Data Stewardship Wizard

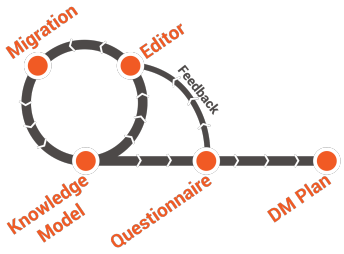
Release 2.2.0

DSW Team

Apr 03, 2020

USER DOCUMENTATION

The Data Stewardship Wizard is an international project to help serious researchers and data stewards with building smart **Data Management Plans for FAIR Open Science**.



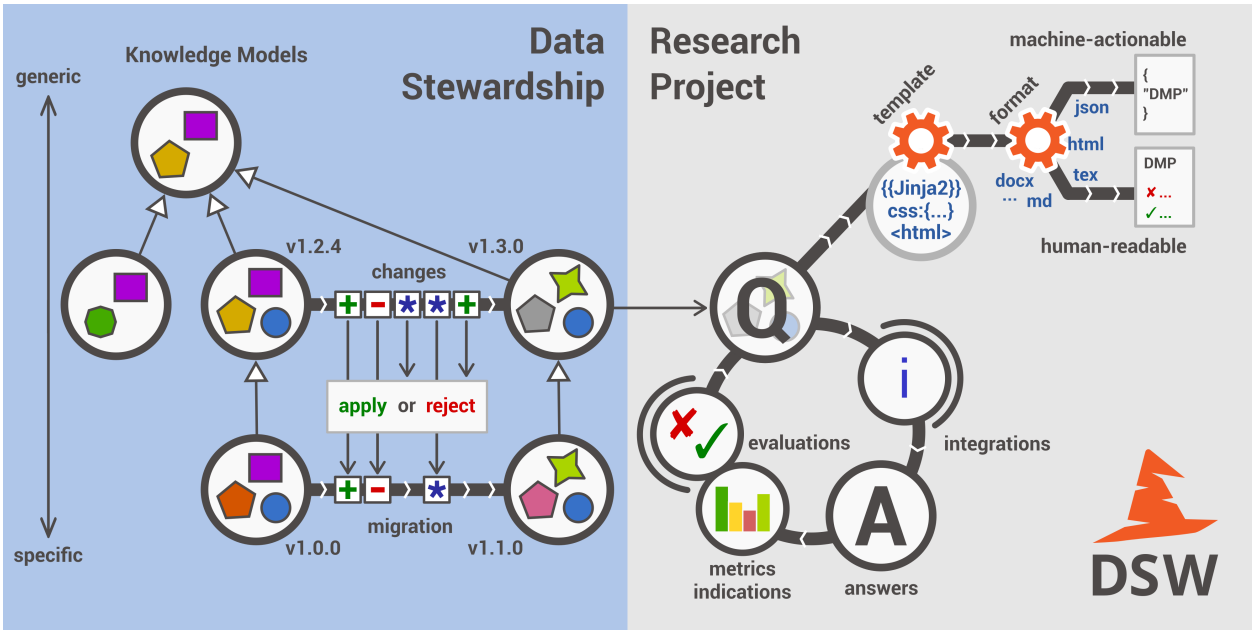
CHAPTER

ONE

ABOUT DSW

1.1 What is DSW?

Data Stewardship Wizard is a joint ELIXIR CZ and ELIXIR NL project bringing a simple but powerful solution for researchers to help them understand what is needed for good, FAIR-oriented Data Stewardship, to find ELIXIR experts to help out, and to build their own Data Management Plans. The DS Wizard can also function as a check list for data management professionals, like the checklists used by pilots before each flight.



1.1.1 FAIR

The main driver for the DSW is now to offer a convenient helpful tool for data stewards and researchers. Given a limited funding, we focus on this mission now. Howevever, from a long-term perspective, the richness of knowledge contained in the Wizard definitively calls for being FAIR. On this page we track the progress of compliance with the FAIR principles.

1.1.2 Machine-Actionable DMPs

We are part of the initiative #activeDMPs. Here, we will post updates on concrete steps, mostly with the respect to the identified use cases. The work on this front will continue according to our available capacity and funding.

1.2 Terminology

1.2.1 Concepts

Organization An infrastructure, institution or a similar body that runs its own copy of DS Wizard. Identified by Organization ID.

Knowledge Model (KM) An ordered collection of interlinked KM Items, from which a questionnaire is generated. Identified by a KM ID. May be customized and released as a package. Identified by a ID that consists of Organization ID, KM ID and Version. It can be exported/imported and further customized.

Questionnaire A representation of a KM in a shape of a form for filling-in.

Data Management Plan Exported filled questionnaire using selected template and format that should help researcher with data management in his/her project.

KM Root A package with no ancestor packages.

Customization of KM An ordered collection of changes of another parent knowledge model.

KM Item A chapter, question, answer, reference, expert, integration, tag, etc.

Change of KM Item Adding, editing, deleting of a knowledge item

Migration of KM Upgrade of a KM with a newer version of the parent KM

1.2.2 Modules

DS Wizard Our portal solution for Data Stewardship Planning. It contains *Questionnaire*, *KM Editor* and other parts for manamement of KMs and users.

Questionnaire A tool for interactive browsing and answering a questionnare.

KM Editor A tool for customization of a KM and its creation and publishing.

1.3 Types

The following diagram shows individual types in DSW and how they are connected.

1.4 Operations

The following diagram shows how the operations interact with the data types in the DSW.

1.4.1 Create Knowledge Model

It creates a new Knowledge Model Branch that contains no Knowledge Model Events.

1.4.2 Edit Knowledge Model

It creates a branch for a Knowledge Model Package that is owned by the organization of the current DSW instance. New versions can be published as the next version of the original Knowledge Model Package.

1.4.3 Fork Knowledge Model

It creates a branch for a Knowledge Model Package that is **not** owned by the organization of the current DSW instance. New versions are published as a new Knowledge Model Package.

1.4.4 Publish Knowledge Model

It creates a new version of the Knowledge Model Package from the new events added to the Knowledge Model Branch.

1.4.5 Import Knowledge Model

Knowledge Model Bundle file from outside of the DSW instance can be imported. The Knowledge Model Packages contained in the bundle can be then used in DSW.

1.4.6 Export Knowledge Model

Knowledge Model Package can be exported out of the DSW instance in the form of Knowledge Model Bundle file (with all its dependencies included).

1.4.7 Editor Action

Using the Knowledge Model editor, Knowledge Model can be extended, and events are generated.

1.4.8 Save Branch

Saving the branch simply saves all the events generated while using the editor.

1.4.9 Create Knowledge Model Migration

User can select a new parent Knowledge Model for a Knowledge Model Branch that is outdated and create a Knowledge Model Migration.

