource_page_references (list[*ResourcePageReference*])

Notes:

- Parent of a question can be of multiple kinds, you may use the of_type test to check what it is if needed.

ValueQuestion

- value_type (str) - type of value, use alias

Aliases:

- is_string (bool)

- is_text (bool)

- is_number (bool)

- is_date (bool)

IntegrationQuestion

- integration (*Integration*)

- props (dict[str,str])

OptionsQuestion

- answers (list[*Answer*])

MultiChoiceQuestion

- choices (list[*Choice*])

ListQuestion

- followups (list[*Question*])

Answer

- uuid (str)

- label (str)

- advice (Optional[str]) - possibly Markdown text

- metric_measures (list[*MetricMeasure*])

- followups (list[*Question*])

- parent (*OptionsQuestion*)
- annotations (dict[str,str])

MetricMeasure

Indication of how an answer affects a certain metric.

- measure (float) - value between 0.0 and 1.0 (inclusive)
- weight (float) - value between 0.0 and 1.0 (inclusive)
- metric (*Metric*)

Choice

- uuid (str)
- label (str)
- parent (*MultiChoiceQuestion*)
- annotations (dict[str,str])

Expert

- uuid (str)
- name (str)
- email (str)
- annotations (dict[str,str])

Reference

As for the *Question* class, Reference is also a superclass and you will always get an object of its subclass.

- uuid (str)
- type (str)
- annotations (dict[str,str])

URLReference

- label (str)
- url (str)

ResourcePageReference

- `short_uuid` (str)
- `url` (str) - URL composed using `client_url` from *ContextConfig*

Metric

- `uuid` (str)
- `title` (str)
- `abbreviation` (str)
- `description` (Optional[str]) - possibly Markdown text
- `annotations` (dict[str,str])

Phase

- `uuid` (str)
- `title` (str)
- `description` (Optional[str]) - possibly Markdown text
- `order` (int) - order of the phase within the KM
- `annotations` (dict[str,str])

Integration

- `uuid` (str)
- `id` (str)
- `name` (str)
- `item_url` (Optional[str])
- `logo` (Optional[str])
- `props` (dict[str,str])
- `rq_method` (str)
- `rq_url` (str)
- `rq_headers` (dict[str,str])
- `rq_body` (str)
- `rs_list_field` (Optional[str])
- `rs_item_id` (Optional[str])
- `rs_item_template` (str)
- `annotations` (dict[str,str])

Operations:

- `item(item_id:  str) -> Optional[str]` - URL of an item identified by string ID

Tag

- `uuid` (`str`)
- `name` (`str`)
- `description` (`Optional[str]`) - possibly Markdown text
- `color` (`str`)
- `annotations` (`dict[str,str]`)

RepliesContainer

Wrapper around a path-key dictionary of replies.

- `replies` (`dict[str,`*Reply*`]`)

Operations:

- `X[path:  str]` (`Optional[`*Reply*`]`) - you can get a reply using square brackets
- `len(X)` (`int`) - number of replies in the container
- `get(path:  str) -> Optional[`*Reply*`]`
- `iterate_by_prefix(path_prefix:  str) -> Iterable[`*Reply*`]` - *O(n)* iteration with filter
- `iterate_by_suffix(path_suffix:  str) -> Iterable[`*Reply*`]` - *O(n)* iteration with filter
- `values() -> Iterable[`*Reply*`]`
- `keys() -> Iterable[str]`
- `items() -> ItemsView[str,`*Reply*`]`

Reply

Superclass with common attributes for all types of replies. You always get a more specific one and never just a `Reply`.

- `path` (`str`)
- `fragments` (`list[str]`) - UUIDs of the path (starting with chapter)
- `type` (`str`)
- `created_at` (`datetime`)
- `created_by` (*SimpleAuthor*)
- `question` (*Question*) - you can assume more specific type of `Question` based on a type of `Reply`

AnswerReply

- answer (*Answer*) - selected answer as the option

Aliases:

- value (`str`) - UUID of the answer (`answer.uuid`)

Notes:

- `question` is always *OptionsQuestion*

MultiChoiceReply

- choices (`list[`*Choice*`]`) - selected answer as the option

Aliases:

- value (`list[str]`) - list of UUIDs of the choices

Notes:

- `question` is always *OptionsQuestion*
- You can iterate directly over reply object(`for choice in reply`)

StringReply

- value (`str`)

Aliases:

- as_number (`Optional[float]`) - tries to cast the value to a number
- as_datetime (`Optional[datetime]`) - tries to cast the value to a timestamp

Notes:

- `question` is always *ValueQuestion*

ItemListReply

- items (`list[str]`) - list of item UUIDs (used in reply paths)

Aliases:

- value (`list[str]`) - same as `items`

Notes:

- `question` is always *ListQuestion*
- You can iterate directly over reply object (`for item in reply`)

IntegrationReply

- `value` (`str`)
- `item_id` (`Optional[str]`) - ID of item if selected using *Integration*

Aliases:

- `id` (`Optional[str]`) - same as `item_id`
- `is_plain` (`bool`) - entered by user ignoring the integration
- `is_integration` (`bool`) - selected by user using the integration
- `url` (`Optional[str]`) - item URL based *Integration* if selected from it

Template Development Kit

Note: **Requirements for Local Template Development**

- Your favorite text editor or IDE
- Template Development Kit (see below)
- DSW instance (recommended to have local one) with your admin account
- Python 3.10+ (with pip) or Docker

Our Template Development Kit (TDK) provides a simple way how to work with templates locally. It is a CLI tool written in Python.

Video Tutorial

This is a comprehensive video tutorial on how to use the Template Development Kit.

https://youtu.be/FFElv-e24NE

Installation

You can install it easily using pip from Python Package Index (PyPI). Optionally, you can use virtual environment or other installation option described in the TDK repository.

```
pip install dsw-tdk
dsw-tdk --help
```

It is also possible to use datastewardshipwizard/dsw-tdk Docker image when you don't have Python locally:

```
docker run datastewardshipwizard/dsw-tdk --help
```

Commands

There are these basic commands:

- `new` = create a new template project, it launches a simple interactive wizard for template metadata
- `list` = list all templates (latest versions) from configured DSW
- `get` = download a template project with specified template ID from DSW
- `put` = upload the local template project to DSW (once or continually on-change when `--watch` flag is used)
- `verify` = check the metadata of the local template project
- `package` = create a ZIP distribution package from the local template project (ZIP is importable to DSW via its web interface)

Default template directory is current one for `put`, `verify`, and `package`. But `new` and `get` will create a new folder according to the template ID if not explicitly set in other way.

