Data Stewardship Wizard

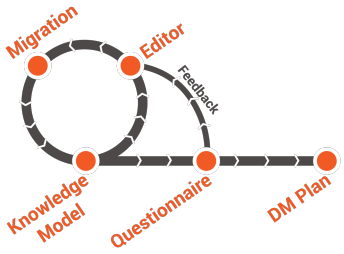
Release 1.7.0

DSW Team

Jun 18, 2019

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**.



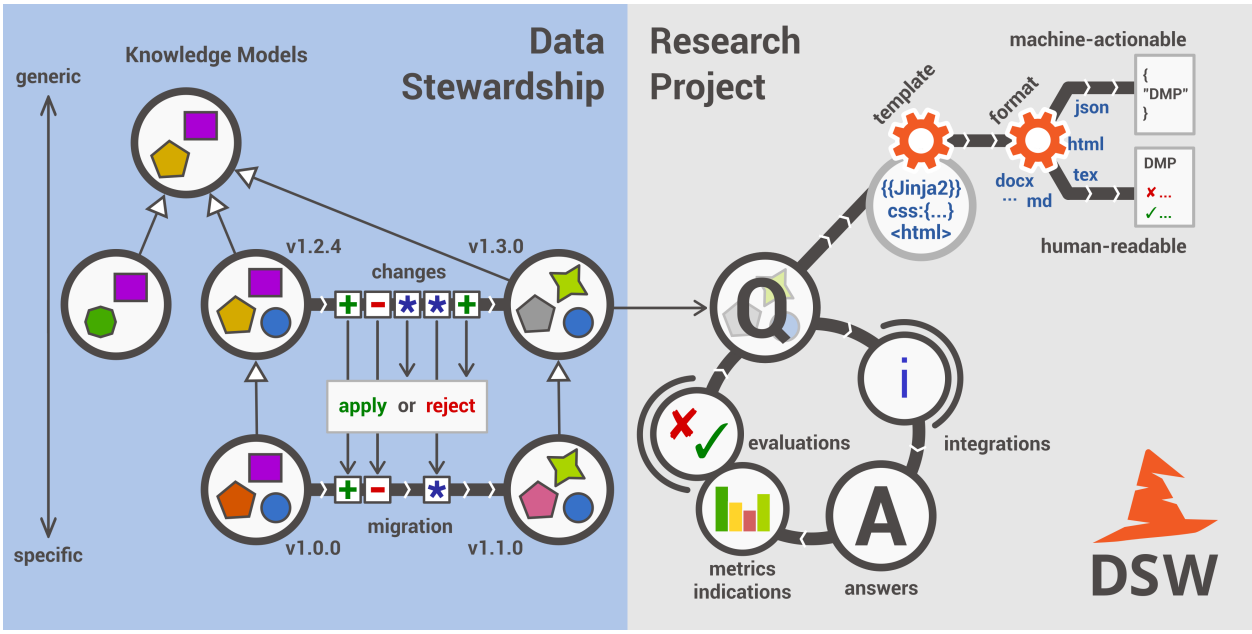
CHAPTER

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 Learn More

- DS Wizard (landing page)
- Leaflet
- Diagrams
- ELIXIR All Hands 2018
  - Poster
  - Flask Talk

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 Planner 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
  - *Export* = export plan in selected format and template
  - *Edit*
  - *Delete*

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)
    - *Delete*
    - *Fork Knowledge Model* = shortcut to create editor from specific version of KM
    - *Create Questionnaire* = shortcut to create planner from specific version of KM

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

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.