1.4.10 Process Knowledge Model Migration Change

During the Knowledge Model migration, each change form the parent Knowledge Model is reviewed and either applied or rejected to the localization Knowledge Model.

1.4.11 Finalize Knowledge Model Migration

After all the changes are applied or rejected, the migration can be finalized, and the new version of the Knowledge Model Package can be published.

1.4.12 Preview Questionnaire

Users can preview a Questionnaire while working in Knowledge Model Editor on a Knowledge Model Branch so that they can be sure with the result before publishing the version.

1.4.13 Create Questionnaire

A Questionnaire can be created from a specific version of a Knowledge Model Package.

1.4.14 Fill Questionnaire

User can fill the answer in the Questionnaire and save them. These are saved in the form of Questionnaire Replies.

1.4.15 Create Questionnaire Migration

Questionnaire Migration is created by choosing a new version of the Knowledge Model Package for a Questionnaire.

1.4.16 Process Questionnaire Migration Change

Users go through each change affecting their answers in the Questionnaire.

1.4.17 Finalize Questionnaire Migration

Once all the changes are resolved, the migration can be finalized, and a new Questionnaire is created.

1.4.18 Generate Document

A Document output can be generated for each Questionnaire using a Document Template.

1.5 Learn More

- DS Wizard (landing page)
- DS Wizard Resources
- DS Wizard Media Kit
- Leaflet
- Diagrams

CHAPTER

TWO

USAGE

Usage is role-dependent and there are three roles of authorized users: *Researcher*, *Data Steward*, and *Administrator*. Higher-level role can do the same as lower-level role and also something more, so let's describe it from the simplest one...

2.1 Anonymous

User of DSW that is not logged in yet is able to register, log in, or recover forgotten password via confirmed email address.

If configured, anonymous user can test out *Questionnaire Demo* (next to *Log In* and *Sign Up* buttons in the navigation bar. That is an example of our **Questionnaire** without any ability to save or export the answers, only to try out and learn. This functionality is always accessible in our public instance.

2.2 Researcher

Researcher is a user who works on a scientific project and has the knowledge about the specific project. His/her goal is to get high-quality FAIR Data Management Plan and learn how to work with data in the project.

2.2.1 Questionnaire

Data Stewardship Wizard provides simple way to create questionnaire from KM package and fill it in smart way - only relevant questions for your case will be shown.

- *Create* = create new questionnaire from certain version of certain KM package
- For each questionnaire:
  - *Fill questionnaire* = starts interactive questionnaire that guides you through answering relevant questions for specific project
  - *Edit* = edit questionnaire details such as name or privacy
  - *Create Document* = create a DMP in selected format and template
  - *View Documents* = view created DMPs for the questionnaire
  - *Clone* = create a duplicate quetionnaire
  - *Create Migration* = migrate questionnaire to a different version of KM
  - *Delete*

2.2.2 Documents

This section displays all documents (or DMPs) that the user is allowed to see. You can download the document, delete it, or navigate to questionnaire used for the DMP.

2.3 Data Steward

Data Steward is a user who has good knowledge of *DS domain* (how to deal with data) and puts this knowledge into a **knowledge model**. The knowledge model is then used by scientists to create the DMP with **Questionnaire**.

2.3.1 Knowledge Models

Knowledge Models are collections of DS knowledge. Each package has own unique identifier consisting of organization ID and km ID (and then also version). It stores all the knowledge units = changes of "zero" knowledge (add, delete, edit - chapter, question, answer, reference, etc.).

- *Import* = import new KM package or new version of KM package from file or from *Registry*
- For each KM package:
  - *Delete*
  - *View detail* = shows detail with versions and basic information about the KM and for each version:
    - *Export* = export specific version of KM, that can be then imported (e.g. in different instance of DSW)
    - *Create KM Eitor* = shortcut to create editor from specific version of KM
    - *Create Questionnaire* = shortcut to create planner from specific version of KM
    - *Delete*

2.3.2 KM Editor

Knowledge Model Editor allows to create new knowledge models:

1. from scratch (i.e. totally new root KM package)
2. as new version of existing KM package (i.e. some improvements needs to be done)
3. as a customization of an existing KM package (i.e., extension for specific subdomain - can be based on organizational, geographical, legal or other expertise)

- *Create* = create editor with specific name and KM ID, optionally based on some parent KM.
- For each KM editor:
  - *Open Editor* = shows editor that allows to go through the all parts of KM, create new parts, edit or delete them.
  - *Delete*
  - *Publish* (*if changes are made*) = create KM with specific version and description of changes
  - *Upgrade* (*if newer version of parent KM*) = migrate to newer version of parent KM in interactive migration tool

2.4 Administrator

Administrator manages overall settings of the Data Stewardship Wizard instance and has the highest privileges.

2.4.1 Organization

Administrator can set two things in organization settings:

- **Organization name** = visible name of the organization that uses DSW instance
- **Organization ID** = unique identifier of the organization, it is then used in identifier of created Knowledge Models

2.4.2 Users

User management is also quite simple. Administrator can see table with registered users, *Delete* or *Edit* single one of them, or *Create User* directly. When editing the user, it is possible to change all the attributes from registration and also manually change the "Active" status.

CHAPTER

THREE

KM EDITOR TUTORIAL

Note: This tutorial has been made for previous version of DSW and may differ in some details.

The KM (Knowledge Model) Editor is used for creating and updating knowledge models. This tutorial will go through the steps of creating a new KM from scratch.

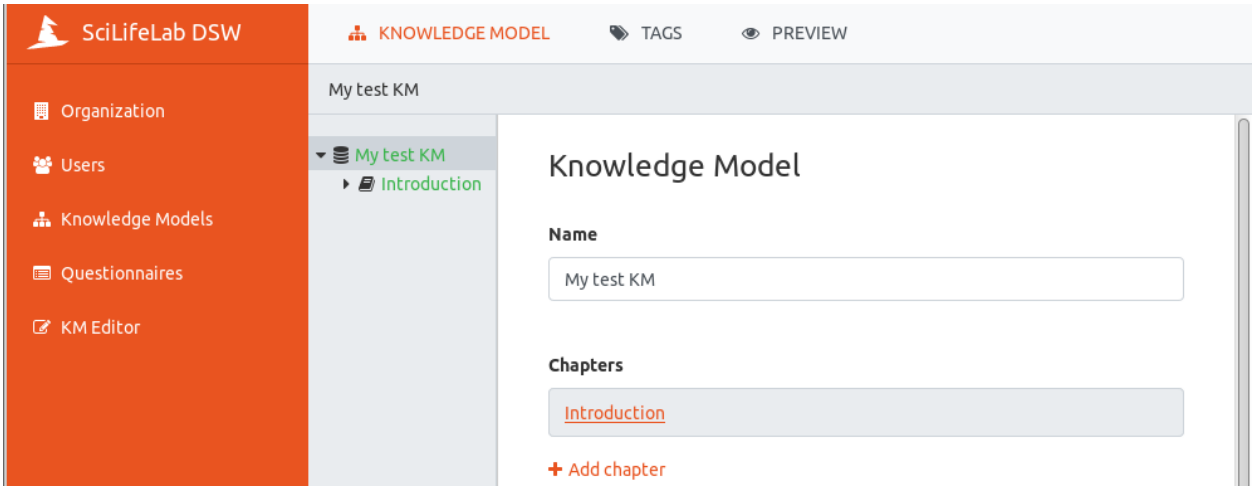
1. Click on "KM Editor" in the left-hand menu of the wizard
2. Click "Create" on the top to the right
3. Fill the fields with appropriate **Name** (e.g. 'KM Editor Test Model') and **Knowledge Model ID** (e.g. 'test-model'). The ID can contain alphanumeric characters and dash but cannot start or end with dash. Leave Parent Knowledge model as is (–). *Note:* If you want to update or reuse an existing model, this is where you select which one.
4. Click on "Save" button