You can use `--help` to find out details:

```
dsw-tdk new --help
```

Environment variables and .env file

You can use environment variables to authenticate:

- `DSW_API_URL` = URL of DSW API with which you want to communicate. Hover mouse over your profile name to find the About section where URL is specified.
- `DSW_API_KEY` = your API Key. Hover mouse over your profile name, click on *Edit Profile* and then navigate to *API Keys* From there, you can generate a new API Key for the authentication.

To make this even easier, you can store those in `.env` file in the project root and it will be loaded automatically. Or you can specify the path to a `.env` file:

```
dsw-tdk --dot-env /path/to/.env list
```

Document Template Specification

Each document template in DSW has metadata stored. If developing locally with *Template Development Kit*, you can find and manage them in `template.json` file. In case of using *Document Template Editors*, you can manage them on **Settings** tab.

Specification Structure

- `id` = composed full ID of the template (`organizationId:templateId:version`)
- `organizationId` = identifier of organization developing the template (lowercase, numerics, dot)
- `templateId` = identifier of template (lowercase, numerics, dash)
- `version` = version (semver) in X.Y.Z format where X, Y, and Z are non-negative numbers
- `name` = name of the template
- `description` = short description of the template

- `license` = name of the used license

- `readme` = longer description usually containing changelog

- `metamodelVersion` = supported version of template metamodel, it affects with which DSW version is can be used

- `allowedPackages` = list of package filters (see *Package Filters*) to specify supported packages

- `formats` = list of available formats (see below *Formats*) with specified steps for generation

- `_tdk` = TDK configuration for local development (not stored in DSW, see *TDK Config*)

Note: TDK handles `id` and `readme` for you, so you can skip them and naturally use `README.md` file separately.

Package Filters

For filtering, the `null` value serves as wildcard, i.e., filter with all `null` values means that all packages are allowed.

- `orgId`: identifier of organization (e.g. `myorg`)

- `kmId`: identifier of knowledge model (e.g. `root`)

- `minVersion`: minimal package version (in format X.Y.Z, inclusive)

- `maxVersion`: maximal package version (in format X.Y.Z, inclusive)

Formats

A template can describe how to produce several formats, each with these metadata:

- `uuid`: UUID of the format (within template)

- `name`: display name of the format

- `icon`: icon style (CSS classes), preferably Font Awesome, e.g. `fas fa-file-word`

- `steps`: list of steps for document worker to produce the document with this format, each step has `name` and `options` (see *Steps*)

Steps

Each step of template produces output based on its (optional) input and options. Steps can be chained in order to generate the document and eventually transform it. All steps have always `name` and `options` based on one of the desired step. There are the details for steps supported by the *document worker* component:

Step: `archive`

Step that puts file from previous step to an archive file (ZIP or TAR).

Input

Any input file provided from the previous step.

Output

ZIP or TAR archive (based on `options`) containing the file from the previous step.

Options

- `inputFileDst` = destination of the file inside the archive (POSIX-like path including filename)
- (optional) `type` = whether to produce `zip` or `tar` (defaults to `zip`)
- (optional) `compression` = compression method to be used (`none`, `gzip`, `bzip2`, `lzma`; defaults to `none`)
- (optional) `compressionLevel` = value specifying level of compression (`0` to `9`; defaults to `9`)
- (optional) `format` = only for `tar` it allows to specify format (`ustar`, `gnu`, `pax`; defaults to `pax`)

Notes

- Currently, only a single file can be put into the produced archive.
- Value of `compressionLevel` must be provided as a string (even though it is a numeric value).
- For `zip`, zipfile standard library from Python is used.
- For `tar`, tarfile standard library from Python is used.
- For `bzip2`, if `compressionLevel` is set to `0`, it is automatically fixed to value 1.

Example

```
{
  "name": "archive",
  "options": {
    "type": "tar",
    "compression": "bzip2",
    "compressionLevel": "5",
    "format": "gnu",
    "inputFileDst": "example/file.html"
  }
}
```

Step: `enrich-docx`

| status | stable | metamodel version | ≥ 11 |

Enrichment step for MS Word (`docx`) documents.

Input

Gets a `docx` document as input.

Output

Results in a `docx` document as input.

Options

Options are used as a dictionary for rewrites with the following syntax:

- Keys can be prefixed with:
    - `rewrite:` and followed by path of file to be rewritten
    - *(currently there are no other prefixes then ``rewrite``)*
- Values can be prefixed with:
    - `static:` and followed by path to a file in a template; then it is used as-is to rewrite the original file in `docx`
    - `render:` and followed by path to a file in a template; then it is rendered first as `jinja` template with document context provided and result is used to rewrite the original file in `docx`

Notes

- Internally, the step unpacks the provided `docx` file, makes adjustments on the level of internal `XML` (and other) files, and packs it back to `docx`.
- To figure out what to rewrite, you should first generate the `docx` later used as input, unzip it and go through the contents.
- A good way to adjust things is to put there some placeholder first (e.g. via `reference.docx` passed to `pandoc`) and then just adjust the placeholder with other / dynamic content.
- Paths to files in a template are relative to template root, i.e. directory with `template.json`.
- It does not matter if the file to be rewritten is missing in the `docx`, then the desired file is simply added.
- The document context is provided in `ctx` variable, other variables, filters, and tests are documented in other documents (same as for `jinja` step).

Example

```
{
  "name": "enrich-docx",
  "options": {
    "rewrite:word/footer1.xml": "static:src/docx/footer1.xml",
    "rewrite:word/header1.xml": "render:src/docx/header1.xml.j2"
  }
}
```

Step: `excel`

status stable   metamodel version   ≥ 11

Step producing Excel spreadsheets from JSON file with instructions.

Input

JSON file containing instructions how to construct the desired Excel spreadsheet as described further in this section.