3.1 Create a chapter

Once you have created a new KM, it's time to fill it with chapters (i.e. sections; think of them as headers of your model).

1. Write a name of the KM (e.g. 'My test KM')
2. Press + Add chapter
3. Write a **Title** (e.g. 'Introduction') and **Text** (e.g. 'Background information') for the chapter. *Note:* The **Text** field is where you add a description of what the chapter will contain questions about. There are two tabs, 'Editor' and 'Preview', since you can use Markdown (https://www.markdownguide.org/basic-syntax/) to format the text and check the result in 'Preview'.
4. Press +Add question in order to create a question

Notice the grey area which gives an overview of your KM. Use this to navigate between the different parts. Whenever you want to see the result of an addition, click on another part in the KM overview (at this stage this would be the "parent" we named 'My test KM'.



3.2 Add a question

Questions can be of different types:

- Options
- List of items
- Value

Let's create some questions of each type to demonstrate:

Value

1. Select *Value* as **Question Type**
2. Write 'Project title' in the **Title** field
3. In the **Text** field, write 'Please enter the title of your project', as instructive text
4. Use **When does this question become desirable?** to indicate in which phase of the project a question should be answered, e.g. *Before submitting the proposal*.
5. **Value type** can be *String*, *Date*, *Number* or *Text*, select *Text*

List of items

1. Click on Introduction in the grey overview area of the editor
2. Press +Add question in order to create a new question
3. Select *List of items* as **Question Type**
4. Write 'Project members' as **Title**
5. Write 'Please specify the researchers participating in the project' as **Text**
6. Select *Before submitting the DMP* as **When does this question become desirable?**
7. In the **Item Template**, click on +Add question
   a. Set the **Question type** to *Value*
   b. Write 'Name' as **Title**
   c. Back to one level up by clicking on Project members in the grey overview part of the editor

d. Scroll down and click on +Add question in the **Item Template**

e. Set the **Question type** to *Value*

f. Write 'Email' as **Title**

Options

1. Click on Introduction in the grey overview area of the editor
2. Press +Add question in order to create a new question
3. Select *Options* as **Question Type**
4. Write 'Research field' as **Title**
5. Write 'Please select the research field for this project' as **Text**
6. Click on +Add answer
7. Write 'Life science' as **Label**
8. Write 'Your project is likely going to produce a lot of data, a full data management plan will prepare you for the various challenges this will entail' as **Advice**
9. Click on Research field in the grey overview area, scroll down to **Answers** and click on +Add answer
10. Write 'Other' as **Label**
11. Scroll down to **Follow-up Questions** and click on +Add follow-upp question
12. Select *Value* as **Question Type**
13. Write 'Which other research field?' as **Title**
14. As **Value Type**, select *Text*

3.3 Save

Whenever there are changes not saved in the KM, the clickable options "Discard" and "Save" is visible in the top row of the wizard. Click on "Save".

Notice that this automatically lead you to the top level of the KM Editor, outside your KM. If you position the mouse on your KM ('Test of KM Editor') you see the options of Open Editor, Publish and Delete. Click on Open Editor in order to continue editing the KM.

3.4 Add Reference and Expert

All question types have the possibility of adding references and experts, to be used for adding additional information and people to contact in order to get help, respectively. Let's add one of each:

1. Click on the arrow at Research field in the grey overview area, and scroll down to **References**
2. Click on +Add reference
3. Write "https://en.wikipedia.org/wiki/List_of_life_sciences" as **URL**
4. Write "List of Life sciences" as **Label**
5. Click on Research field again and then click on +Add reference
6. Write "My science for life guru" as **Name**

7. Write "help@scilifeguru.com" as (the fake) **Email**
8. Click on "Save" in top right corner, position the mouse on your KM ('Test of KM Editor'), and click on Open Editor

3.5 Change order of questions

It is possible to rearrange the order of questions, if they are on the same 'level' (but not between levels at the moment of writing this tutorial). Let's try:

1. Click on Introduction and scroll down to **Questions**. The three questions created are all on the same level.
2. Position the mouse on the grey area next to Research field, hold down the left button of the mouse and drag-and-drop above Project title

If and when the need arise to move a question to another level/group of questions, there are currently two alternatives:

- Rewrite the question in the new position, or
- *Export* the knowledge model and edit the resulting json file in an appropriate editor, and then import it again

3.6 Preview

In order to see what the resulting questionnaire will look like, you can click on "PREVIEW" on top row of the wizard. When you are finished just klick on KNOWLEDGE MODEL on the rop row, to return to editing mode.

3.7 Tags

Also on the top row of the wizard, is the "TAGS" viewer function. Tags can be used to mark questions of interest to certain stakeholders, when only a subset of the questions are of interest. We have not defined any tags yet, so let's do that:

1. Click on 'My test KM' in the grey overview area, and scroll down to **Tags**
2. Click on + Add tag
3. Write 'Library' as **Name**
4. Pick a color by clicking on one of the colored squares
5. Expand Introduction and click on Project title in the grey overview area
6. Scroll down to **Tags** and mark this question as of interest to the library by clicking the check box
7. Do the same for Research field
8. Click on "PREVIEW" and notice that all questions are visible. Select the Library tag, by clicking in it's check box. Now only the two questions *Project title* and *Research field* are visible.

3.8 Publish

When you are happy with the content and look of your KM, it is time to make it available for people to use it (either as a start for their own KM or for users to fill it out in form of so called Questionnaires):

1. Click on "KM Editor" in the leftside menu
2. Position the mouse on your KM ('Test of KM Editor') and click on Publish among the alternatives that becomes visible
3. Add a version number in **New version** (e.g. '1.0.0')
4. Write a **Description** (e.g. 'This is the root version')
5. Click on "Publish"

3.9 Export

A knowledge model kan be exported into a json formatted file:

1. Click on "Knowledge Models" in the left hand menu
2. Position your mouse on the KM ('Test of KM Editor') and click on View detail
3. Click on Export and save the file on your computer.

3.10 Import

A knowledge model can also be imported:

1. Click on "Knowledge Models" in the left hand menu
2. Click on "Import" in top-right corner
3. Click on "Choose file", select the .km-file
4. Click on "Upload"

Note: If the KM already exists in the wizard it needs to be deleted first, both in the "KM Editor" (first) and in "Knowledge Models"

CHAPTER

FOUR

INTEGRATIONS

The DS Wizard support integrations of external services with API responding with JSON result containing a list of items. Such a list can be used for type hints in special type of questions - Integration question.

4.1 Integration

You can create a new integration in the root of any KM by clicking on **Add integration** in **Integrations** section. A form that might seem to be a bit complicated on the first sight will appear. But filling it up is very simple. All you need to know is how the API of what you want to integrate works. Every API should have its documentation that you can use and we recommend also to try it with Postman or curl.

- **Id** = internal identifier of integration used in the Wizard, not visible in questionnaire
- **Name** = name of the integration, typically name of the integrated service
- **Logo** = BASE64 encoded logo of the service (we recommend PNG with width around 100px, there are services that can encode the logo from file for you, for example, Base64 Image Encoder
- **Props** = variables for building the request specific to integration questions (for example, `filter` when you want to filter the results based on some attributes specific to a question and the API allows that)
- **Item URL** = URL that will be displayed when item from integration is selected, it should lead the user to more information about that item, you can use `${id}` variable to build the URL
- Request
  - **Request Method** = HTTP method used to request result from API (typically `GET`)
  - **Request URL** = target of the request, you should use `${q}` variable that contains what the user fills in the field (to search), then you may use variables specified in **Props** (e.g. `${filter}`) and also variables from the configuration file
  - **Request Headers** = various headers specified by API (usually `Accept: application/json` or some authentication header)
  - **Request Body** = content of the request (for `GET` it should be empty)
- Response
  - **Response List Field** = path in the JSON response to list of results (dot notation can be used to navigate in complex structures)
  - **Response Id Field** = name of an identifier field in the item of results list (dot notation allowed), used then in `${id}` variable
  - **Response Name Field** = name of a name field in the item of results list (dot notation allowed), used as possible answer visible to the user

Dot notation is used to navigate in more complex JSON structures when the list of items or attributes of items that needs to be extracted are hidden inside. For example you would use `result.list` together with `attributes.id` and `attributes.data.name` in case of:

```
{
  "result": {
    "list": [
      { "attributes": { "id": "id01",
                        "data": { "name": "Foo",
                                  //...
                                }
                      }
      }, //...
    ], //...
  }, //...
}
```

4.2 Configuration file

There is a special configuration file for integrations (typically `integrations.yml`, more in documentation of *Integration* configuation) that you can use to keep some settings out of the KM (e.g. URL of the API endpoint or API key that you don't want to share). The file can contain for each integration (using its **Id**) a list of key-values.

Danger: We highly recommend to store sensitive data such as API keys in the configuration file raher than directly as text in the knowledge model that can be exported to file and distributed easily by mistake.

4.3 Integration question

When you have some integration(s) configured, you can create questions of type **Integration**, then you have to select which integration should be used and if has any **Props** defined, you can fill them as well. Using the **Preview** functionality, you can easily verify if it works. If *Unable to get type hints* error appears, your configuration is not correct (looking at the server log can be helpful if you have access to it).

4.4 FAIRsharing.org

We recognize FAIRsharing.org as high-quality service for curated, informative and educational resource on data and metadata **standards**, inter-related to **databases** and data **policies**. It is recommended to use FAIRsharing integration (for more, see *Integrations*) when need a hints for **standards**, **databases**, **collections**, and **policies**. In case that requested item is not present there, use add/claim content rather than another integration service that does not provide such level of confidency.

CHAPTER

FIVE

INSTALLATION

Attention: If you've deployed a local instance of the Wizard (Docker or build from source), we kindly request you to fill out this DS Wizard instance registration.

5.1 Public Instances

The application is currently deployed on a server provided by CESNET. Here are the addresses of running applications:

- Landing page with additional information
- Demo instance (free to use, for trying out all the features, unstable)
- Researchers instance (free to use, to build own DMPs, prepared for serious work)

Tip: You are free to register and test out the Wizard within the ds-wizard.org. Then you can decide if you want a local instance for you or your organization. Eventually you can contact us about *DSW Cloud* service where we can host and maintain your DSW instance, see our Get Started page.

5.2 Via Docker

The simplest way is to use Docker Compose. Requirements are just to have Docker installed, privileges for current user and the Docker daemon started.

1. Create a folder (e.g., `/dsw`, all commands in this manual are from this working directory)
2. Prepare all config files described in *Configuration* (especially `application.yml`), paths are visible from the *docker-compose.yml*
3. Copy (and adjust) *docker-compose.yml* provided below
4. Run the DSW with Docker compose `docker-compose up -d`
5. After starting up, you will be able to open the Wizard in your browser on http://localhost
6. You can use `docker-compose logs` to see the logs and `docker-compose down` to stop all the services

Listing 1: docker-compose.yml

```yaml
version: '3'
services:

  server:
    image: datastewardshipwizard/wizard-server
    restart: always
    ports:
      - 3000:3000
    volumes:
      - /dsw/server/application.yml:/application/engine-wizard/config/application.yml:ro
      - /dsw/server/integration.yml:/application/engine-wizard/config/integration.yml:ro
      - /dsw/templates/dmp:/application/engine-wizard/templates/dmp:ro

  client:
    image: datastewardshipwizard/wizard-client
    restart: always
    ports:
      - 80:80
    environment:
      API_URL: http://localhost:3000

  docworker:
    image: datastewardshipwizard/document-worker
    restart: always
    volumes:
      - /dsw/docworker/config.cfg:/app/config.cfg:ro
      - /dsw/templates/dmp:/app/templates:ro

  rabbitmq:
    image: rabbitmq:3.8.2-management
    restart: always
    environment:
      RABBITMQ_DEFAULT_USER: rabbit
      RABBITMQ_DEFAULT_PASS: password

  mongo:
    image: mongo:4.2.3
    restart: always
    ports:
      - 27017:27017
    volumes:
      - /dsw/data:/data/db
    command: mongod
```

Tip: You can take a look at https://github.com/ds-wizard/dsw-deployment-example

5.3 Locally without Docker

We highly recommend using Docker, but you are open to compile and run everything directly on your device. It is tested on Ubuntu 16.04 and you might encounter problems when using other plaforms and Linux distributions.