Properties

It allows to set both basic and custom properties of the workbook / spreadsheet.

```
{
  "properties": {
    "document": {
      "title": "My example workbook",
      "subject": "",
      "author": "Albert Einstein",
      "manager": "",
      "company": "ACME",
      "category": "",
      "keywords": "test,example,foo,bar",
      "created": "2018-01-01",
      "comments": ""
    },
    "custom": [
      {
        "projectUuid": "...",
      }
    ]
  }
}
```

Options

It allows to specify various workbook options.

```
{
  "options": {
    "strings_to_numbers": true,
    "strings_to_urls": true,
    "use_future_functions": true,
    "max_url_length": 255,
    "nan_inf_to_errors": true,
    "default_date_format": null,
    "remove_timezone": true,
    "use_zip64": false,
    "date_1904": false,
    "calc_mode": "auto",
    "read_only_recommended": false,
    "active_sheet": 0,
    "vba_name": "foo",
    "size": {
      "width": 0,
      "height": 0
    },
    "tab_ratio": 50
  }
}
```

Notes:

- `active_sheet` is an index of sheet to be active when document is opened.

- `size` sets the default window size.

- `tab_ratio` sets ratio between the worksheet tabs and the horizontal slider.

Definitions

It allows to define a name to be then used as a variable (see define_name).

```
{
  "definitions": {
    "Exchange_rate": "=0.96"
  }
}
```

Formats

It specifies formats in the spreadsheets that can be then used for cells in sheets. Possible options can be found in the documentation.

```
{
  "formats": {
    "myBoldFormat": {
      "bold": true
    }
  }
}
```

The example above creates a format named `myBoldFormat` that has bold text.

Charts

It specifies charts that can be then inserted inside sheets or used as chartsheets.

The options are documented here. Basically, each chart must have a unique `name` and then can have some `options`, series, and `axis` (e.g. x axis).

Finally, there are some basic and advanced settings:

- Basic: size, title, legend, chartarea, plotarea, style, table

- Advanced: combine, up_down_bars, drop_lines, high_low_lines, show_blanks_as, show_hidden_data

```
{
  "name": "myChartA",
  "combine": "myChartB",
  "options": {
    "type": "bar",
    "subtype": "percent_stacked"
  },
  "series": [
    {
      "name": "=Sheet1!$B$1",
      "categories": "=Sheet1!$A$2:$A$7",
      "values": "=Sheet1!$B$2:$B$7"
    }
  ],
  "axis": {
    "x": {"name": "Test number"},
    "y": {"name": "Sample length (mm)"}
  }
}
```

sheets

It is the main part specifying a list of sheets in the workbook, where each sheet has `name` (optional), `type` (optional, `work` or `chart`), `options` and then based on the `type` it has either `chart` (for chartsheet) or `data` (for worksheet). Some of the `options` are common for both chartsheet and datasheet. The order of sheets in the list corresponds to the order in the Excel spreadsheet.

Chartsheet

A chartsheet simply refers to a `chart` (by its `name`) that should be placed in this chartsheet.

The possible `options` are:

- Basic: first_sheet, protect, zoom, tab_color, page_view, select, hide

- Print: `orientation` (landspace or portrait), paper, margins, header, footer, center_horizontally, center_vertically

```
{
  "name": "Nice chart",
  "type": "chart",
  "chart": "myChartA",
  "options": {
    "tab_color": "red"
  }
}
```

Worksheet

Traditional worksheet with many options and data placed into cells. There are more options when compared to chartsheets.

The possible `options` are:

- Basic (common): first_sheet, protect, zoom, tab_color, page_view, select, hide

- Print (common): `orientation` (landspace or portrait), paper, margins, header, footer, center_horizontally, center_vertically

- Basic: comments_author, hide_zero, hide_row_col_headers, right_to_left, hide_gridlines, ignore_errors, vba_name

- Print (advanced): print_row_col_headers, print_area, print_across, fit_to_pages, start_page, print_scale, print_black_and_white

- Special ranges: unprotect_ranges, top_left_cell, selection

- Repeats: repeat_rows, repeat_columns, default_row

- Paging: h_pagebreaks, v_pagebreaks, outline_settings

- Panes: split_panes, freeze_panes

- Filters: filter_column_lists each with `col` and `filters`, filter_columns each with `col` and `criteria`, autofilter with range directly or `first_row`, `first_col`, `last_row`, `last_col` attributes (see docs)

- Merge ranges: merge_ranges list can be used to merge cells (`range` or combination of `first_row`, `first_col`, `last_row`, `last_col` attributes) together and apply a format (via `format` reference attribute), also contents can be set via `data` attribute

- Data validations: data_validations list can be used to validate data in cell ranges (see docs)

- Conditional formats: conditional_formats list can be used to define conditional formats on cell ranges (see docs)

- Tables: tables list can be used to define formatted tables (see docs)

- Sparklines: sparklines list (see docs)

- Row/col sizing: columns, column_pixels, rows, and row_pixels lists can be used to adjust cell sizing (unfortunately automatic sizing is not possible)

- Background: background can be used to set worksheet background via `filename` or `b64bytes` attributes

Inserting data

In JSON as part of worksheet's attribute `data`, you can in the list specify data to be inserted to cells in four ways (`type`):

- `cell` writes cell according to the possibly specified`subtype` (see below)

- `row` writes row using provided data as a list

- `column` writes column using provided data as a list

- `grid` writes rows using provided data as a list of lists (list of rows)

For data, there are the following subtypes possible (for `type` set to `cell`):

- (unspecified) tries to directly all write with provided arguments, type should be then decided based on provided values and attributes

- `string` writes string from `value`

- `number` writes number from numeric `value`

- `datetime` writes datetime; it tries to parse date/datetime `value` from string as JSON does not have a format for datetime, standard ISO formats are recommended (e.g. `2022-12-24` or `2022-12-24T12:00:00Z`)

- `formula` writes formula with formula in `value` and optional `result` value

- `blank` writes blank value (no attributes except `format`)

- `boolean` writes boolean value with boolean `value` (i.e. `true` or `false`)

- `url` writes URL value with `url`, `value`, and `tip` attributes

- `rich_string` writes rich string that allows formatting; for using format in the `string_parts` use prefix `!fmt::` before name of the desired format