5.3.1 General Requirements

- The Haskell tool Stack (for server side)
- NPM (for client side)
- MongoDB (database, needs to be running)
- RabbitMQ (messaging, needs to be running)
- Python (>= 3.7, document worker)
- wkhtmltopdf (>= 0.12.5, for DMP exports in PDF)
- Pandoc (>= 2.6, for DMP exports in other formats aside HTML, PDF, and JSON)

5.3.2 Server

1. Get the server app (dsw-server)

   ```
   git clone https://github.com/ds-wizard/engine-backend.git
   ```

2. Copy and edit configuration (see *Configuration*)

   ```
   cp engine-wizard/config/application.yml.example engine-wizard/config/
   application.yml
   ```

3. Build (may take some time to download & build dependencies)

   ```
   stack build engine-wizard
   ```

4. Run (requires MongoDB according to configuration)

   ```
   stack exec engine-wizard
   ```

Note: Be aware that running engine-wizard requires its assets (e.g. `templates/` and `config/`) to be present in the working directory or where configured, e.g, *`templateFolder`*.

5.3.3 Client

1. Get the client app (dsw-client)

   ```
   git clone https://github.com/ds-wizard/engine-frontend.git
   ```

2. Install the app (dependencies)

   ```
   npm install
   ```

3. Change configuration if the server is not running on http://localhost:3000 (see *Configuration*)

4. Run the app

   ```
   npm run start:wizard
   ```

5. Open app in your favorite browser
6. For minified production-ready version, use

   ```
   npm run build:wizard
   ```

5.3.4 Document Worker

1. Get the document-worker

   ```
   git clone https://github.com/ds-wizard/document-worker.git
   ```

2. Install it using `pip` (or using `setup.py`)

   ```
   pip install .
   ```

3. Adjust configuration for your setup (see *Configuration*)
4. Run the app

   ```
   docworker config.cfg /path/to/templates
   ```

If you need to upgrade MongoDB version, follow the official instructions in their documentation.

5.4 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*)

You can use those accounts for testing or to initially made your own account admin and then delete them.

> **Danger:** Having public instance with default accounts is a **security risk**. Delete or change default accounts (mainly Albert Einstein) if your DSW instance is public as soon as possible.

5.5 Registry

When you have your own self-hosted instance, it is essential for you to register within the Registry service. It is source of shared knowledge models and can support your deployment. After registration of your organization with unique ID and email verification, you will get your **token**. This token is then used in *Settings*. Then your instance is connected automatically to the Registry service for specific functionality such as accessing shared knowledge models.

5.6 Other "Setups"

5.6.1 Initial Knowledge Model

When you have a fresh installation, there are just the default users and no knowledge models. You are free to create a new one from scratch if you want. Other option is to import existing KM `dsw:root:X.Y.Z` from the Registry. It is the core knowledge model for general data stewardship. The specific latest version (or other version that is the best for you) as well as other shared knowledge models can be found on the landing page of the Registry service. Other option is to import it from file if you have any (according to *Usage*)

5.6.2 Public Questionnaire

If you also need to enable public questionnaire for *Questionnaire demo* functionality with this core knowledge model, you have to download *public-package-root-2.0.0.json* file below and import it directly to the database into *publicPackages* collection. Optionally, you can move some of your packages similarly.

`public-package-root-2.0.1.json`

```
$ mongoimport --db dsw-server \
              --collection publicPackages \
              --file public-package-root-2.0.1.json
```

(If using Docker, you will need docker exec.)

Note: For public questionnaire correctly running, you need to import the related Knowledge Model in the Wizard otherwise you will end up with `Entity does not exist` error.

5.6.3 Database Backup

If you want to regularly backup your database (and you should!), all you need to do is to set-up simple script as cronjob:

Listing 2: dsw-backup.sh

```
#!/bin/bash
# Location of Mongo's data folder (Dockerized Mongo)
MONGO_DATA_DIR=/dsw/mongo/data
# - or your Mongo without using Docker
# - MONGO_DATA_DIR=/data/db
# Target for storing backups
TARGET_DIR=/var/backups/dsw
# Backup
BACKUP_FILE=$TARGET_DIR/backup_$(date +%d%m%y-%H%M).tgz
tar czf $BACKUP_FILE $MONGO_DATA_DIR
```

Make it executable (chmod a+x dsw-backup.sh) and add it as cronjob with crontab -e:

```
0 4 * * * /dsw/dsw-backup.sh
```

(This will do the backup every day at 4:00 AM. For more information, see crontab.guru.)

CHAPTER

SIX

CONFIGURATION

6.1 Settings

Settings are in-app administration interface for changing various behavior and look. An adminitrator can navigate to settings from the main (left) menu. *(since 2.2.0)*

It allows to configure (among others):

- toggle optional DSW features including registration
- external authentication using OpenID standard
- set up information texts and dashboard shown in the client (before login, after login, etc.)
- connection to registry
- feedback and support links and repository
- organization details (such as name, ID, and pre-configured affiliations)

6.1.1 OpenID

To configure OpenID service for external authentication, you need to have the following

- *Client ID* and *Client secret* for the service
- Root URL of the service (you can verify it that this returns valid JSON: `<URL>/.well-known/openid-configuration`

You can test the OpenID service using official OpenID Connect Playground

6.2 Configuration files

Files are used for setting the server-side configuration.

6.2.1 Server

Server configuration is done using YAML format. We provide examples directly in `config` folder of the repository https://github.com/ds-wizard/engine-backend. For tests there exist special versions suffixed by `-test`.

Build Info

A build configuration (`build-info.yml` file) contains information for the build of the application. This configuration can be created simply by running a prepared script `build-info.sh`. Normally this script is run by a CI Tool (Travis CI) during build process.

Important: If you build server app locally, you should also run `build-info.sh` otherwise your app will show bad build information and version.

Application

The mail configuration file that provide a lot of options and contains necessary settings for server to be running properly. You can see the example *application.yml* below. Next, all options are described in detail.

Listing 1: application.yml

```yaml
general:
  environment: Production
  serverPort: 3000
  secret: TteM1a0QkGe6ED5OvihPkA82LmwkWU6f
  clientUrl: http://localhost:8080
  serviceToken: NlQSNbGvh7EtcpinGnHE9g91
  integrationConfig: config/integration.yml
  templateFolder: templates

database:
  host: mongo
  databaseName: dsw-server
  port: 27017
  authEnabled: false
  username:
  password:

messaging:
  host: rabbitmq
  port: 5672
  username: rabbit
  password: password
  vhost: /

mail:
  enabled: false
  name:
  email:
  host:
  port:
  ssl:
  authEnabled:
```

(continues on next page)

(continued from previous page)

```
  username:
  password:

analytics:
  enabled: false
  email:
```

Section aside *General* may be omitted and default values will be used for whole sections (typically disabled, see below).