All options above may specify `format` (refer to defined format via its `name`).

```json
{
  "type": "cell",
  "subtype": "string",
  "cell": "A1",
  "value": "X"
}
```

```json
{
  "type": "column",
  "subtype": "string",
  "cell": "A3",
  "data": [
    "ID",
    "Name",
    "Created at",
    "Author"
  ],
  "format": "myBoldFormat"
}
```

```json
{
  "type": "grid",
  "row": 5,
  "col": 5,
  "data": [
    ["A", "B", "C"],
    ["D", "E", "F"]
  ]
}
```

Inserting other elements

Aside from data in cell, there is also possibility to insert other elements to the worksheet (`type` vales):

- `button` with options such as `macro` or `caption`
- `textbox` with `text` and options such as styling or offset in pixels
- `comment` with `comment` text and options such as `color` or `author`
- `chart` with `chart` (name) and options
- `image` with `filename`, `b64bytes`, and options

All of these are used with corresponding `type` and are placed to desired `cell` (or `col`/`row` indices).

```json
{
  "type": "button",
  "cell": "B5",
  "options": {
    "caption": "Press Me"
  }
}
```

Header, Footer, Images

To allow easily add figures, those can be supplied as BASE64 encoded data directly via JSON as shown in the examples below.

For header and footer, the syntax of `content` is according to the documentation.

```json
{
  "header": {
    "content": "&L&G &CExample Excel Document",
    "options": {
      "image_left": "logo.png",
      "image_data_left": "data:image/png;base64,iVBORw0KGgoAA..."
    }
  }
}
```

```json
{
  "name": "bg-test",
  "options": {
    "background": { "b64bytes": "data:image/png;base64,iVBORw0KGgoAA..." }
  }
}
```

```json
{
  "type": "image",
  "cell": "D2",
  "filename": "logo.png",
  "b64bytes": "data:image/png;base64,iVBORw0KGgoAA...",
  "options": {
    "x_scale": 0.6,
    "y_scale": 0.6,
    "url": "https://ds-wizard.org"
  }
}
```

vba_projects

List of VBA projects (with macros) to be embedded in the spreadsheet.

```json
{
  "vba_projects": [
    {
      "project": "./vbaProject.bin",
      "is_stream": false
    }
  ]
}
```

Output

Desired Excel spreadsheet based on instructions from input JSON, it can be one the following formats (whether it uses macros or not):

- **application/vnd.openxmlformats-officedocument.spreadsheetml.sheet** (extension `.xlsx`)

- **application/vnd.ms-excel.sheet.macroEnabled.12** (extension `.xlsm`)

Options

No options, everything comes from the input JSON file

Notes

- [XlxsWriter](#) library is used to construct Excel spreadsheet.

- Most likely this step will follow `jinja` step that constructs the JSON file.

Example

```
{
  "name": "excel",
  "options": {}
}
```

Step: `jinja`

status stable   metamodel version   ≥ 1

Renders requested Jinja2 template with document context and optionally other data.

Input

If not used as a first step, then the previous document is available from `document` variable.

Output

Results to a file of specified type (via `content-type` option) and file extension (via `extension` option).

Options

- `template` = path to template file to be rendered
- `content-type` = MIME type of resulting file
- `extension` = file extension for the produced file (without leading dot)

Optional:

- `jinja-ext` = comma-separated list of Jinja2 extensions to be enabled (supported values: `debug`)
- `i18n-dir` = location (relative to template root) of translations
- `i18n-domain` = domain string of translations
- `i18n-lang` = language code used in the template
- `extras` = comma-separated list of related entities to query in addition to *Document Context* (possible values: `submissions`, `questionnaire`); values will be added to `extras` attribute of the document context

Template (Jinja2)

Variables

The following variables are set:

- `ctx` = contains JSON-like plain *Document Context* (possibly with `extras` attribute, if configured)
- `secrets` = dictionary of secret values, only if enabled by configuration file
- `requests` = wrapper of requests module only if enabled by configuration file

Filters

Within Jinja templates, you can use so-called filters.Basically, those are functions applied to a first argument using pipe | symbol.

Bultin Filters

There are several widely used builtin filters directly in Jinja.

Value Conversion

We provide several filters that can be used for conversion of values:

- `datetime_format` = *Formats timestamp*
  - Example: `x.created_at|datetime_format("%d/%m/%y")`
  - Arguments:
    * `iso_timestamp` - `datetime` or ISO 8601 `str`
    * `fmt` - datetime format passed to strftime
- `of_alphabet` = *Converts integer to characters*

- – Example: `x|of_alphabet`

- – It prints `a` (for 0) to `z` and then continues with `aa`, `ab`, etc.

- – Arguments:

  - * `n` - integer >= 0, usually some index

- `roman` = *Converts integer to Roman numeral*

  - – Example: `x|roman`

  - – Arguments:

    - * `n` - integer >= 0, usually some index

- `markdown` = *Converts markdown to HTML*

  - – Example: `x|roman`

  - – Arguments:

    - * `md_text` - string containing Markdown syntax

- `dot` = *Ends sentence if not already ended*

  - – Example: `"This sentence has no end"|dot`

  - – Arguments:

    - * `text`

- `extract` = *Extracts values from object by having keys*

  - – Example: `entities.questions|extract([uuid1, uuid2, uuid3])`

  - – Arguments:

    - * `obj` - object for getting values (typically `dict`)

    - * `keys` - list of keys to retrieve

Reply Helpers

These filters are handy when you need to work with `repliesMap` from the plain JSON-like context.

- `reply_path` = *Joins list of UUIDs into a path*

  - – Example: `[uuid1, uuid2, uuid3]|reply_path`

  - – Arguments:

    - * `uuids` - list of UUIDs

- `find_reply` = *Tries to find a reply value using a path*

  - – Example: `replies|find_reply(path, "list")`

  - – Arguments: - `replies` - dict with replies - `path` - list of UUIDs or path-string - `xtype` (optional) - desired type of return value (`"string"`, `"int"`, `"float"`, `"list"`)

- `reply_str_value` = *Extracts string value from a reply if possible*

  - – Returns an empty string if not possible to extract it from the reply. Suitable for `AnswerReply`, `StringReply` and `IntegrationReply`.