General

Configuration related to general operations of the server application.

`environment`

Type Enumeration [Production, Staging, Development, Test]

Default `Production`

Environment that your deployment is using. This affects which migrations are used and other minor things can be different in various environments.

`serverPort`

Type Integer [0-65535]

Default `3000`

Port that will be the web server listening on.

`secret`

Type String

Secret string of 32 characters for encrypting configuration in database and JWT tokens.

`clientUrl`

Type URI

Default `""` (empty)

Address of client application (e.g. `https://localhost:8080`).

`serviceToken`

Type String

Default `""` (empty)

Randomly generated string that matches configuration of *Feedback synchronization* component.

`integrationConfig`

Type String

Default `"engine-wizard/config/integration.yml"`

Filename or whole path of integration configuration file (see *Integration*). The path is relative to the working directory from where the Wizard runs.

`templateFolder`

Type String

Default `"engine-wizard/templates"`

Path to the folder where DMP and mail template are stored. The path is relative to the working directory from where the Wizard runs.

Database

Configuration of connection to MongoDB database.

`host`

Type String

Default `"mongo"`

Hostname or IP address of the server running MongoDB.

`port`

Type Integer [0-65535]

Default `27017`

Port that is used for MongoDB on the server (usually `27017`).

`databaseName`

Type String

Default `"dsw-server"`

Name of the database for DS Wizard within MongoDB.

`authEnabled`

Type Boolean

Default `false`

Whether authentication is enabled on MongoDB server and is required to connect to the database.

`username`

Type String

Default `""` (empty)

Username for authentication to database connection (if *`authEnabled`* is `true`).

`password`

Type String

Default `""` (empty)

Password for authentication to database connection (if *`authEnabled`* is `true`).

RabbitMQ

Configuration of connection to RabbitMQ.

host

Type String

Default `"rabbitmq"`

Hostname or IP address of the server running RabbitMQ.

port

Type Integer [0-65535]

Default `5672`

Port that is used for RabbitMQ on the server (usually `5672`).

username

Type String

Default `"guest"`

Username for authentication to RabbitMQ connection.

password

Type String

Default `"guest"`

Password for authentication to RabbitMQ connection.

vhost

Type String

Default `"/"`

Virtual host on RabbitMQ server (see RabbitMQ docs).

Mail

Configuration for sending emails (such as registration activation or for forgotten password mechanism) from the DS Wizard using SMTP connection.

Tip: You should enable emails unless you test the application for yourself only. It is used for email confirmations and mechanism of resetting password.

---

enabled

Type Boolean

Default `true`

If emails will be sent and SMTP configured.

name

Type String

Name of the DS Wizard instance that will be used as "senders name" in email headers.

email

Type String

Default "" (empty)

Email address from which the emails will be sent.

host

Type String

Default "" (empty)

Hostname or IP address of SMTP server.

port

Type Integer [0-65535]

Default `465`

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

ssl

Type Boolean

Default `false`

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

authEnabled

Type Boolean

Default `false`

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

username

Type String

Default "" (empty)

Username for the SMTP connection.

password

Type String

Default "" (empty)

Password for the SMTP connection.

Analytics

Configuration of analytic/informational emails for administrators, e.g., that a new user registered into the DS Wizard.

enabled

Type Boolean

Default `false`

If analytic emails should be sent.

email

Type String

Default `""` (empty)

Target email address where analytics to which will be sent.

Integration

Integrations in the DS Wizard are using external APIs and you might need some configured variables such as API keys or endpoints. For example, integration with ID `dbase` might use following configuration.

Listing 2: integration.yml

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

There can be multiple integrations configured in 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 *Integrations* 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.

6.2.2 Client

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

Listing 3: config.js

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

Client also provides wide variety of style customizations using SASS variables. Then you can mount it as volumes in case Docker as well:

```yaml
volumes:
  # mount SCSS file
  - /path/to/extras.scss:/src/scss/customizations/_extras.scss
  - /path/to/overrides.scss:/src/scss/customizations/_overrides.scss
  - /path/to/variables.scss:/src/scss/customizations/_variables.scss
  # mount other assets, you can then refere them from scss using '/assets/...'
  - /path/to/assets:/usr/share/nginx/html/assets
```

- `_extras.scss` = This file is loaded before all other styles. You can use it, for example, to define new styles or import fonts.
- `_overrides.scss` = This file is loaded after all other styles. You can use it to override existing styles.
- `_variables.scss` = A lot of values related to styles are defined as variables. The easiest way to customize the style is to define new values for these variables using this file.

For more information about variables and assets, visit Theming Bootstrap. The color of illustrations can be adjusted using `$illustrations-color` variable.

6.2.3 Document Worker

Configuration of document worker must match with server configuration.

Listing 4: docworker.cfg

```
[mongo]
host=mongo
port=27017
database=dsw-server
collection=documents
fs_collection=documentFs

[mq]
host=rabbitmq
port=5672
vhost=/
username=rabbit
password=password
queue=document.generation

[logging]
level=DEBUG

[pandoc]
executable=pandoc
args=--standalone
timeout=5

[wkhtmltopdf]
executable=wkhtmltopdf
args=
timeout=5
```

mongo

host

Type String

Default `"localhost"`

Hostname or IP address of the server running MongoDB.

port

Type Integer [0-65535]

Default `27017`

Port that is used for MongoDB on the server (usually `27017`).

username

Type String

Default `None` (optional)

Username for authentication to database connection (will be used if set).

password

Type String

Default `""` (optional)

Password for authentication to database connection (will be used if set).

database

Type String

Name of the database for DS Wizard within MongoDB.

collection

Type String

Name of the collection for documents (typically `documents`).

fs_collection

Type String

Name of the collection for files (typically `documentFs`).

auth_database

Type String

Default `<database>` (optional)

Authentication database used for MongoDB, defaults to the same value provided in `database` option.

fs_collection

Type auth_mechanism

Default `"SCRAM-SHA-256"` (optional)

Authentication mechanism used for MongoDB.

mq

host

Type String

Default `"localhost"`

Hostname or IP address of the server running RabbitMQ.

port

Type Integer [0-65535]

Default `5672`

Port that is used for RabbitMQ on the server (usually `5672`).

vhost

Type String

Default `"/"`

Virtual host on RabbitMQ server (see RabbitMQ docs).

username

Type String

Default `None` (optional)

Username for authentication to RabbitMQ connection (if any).

password

Type String

Default `None` (optional)

Password for authentication to RabbitMQ connection (if any).

queue

Type String

Name of queue used for passing document jobs (typically `document.generation`).

logging

level

Type String

Default `"WARNING"`

Name of logging level (use names of Python logging levels).

pandoc

executable

Type String

Default `"pandoc"`

Executable command for running Pandoc (can be aliased or defined with absolute path).

args

Type String

Default `"--standalone"`

Default arguments used for all Pandoc calls.

timeout

Type Float

Default `None` (optional)

Timeout for Pandoc subprocess call.

wkhtmltopdf

executable

Type String

Default `"pandoc"`

Executable command for running WkHtmlToPdf (can be aliased or defined with absolute path).

args

Type String

Default `""` (empty)

Default arguments used for all WkHtmlToPdf calls.

timeout

Type Float

Default `None` (optional)

Timeout for WkHtmlToPdf subprocess call.