  - – Example: `reply|reply_str_value`

- **Arguments:** - `reply` - object that might a reply
- `reply_int_value` = *Extracts integer value from a reply if possible*
    - Returns zero if not possible to extract it from the reply. Suitable for `StringReply` with numeric value type.
    - Example: `reply|reply_int_value`
    - Arguments:
        * `reply` - object that might a reply
- `reply_float_value` = *Extracts float value from a reply if possible*
    - Returns zero if not possible to extract it from the reply. Suitable for `StringReply` with numeric value type.
    - Example: `reply|reply_float_value`
    - Arguments:
        * `reply` - object that might a reply
- `reply_items` = *Extracts list of strings from a reply if possible*
    - Returns empty list if not possible to extract it from the reply. Suitable for `MultiChoiceReply` and `ItemListReply`.
    - Example: `reply|reply_items`
    - Arguments:
        * `reply` - object that might a reply

Special

These filters are more complex and add various support to template development.

- `to_context_obj` = *Converts plain context to well-defined objects*
    - This filter is used for easier transition and might be removed in the future.
    - Arguments:
        * `ctx` - plain JSON-like document context

Tests

Within Jinja templates, you can use so-called tests. Basically, those are helpers usable in conditions after `is` keyword:

```
{% if loop.index is divisibleby 3 %}
    {# ... #}
{% endif %}
```

Bultin Tests

There are several widely used builtin tests directly in Jinja.

Custom Tests

- `not_empty` = *Checks if size of a collection is higher than 0*
  - Example: `items is not_empty`
- `of_type` = *Checks if an object is instance of a certain type / class*
  - The name must be a string; however, it is case-insensitive. It also checks all superclasses.
  - Example: `parent is of_type "ListQuestion"`

Notes

- All paths (e.g. for `import` or `extends` in Jinja2 templates are relative from the template root, i.e. directory with `template.json`).
- The do Jinja2 extension is enabled.
- Using file extension `.j2` or `.jinja2` for templates is just a convention.
- The *document context* is provided in `ctx` variable, other variables, filters, and tests are documented in other documents.

Example

```json
{
  "name" : "jinja",
  "options" : {
    "template" : "src/default.html.j2",
    "content-type" : "text/html",
    "extension" : "html"
  }
}
```

Step: `json`

status   stable   metamodel version   ≥ 1

Trivial step that dumps document context to JSON file.

Input

No input from previous step (should be first step)

Output

Always results in a JSON file (`application/json`) with file extension `.json`.

Options

No options

Example

```
{
  "name" : "json",
  "options" : {}
}
```

Step: `pandoc`

status   stable   metamodel version: ≥ 1

Transformation step that converts Pandoc-compatible document formats.

Input

Gets a file from the previous step (otherwise it fails), format needs to be specified using `from` option.

Output

Results in a document in desired format specified using `to` option.

Options

- `from` = specification of the input format (passed to Pandoc via `--from`, see docs)
- `to` = specification of the output format (passed to Pandoc via `--to`, see docs)
- (optional) `args` = additional command line arguments passed to pandoc
- (optional, experimental) `filters` = additional Pandoc filters to be used, need to be located under `/pandoc/filters` directory (or other set by `PANDOC_FILTERS` environment variable), comma separated
- (optional, experimental) `template` = Pandoc template to be used, need to be located under `/pandoc/templates` directory (or other set by `PANDOC_TEMPLATES` environment variable)

Notes

- Pandoc filter `pandoc-docx-pagebreakpy` can be found in addons directory.

- Pandoc filter `pandoc-docx-pagebreakpy` will be removed with the next template metamodel version, use `` `` for the `filters` option instead.

Example

```
{
  "name" : "pandoc",
  "options" : {
    "from" : "html",
    "to" : "docx",
    "args": "--filter=pandoc-docx-pagebreakpy --reference-doc=src/reference.docx",
    "filters": "docx-pagebreak.lua, docx-toc.lua"
  }
}
```

Step: `rdflib-convert`

status   stable   metamodel version: ≥ 1

Transformation step that converts between RDF formats.

Input

Gets an RDF file from the previous step of one of the following formats: `rdf` (XML), `nt`, `n3`, `ttl`, `trig`, or `json-ld` (specified using `from` option).

Output

Results in an RDF document of one of the following formats: `rdf` (XML), `nt`, `n3`, `ttl`, `trig`, or `json-ld` (specified using `from` option).

Options

- `from` = specification of the input format (`rdf`, `nt`, `n3`, `ttl`, `trig`, `json-ld`)

- `to` = specification of the output format (`rdf`, `nt`, `n3`, `ttl`, `trig`, `json-ld`)

Example

```json
{
  "name" : "rdflib-convert",
  "options" : {
    "from" : "ttl",
    "to" : "rdf"
  }
}
```

Step: `weasyprint`

status   stable   metamodel version   ≥ 12

Transformation step that converts HTML file from previous step to PDF using WeasyPrint.

Input

Gets HTML file from the previous step (otherwise it fails).

Output

Always results in a PDF file (`application/pdf`) with file extension `.pdf`.

Options

- (optional) `render.presentational_hints` = whether HTML presentational hints are followed (default: `False`)

- (optional) `render.optimize_size` = specify what should be optimized (`''`, `'fonts'`, `'images'`, `'fonts, images'`, default: `'fonts'`)

- (optional) `render.forms` = whether PDF forms have to be included (default: `False`)

- (optional) `pdf.zoom` = zoom value as a floating number (default: `'1'`)

- (optional) `pdf.variant` = a PDF variant name

- (optional) `pdf.version` = a PDF version number

- (optional) `pdf.custom_metadata` = whether custom HTML metadata should be stored in the generated PDF

Notes

- Check the official WeasyPrint documentation and examples for more information.

Example

```
{
  "name" : "weasyprint",
  "options" : {
    "render.optimize_size": "fonts,images",
    "render.forms": "True",
    "pdf.zoom": "1.2"
  }
}
```

TDK Config

Those are local-only metadata used for development of the template. You can use them in versioned `template.json` but those are never stored directly in DSW.

- `version`: metadata version for needs of migrations

- `readmeFile`: files used to get content for `readme` of the template, usually `README.md`

- `files`: list of patterns to specify files that are part of the document template (it uses Git wildcard-match patterns, so you can also exclude files or directories)

Template Metamodels

Here are described the changes in metamodel for template specification as well as *document context* so developers can easily update their templates to a newer metamodel version when needed. It is also possible to check JSON schemas in higher detail, see *Metamodel Schemas*.

Version 12 (since 4.1.0)

- Dropped support of deprecated `wkhtmltopdf` (for PDF, `weasyprint` is used instead).

- Changed several properties of `Integration` and `IntegrationReply` to optional (see *Document Context*).

Version 11 (since 3.20.0)

- Removed `recommendedPackageId` from template metadata and `shortName` together with `color` from formats.

Version 10 (since 3.12.0)

- New possible value types for value questions: `DateTimeQuestionValueType`, `TimeQuestionValueType`, `EmailQuestionValueType`, `UrlQuestionValueType`, and `ColorQuestionValueType` (no changes needed in existing KM-specific templates).

Version 9 (since 3.10.0)

- If you are using integration object, the `requestItemUrl` is changed to `itemUrl`.
- Integrations now have type, where the new Widget Integration has a different fields than API Integration (see schema).

Version 8 (since 3.8.0)

- Annotations and integration HTTP headers are changed from dict-like object with string-string key and value to a list of string-string tuples. Be aware that now there can be more values with the same "key" but that is usually unlikely.

Version 7 (since 3.7.0)

- Added description and project tags to the questionnaire object (if you do not need them, nothing has to be changed in the template).

Version 6 (since 3.6.0)

- Integration item template replaced item name. In templates you probably need to rename for integrations the property `itemUrl` to `responseItemUrl`.

Version 5 (since 3.5.0)

- All KM entities has now annotations (key-value dictionary). If you do not want to use those in your template, no changes are required.

Version 4 (since 3.2.0)

- Levels are renamed into phases and are using UUIDs. Phases are as part of the KM in `knowledgeModel.entities` of the context.
- Metrics are now also identified by UUID and part of the KM.

Version 3 (since 2.12.0)

- Additional metadata about each replies has been added and structure of reply is changed (extra `.value` needed). In case you are using filters such as `reply_str_value` no changes are needed.

- For integration reply, the type values are renamed `IntegrationValue` -> `IntegrationType` and `PlainValue` -> `PlainType` for consistency.

Version 2 (since 2.6.0)

- Changed `questionnaireReplies` to use path-reply map and removed then redundant `questionnaireRepliesMap` from document context.

- Replies for list question represented as list of UUIDs instead of size used for numeric indexing.

Version 1 (since 2.5.0)

- Initial version of metamodel, introduced in DSW 2.5.0 as start of versioning.

1.9.3 Integration Questions

DSW can be integrated with other services using so called *integration question*. The answer to that type of question does not contain only the answer itself but also a link to that external resource (which can be done, for example, using a persistent identifier). Therefore, these answers help clearly understand what the researchers use and promotes interoperability.

Examples of such integrations that are used within Common DSW Knowledge Model is FAIRsharing or ROR.

There are two ways of how we can connect DSW to these services:

- **API** - using an API provided by the external service to search for the results
- **Widget** - using a specialized widget implemented for the connection with the DSW

Integration Question - API

Integration question can be connected to an external resource using its API. We can then search for the results from the external service using the DSW questionnaire interface. When we select an answer it is not only the text (such as a name of the database), but also a link to the external service to the selected item. The whole flow is denoted in the following diagram.

External Service Requirements

If we want to connect an external service using the API there are certain requirements for it to make the connection to DSW possible.

- **Allows search using free text**

  - There must be a way to send a search phrase to the API so that it can filter the results based on it

- **Returns a JSON response with a list of results**

  - The response must be JSON so DSW can parse it

  - There needs to be a JSON list where all the items matching the search query are

Fig. 82: How integration question connected to, for example, FAIRsharing API works.
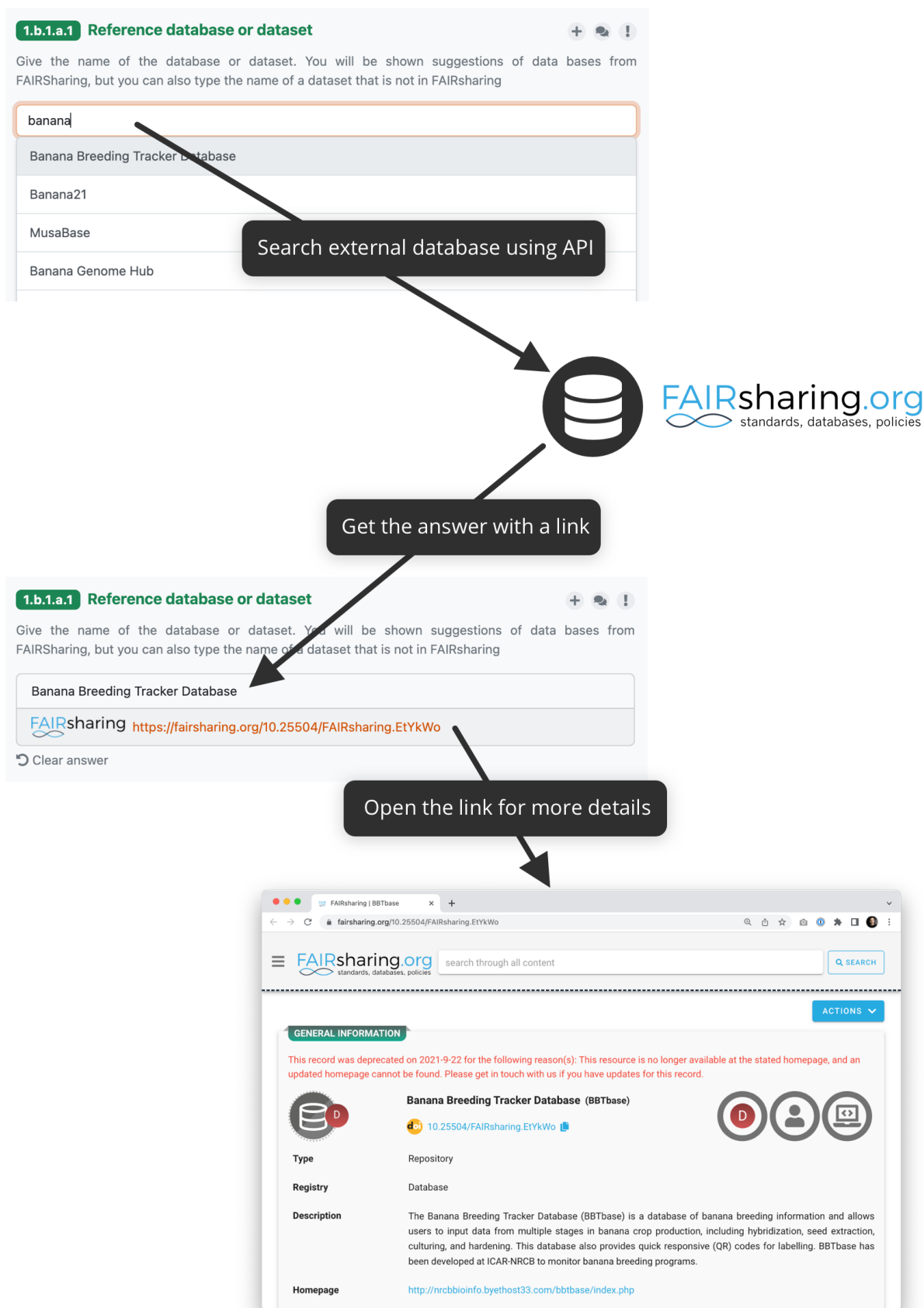
- **It is possible to construct a link to the selected item**