6.3 Feedback synchronization

Because our feedback functionality is based on GitHub issues, it requires synchronization. If you are using our Docker image, all you have to do is define environment variables for the server image:

```
API_URL: "..."
SERVICE_TOKEN: "..."
ENABLE_CRON: "1"
ENABLE_CRON_FEEDBACK_SYNC: "1"
```

Locally, you need to set up cron job on your machine similarly. See our script for setting up the cron job.

6.4 DMP templates

You can freely customize and style templates of DMPs (filled questionnaires). 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. The location of templates root folder is configurable via *`templateFolder`*.

6.4.1 Template files

We provide currently basic `default` template (see here) but it is possible to get inpired and create more or edit it. The basic structure is following:

- `templates/dmp/my-template/template.json` = metadata about the template (must be named `template.json`)
- `templates/dmp/my-template/...` = other template files (and sub-directories)

Templates allow you to iterate through questions and answers and find what you need to compose some output. For example, you can generate longer text based on answers of various questions by knowing its texts or UUIDs. To the template, object `ctx` (document context) is injected and can be used as variable - for information about its structure, browse current default template or visit source code.

You can have multiple DMP templates and users will be able to pick one of them when exporting a filled questionnaire. Each template must have its metadata JSON file that contain random and unique UUID, name to be displayed when picking a template, and relative path to root file of the template:

Listing 5: template.json

```
{
  "uuid": "43a3fdd1-8535-42e0-81a7-5edbff296e65",
  "name": "Common DSW Template",
  "allowedKMs": [
      {
        "orgId": null,
        "kmId": null,
        "minVersion": null,
        "maxVersion": null
      }
  ],
  "formats": [
    {
      "uuid": "d3e98eb6-344d-481f-8e37-6a67b6cd1ad2",
      "name": "JSON Data",
      "icon": "far fa-file",
      "steps": [
        {
          "name": "json",
          "options": {}
        }
      ]
    },
    {
      "uuid": "a9293d08-59a4-4e6b-ae62-7a6a570b031c",
      "name": "HTML Document",
      "icon": "far fa-file-code",
      "steps": [
        {
          "name": "jinja",
          "options": {
            "template": "default.html.j2",
            "content-type": "text/html",
            "extension": "html"
          }
        }
      ]
    },
    {
      "uuid": "68c26e34-5e77-4e15-9bf7-06ff92582257",
      "name": "PDF Document",
      "icon": "far fa-file-pdf",
      "steps": [
        {
          "name": "jinja",
          "options": {
            "template": "default.html.j2",
```

(continues on next page)

(continued from previous page)

```
            "content-type": "text/html",
            "extension": "html"
          }
        },
        {
          "name": "wkhtmltopdf",
          "options": {
            "args": ""
          }
        }
      ]
    },
    {
      "uuid": "f4bd941a-dfbe-4226-a1fc-200fb5269311",
      "name": "MS Word Document",
      "icon": "far fa-file-word",
      "steps": [
        {
          "name": "jinja",
          "options": {
            "template": "default.html.j2",
            "content-type": "text/html",
            "extension": "html"
          }
        },
        {
          "name": "pandoc",
          "options": {
            "from": "html",
            "to": "docx"
          }
        }
      ]
    }
  ]
}
```

For `allowedKMs`, you can specify a list of knowledge models that the template can be used with (for example, when it is bound to its questions). You can even bound minimal and maximal version or let it unbound using `null` value. Using `wkhtmltopdf` and `pandoc`, you can specify template-related extra arguments for calls of those commands in case of document conversion.

Each template has information what formats and how are provided. A format has its own `uuid`, `name`, and `icon` for UI. Then there is a list of steps how to produce the output. Types of steps:

- `json` = only dumps document context as JSON; no `options`; must be first step; produces a JSON document
- `jinja` = uses Jinja2 templates to produce a document starting by root file specified in `options.template` (full Jinja2 can be used), `options.content-type` and `options.extension` must be used to specify type of the output; must be first step
- `wkhtmltopdf` - runs wkhtmltopdf; `options.args` can be used for additional arguments and options to run it; must be after step producing a HTML document; produces a PDF document
- `pandoc` - runs Pandoc; `options.from` and `options.to` define from which and to which type the transformation is used (use names according to docs), additionally `options.args` can be used for additional arguments and options to run it; must be used after step producing a document conforming `options.from`; produces a document according to `options.to`

6.4.2 Graphics and scripts

If you want to include some graphics or JavaScript, we recommend you to put it directly into the HTML template file. In case of graphics, use base64 encoded content (suitable for smaller images like icons and logos):

```
<img src=" iVBORw0KGgoAAAANSU
 hEUgAAAAUAAAAFCAYAAACNbyblAAAAHElEQVQI12P4//8/w38GI
 AXDIBKE0DHxgljNBAAO9TXL0Y4OHwAAAABJRU5ErkJggg=="
 alt="Red dot" />
```

Alternatively, you can of course reference picture that is accessible online. For JavaScript, again you can put there directly some script or reference it, for example, from some CDN:

```
<style type="text/javascript" src="https://code.jquery.com/jquery-3.3.1.min.js"></
↪style>
<style type="text/javascript">
  jQuery(".btn").click(function(){
    jQuery(this).toggleClass(".clicked");
  });
</style>
```

You can split your template code into multiple files and the use include directive that opens the file and inserts its content where the directive is placed - like we do for including CSS style in HTML template (only one complex HTML file is generated in the end):

```
<head>
  <title>Data Management Plan</title>
  <meta charset="utf-8">
  <style>{% include "root.css" %}</style>
</head>
```

6.4.3 Docker deployment

If you deploy the DS Wizard using Docker, you can mount custom files to templates/dmp folder and overwrite default template within *docker-compose.yml*:

```
server:
  image: datastewardshipwizard/wizard-server
  restart: always
  ports:
    - 3000:3000
  volumes:
    - /dsw/server/application.yml:/application/engine-wizard/config/application.yml
    - /dsw/templates/dmp:/application/engine-wizard/templates/dmp:ro
  # ... (continued)

docworker:
  # ...
  volumes:
   - /dsw/docworker/config.cfg:/app/config.cfg:ro
   - /dsw/templates/dmp:/app/templates:ro
```

6.5 Email templates

Similarly to *DMP templates*, you can customize templates for emails sent by the Wizard located in `templates/mail` folder. It also uses Jinja templating language (for email templates we use its implementation called Ginger). And you 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). The location of templates root folder is configurable via *`templateFolder`*.