  - We also need to be able to construct the link to the item from the data we get in the response so we can provide it with the answer

Configuration

The configuration is done in the *knowledge model editor*. First of all, we need to create a new integration and choose its **Type** to be **API**. Then, there are some metadata, such as **ID**, **Name**, or **Logo URL**.

Request Configuration

In the **Request** section, we configure how to make an HTTP requrests to the external service's API. For that, we need to configure the following (the specific values depends on how the API works):

- **Request URL** - what is the URL where we want to send search requests

- **Request HTTP Method** - what HTTP method should be used

- **Request HTTP Headers** - some headers might be needed, such as `Accept: application/json` to have a correct response type

- **Request HTTP Body** - if we need to send some HTTP body

- **Allow Empty Search** - some APIs don't work if we try to search with an empty string, turn this of it's the case

There is a special property `${q}` that we can use within those fields. The property represents the string that users type to the questionnaire. So for example, we can write **Request URL** as:

```
http://example.com/api/search?q=${q}
```

Response Configuration

In the **Response** section, we configure how to process the JSON respnonse from the external service. For that, we need to configure the following:

- **Response List Field** - where in the JSON response is the list of items corresponding to the search query

- **Respone Item ID** - what field represents an item ID in the returned JSON

- **Response Item Template** - how we want to present the result to the user

We can use Jinja2 templates (Ginger implementation) in Response Item ID and especially in Response Item Template to make the response item look better.

Secrets and Other Properties

Sometimes, we might need to use some secrets (for example for authentication token), additional properties (such as API URL if we want to use different one for testing and production), or basically any information that we do not want to include in the knowledge model. In that case, we can define some properties in the instance settings.

We need to navigate to *Administration → Settings → Knowledge Models* and there is a field called **Integration Config**. It is a YAML organized by the **Integration ID** at the top level and key value pairs for each property.

We can fill some propertes in. So, for example, if the **Integration ID** of our integration is *ourIntegration* we can write:

```
ourIntegration:
    authorizationToken: "abcd"
    apiUrl: "http://example.com/api"
```

Then, in the configuration of our integration, we can use these properties in the request configuration, so for example the **Request URL** can be:

```
${apiUrl}/search?q=${q}
```

And we can add a header such as:

```
Authorization: Bearer ${authorizationToken}
```

Note: These properties can be accessed only from the integration with matching ID.

Video Tutorial

We have the following video tutorial showing how to set up the integration question using API.

https://youtu.be/x-kx6ppVBo0

External Resources

- How to Configure Integration Question in FAIR Wizard
- How to Improve Integration Question Item Template in FAIR Wizard
- Ginger Documentation

Integration Question - Widget

Integration question can be connected to an external resource using a widget integration. When there is this type of question, instead of writing an answer, reserachers click on *Select* button. It will open the widget where they can pick their answer and it is then sent back to the DSW. The whole flow is denoted in the following diagram.

Configuration

The configuration is done in the *knowledge model editor*. First of all, we need to create a new integration and choose its **Type** to be **Widget**. Then, there are some metadata, such as **ID**, **Name**, or **Logo URL**, as well as the **Widget URL** which is the URL where the widget is deployed. The URL will be open in popup window when researchers click on the *Select* button when filling in the questionnaire.
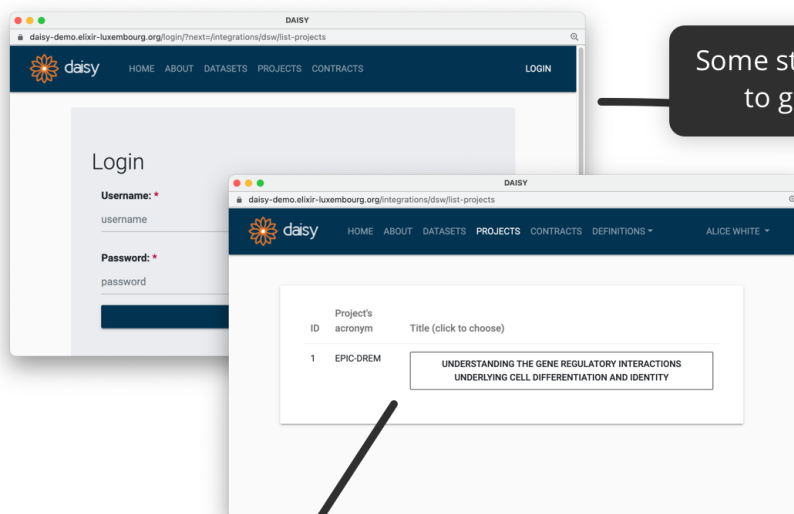
Fig. 83: How integration widget connected to, for example, DAISY works.

Implementation

The actual implementation is done using DSW Integration SDK. We recommend reading the readme and explore the examples to understand how it works.

1.9.4 Project Importers Development

> **Warning:** Project importers are an experimental feature.

Project importers are can be used to import the data from an external resource to DSW questionnaire. The importer creates the replies based on the data, therefore it needs to know the structure of the knowledge model it is compatible with.

We can implement a project importer using DSW Integration SDK. It is a JavaScript library we can import and use its API for the communication with DSW. The installation and usage is described in the SDK readme.

Example Importers

There are some importers already implemented. They are a good resource to see how to use the SDK:

- DSW Replies Importer
- DSW maDMP Importer

1.9.5 Submission Service

As administrators, we can configure submission services using *Document Submission Settings*. The configured HTTP request is then used when a user clicks *Submit* for an allowed document for submission and selected the desired submission service. The document is sent as a body of the request (or as multipart, based on the configuration) to the external service that should process it and return HTTP response with status code, and possibly also the *Location* header and some textual message.

Usually, we will need a simple proxy service to be developed that will accommodate this to API of some information system, database, storage, or other service. For example, such a proxy service will be able to receive the JSON documents from DSW, retrieve additional information through DSW API as needed, transform it to some other resulting artifact and store it in some local database that is used by other systems.

Example Submission Services

There are some submission services already implemented and can be used to check the implementation possibilities:

- Dummy Submission Service which just based on the headers returns example result or error.
- Email Submission Service sends an email through SMTP connection with the submitted document attached (or processed in case of JSON).
- Nanopub Submission Service allows to store a nanopublication (in RDF TRiG format) in the distributed network of nanopublication servers.

1.9.6 Contributing

Interested in contributing to the DSW development?

Ideas

If you have some idea (feature request) how to extend the DS Wizard, you can add it to our Ideas website or you can send us an email at support@ds-wizard.org.

Reporting

Bugs

In case you find some bug, please create an issue and provide requested information or contact us via email support@ds-wizard.org.

Vulnerabilities

If you find an security issue within DSW, please create appropriate issue. However, never include sensitive information in the issue as it is publicly available. Such information (e.g. logs) send to us via e-mail support@ds-wizard.org.

Development

Our projects are open source and you can contribute via GitHub (fork and pull request):

- https://github.com/ds-wizard
- https://github.com/ds-wizard/engine-backend
- https://github.com/ds-wizard/engine-frontend
- https://github.com/ds-wizard/document-worker
- https://github.com/ds-wizard/dsw-tdk

Note: Carefully read README and CONTRIBUTING files (if present) and also try to contact the main developer of the project for further details. You should follow the same code style, be DRY, and fit our overall architectures and structuring.

Test Policy

Testing is essential to ensure the successful construction and implementation of DSW. It is necessary to keep tests updated together with new features and other changes in the code.

Each component may use its own test suite (unit tests, acceptance tests, integration tests), which shall be described in the CONTRIBUTING file within the corresponding repository. To test all components together, we have the E2E test suite (Cypress) that tests according to various use cases, i.e., what can a user do within DSW using its web user interface.

Whenever a new feature is developed, it must be covered by tests. For the E2E test suite, a specific sub-task is created in our JIRA when applicable. All components must pass all tests before releasing (including release candidate versions).

The release candidate versions are tested with OWASP ZAP. Eventual found vulnerabilities related to the code or dependencies (not deployment) are solved prior to the release or listed in *Vulnerabilities* if not possible to solve for the release.

In case that a bug is found, it must be analyzed why tests did not find it. If possible and needed, test suites are extended to cover these (and similar) bugs.

1.9.7 Vulnerabilities

All known vulnerabilities are listed here so user's can be aware of them and possibly avoid them.

Reporting

Vulnerabilities should be reported using issues. To submit a private report, please send it to us via email: support@ds-wizard.org. Vulnerability issues are the top priority and resolved in the shortest time possible:

1. Accept vulnerability issue report (GitHub or email)
2. Verify and reproduce the issue, classify severity
3. Publish as known vulnerability
4. Design solution
5. Implement the change and release a hotfix
6. Move to solved vulnerabilities

Known Vulnerabilities

No vulnerabilities has yet been found or reported.

Solved Vulnerabilities

No vulnerabilities has yet been found or reported.

Basic Hints for Security

- Change or remove default users
- Adjust default role after registrations based on your needs
- Provide DSW through proxy with HTTPS (both client and server application)
- Backup data regularly (e.g. daily)
- Use secured SMTP (SSL)
- Use strong passwords, esp. for administrator accounts

1.10 Miscellaneous

Additional information related to DSW that might be useful.

1.10.1 DSW Registry

DSW Registry is a place where we publish knowledge models, document templates and locales. It is very easy to get those into a DSW instance and use.

DSW Registry	Knowledge Models Document Templates Locales		Log In Sign Up About

Common DSW Knowledge Model   Data Stewardship Wizard
DSW Knowledge Model originating from mindmap made by Rob Hooft

FAIR Convergence Matrix Questionnaire   GO FAIR
Questionnaire prompting communities to explicitly declare their FAIR Implementation Profiles

Life Sciences DSW Knowledge Model   Data Stewardship Wizard
Life Sciences customization of DSW Knowledge Model

RNAct ESR Training KM   RNAct
Teach early-stage researchers (ESRs) about DMPs

Fig. 84: DSW Registry with a list of knowledge models.

We first need to connect our DSW instance to the DSW Registry in *DSW Registry Settings*. Once we have that, we can:

- *Import knowledge models from DSW Registry*

- *Import document templates from DSW Registry*

- *Import locales from DSW Registry*

1.10.2 Markdown Cheatsheet

Various text fields in DSW can be formatted by using Markdown formatting language. Here you can get a basic overview of what can be achieved with Markdown.

Basic Syntax

These are the basic Markdown elements supported by all applications.

Extended Syntax

These elements extending the basic syntax are supported in DSW.

1.10.3 Help

Still need help? Or you think that some topic is missing? Do not hesitate to contact us!

The best way to reach us is by sending an email to info@ds-wizard.org.

In case you found a bug, vulnerability, or have a generic question related directly to DSW tool – please create a GitHub issue and also visit *Contributing* section.

For feature requests, we recommend using the Ideas page.