6.5.1 Templates structure

The structure is following:

- `templates/mail/_common` = layout, styles, common files
- `templates/mail/_common/attachments` = attachments for all emails
- `templates/mail/_common/images` = inline images for all emails
- `templates/mail/<name>` = templates specific for this email type, should contain `message.html.j2` and `message.txt.j2` files (or at least one of them, mail servers prefer both variants)
- `templates/mail/<name>/attachments` = attachments specific for email type
- `templates/mail/<name>/images` = inline images specific for email type

All attachments are loaded from the template-specific and common folders and included to email with detected MIME type. It similarly works for inline images but those are not displayed as attachments just as related part to HTML part (if present). We highly recommend to use ASCII-only names without whitespaces and with standard extensions. Also, sending minimum amount of data via email is suggested.

6.5.2 Templates variables

All templates are provided also with variables:

- `appTitle` = from the configuration `appTitle`
- `clientAddress` = from the configuration *`clientUrl`*
- `mailName` = from the configuration *`name`*
- `user` = user (subject of an email), structure with attributes accessible via . (dot, e.g. `user.name`)

6.5.3 Email types

Currently, there are following types of mail:

- `registrationConfirmation` = email sent to user after registration to verify email address, contains `activationLink` variable
- `registrationCreatedAnalytics` = email sent to address specified in the configuration about registration of a new user (see *Analytics* config)
- `resetPassword` = email sent to user when requests resetting a password, contains `resetLink` variable

6.5.4 Docker deployment

Including own email templates while using dockerized Wizard is practically the same as for DMP templates. You can also bind whole `templates/mail` folder (or even `templates` if want to change both):

```
server:
  image: datastewardshipwizard/wizard-server
  restart: always
  ports:
    - 3000:3000
  volumes:
    - /dsw/application.yml:/application/engine-wizard/config/application.yml
    - /dsw/staging/templates/mail:/application/engine-wizard/templates/mail:ro
# ... (continued)
```

CHAPTER

SEVEN

UPGRADE GUIDELINES

7.1 Upgrading DSW

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

7.1.1 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-compose down
$ docker-compose up -d
```

7.1.2 Other setup (using Git)

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

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

7.1.3 Dependencies

If you need to upgrade MongoDB version, follow the official instructions in their documentation.

7.2 Upgrade process

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

7.2.1 2.1.X to 2.2.0

- Configuration option `issueUrl` is replaced by `webUrl` for feedback in application.yml.
- 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.
- 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.

7.2.2 2.0.X to 2.1.0

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

7.2.3 1.10.X to 2.0.0

- 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 you are using Docker for running DSW, you need to change it according to new documentation of *docker-compose.yml* and *Configuration*.
- Crontab image is no longer needed, see *Feedback synchronization*.
- A DMP template configuration file must contain list of `allowedKMs` (see the default *root* template).

7.2.4 1.9.X to 1.10.0

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

7.3 Compatibility

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

The DS Wizard is compatible with all recent versions of web browsers Chrome, Opera, Firefox, and Edge. We do not recomment use of Internet Explorer. Internally, there are components between is are following compatibility of versions:

DS Wizard	KM Metamodel	Registry
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 (or lower)	–	–

CHAPTER

EIGHT

CONTRIBUTING

8.1 Bugs and Ideas

If you have some idea how to extend the DS Wizard or find some bug, please create an issue according to information directly under *Report issue*: https://github.com/ds-wizard/ds-wizard/issues or contact us via email support@ds-wizard.org.

8.2 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

Tip: 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 codestyle, be DRY, and fit our overall architectures and structuring.

CHAPTER

NINE

ROADMAP

Note: DSW is currently being actively developed with montly releases. Backlog is being highly affected by provided feedback, community needs, and discussion with our key users.

9.1 Planned

9.1.1 2.3.0

- End of development: 29 April 2020
- Release: 6 May 2020
- Jira issues 2.3.0

9.2 Released

9.2.1 2.2.0

- End of development: 25 March 2020
- Release: 1 April 2020
- Jira issues 2.2.0

9.2.2 2.1.0

- End of development: 25 February 2020
- Release: 3 March 2020
- Jira issues 2.1.0

9.2.3 2.0.0

- End of development: 14 January 2020
- Release: 14 January 2020
- Jira issues 2.0.0

9.2.4 1.10.1

- End of development: 18 September 2019
- Release: 18 September 2019
- Jira issues 1.10.1

9.2.5 1.10.0

- End of development: 27 August 2019
- Release: 3 September 2019
- Jira issues 1.10.0

9.2.6 1.9.2

- End of development: 13 August 2019
- Release: 13 August 2019
- Jira issues 1.9.2

9.2.7 1.9.1

- End of development: 7 August 2019
- Release: 7 August 2019
- Jira issues 1.9.1

9.2.8 1.9.0

- End of development: 23 June 2019
- Release: 30 June 2019
- Jira issues 1.9.0

9.2.9 1.8.1

- End of development: 13 June 2019
- Release: 13 June 2019
- Jira issues 1.8.1

9.2.10 1.8.0

- End of development: 11 June 2019
- Release: 13 June 2019
- Jira issues 1.8.0

9.2.11 1.7.0

- End of development: 15 May 2019
- Release: 16 May 2019
- Jira issues 1.7.0

9.2.12 1.6.0

- End of development: 30 April 2019
- Release: 7 May 2019
- Jira issues 1.6.0

9.2.13 1.5.0

- End of development: 2 April 2019
- Release: 9 April 2019
- Jira issues 1.5.0

9.2.14 1.4.0

- End of development: 3 March 2019
- Release: 10 March 2019
- Jira issues 1.4.0

9.2.15 1.3.0

- End of development: 3 February 2019
- Release: 10 February 2019
- Jira issues 1.3.0

9.2.16 1.2.1

- End of development: 14 January 2019
- Release: 14 January 2019
- Jira issues 1.2.1

9.2.17 1.2.0

- End of development: 6 January 2019
- Release: 13 January 2019
- Jira issues 1.2.0

9.2.18 1.1.0

- End of development: 9 December 2019
- Release: 16 December 2019
- Jira issues 1.1.0

9.2.19 1.0.0

- End of development: 24 October 2019
- Release: 30 October 2019

INDEX

M

N

O

P

Q

S

T

U

V