3.1 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",
                                  //...
                                }
                      }
      }, //...
    ], //...
  }, //...
}
```

3.2 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.

3.3 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).

CHAPTER

FOUR

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.

4.1 Public Instance

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

- Landing: https://ds-wizard.org
- Demo instance (free to use, for trying out all the features, unstable)
  - Client: https://demo.ds-wizard.org
  - Server: https://api.demo.ds-wizard.org
- Researchers instance (free to use, to build own DMPs, prepared for serious work)
  - Client: https://researchers.ds-wizard.org
  - Server: https://api.researchers.ds-wizard.org

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.

4.2 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` and `worker-config.json`), 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:

  dsw_server:
    image: datastewardshipwizard/server
    restart: always
    ports:
      - 3000:3000
    volumes:
      - /dsw/app-config.cfg:/dsw/config/app-config.cfg
    links:
      - rabbitmq
      - mongo

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

  dsw_server_worker:
    image: datastewardshipwizard/crontab
    restart: always
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock:ro
      - /dsw/worker-config.json:/opt/crontab/config.json
    environment:
      - API_URL=http://localhost:3000

  rabbitmq:
    image: rabbitmq

  mongo:
    image: mongo:3.0.15
    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

4.2.1 Upgrade

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

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/server
$ docker pull datastewardshipwizard/client
$ docker pull datastewardshipwizard/crontab
$ docker-compose down
$ docker-compose up -d
```

4.3 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.

4.3.1 General Requirements

- The Haskell tool Stack (for server side)
- NPM (for client side)
- MongoDB (database, needs to be running)
- RabbitMQ (optional, based on your *Configuration*)
- wkhtmltopdf (>= 0.12.5, for DMP exports in PDF)
- Pandoc (>= 2.6, for DMP exports in other formats aside HTML, PDF, and JSON)

4.3.2 Server

1. Get the server app (dsw-server)

   ```
   git clone git@github.com:ds-wizard/dsw-server.git
   ```

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

   ```
   cp config/app-config.cfg.example config/app-config.cfg
   ```

3. Build (takes a lot of time, downloads & builds all dependencies)

   ```
   stack build
   ```

4. Run (requires MongoDB and RabbitMQ according to configuration)

   ```
   stack exec dsw-server
   ```

4.3.3 Client

1. Get the client app (dsw-client)

   ```
   git clone git@github.com:ds-wizard/dsw-client.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 start
   ```

5. Open app in your favorite browser

6. For minified production-ready version, use

   ```
   npm run build
   ```

4.3.4 Upgrade

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

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
```

4.4 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.

4.5 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 *Registry* configuration. Then your instance is connected automatically to the Registry service for specific functionality such as accessing shared knowledge models.

4.6 Compatibility

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
1.8.0	3	1.0.0
1.7.0	2	–
1.6.0	1	–
1.5.0 (or lower)	–	–

Important: DSW Client and Server should always use matching version (compatibility is assured)!

4.7 Other "Setups"

4.7.1 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*)

4.7.2 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-1.0.0.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-1.0.0.json`

```
$ mongoimport --db dsw-server \
              --collection publicPackages \
              --file public-package-root-1.0.0.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.

4.7.3 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
```

(continues on next page)

(continued from previous page)

```
# 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

FIVE

CONFIGURATION

5.1 Server

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

5.1.1 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.

5.1.2 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
  clientUrl: http://localhost:8080
  serviceToken: NlQSNbGvh7EtcpinGnHE9g91
  integrationConfig: integration.yml
  registrationEnabled: true
  publicQuestionnaireEnabled: true
  levelsEnabled: true
  itemTitleEnabled: true
  questionnaireAccessibilityEnabled: true

client:
  appTitle:
  appTitleShort:
  welcomeWarning:
  welcomeInfo:
```

(continues on next page)

(continued from previous page)

```
  privacyUrl:

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

jwt:
  secret: gVoYNuEJuGGAQMIaYw43BN3e
  version: 1
  expiration: 14

roles:
  defaultRole: DATASTEWARD
  admin: [UM_PERM, ORG_PERM, KM_PERM, KM_UPGRADE_PERM, KM_PUBLISH_PERM, PM_READ_PERM, PM_WRITE_PERM, QTN_PERM, DMP_PERM]
  dataSteward: [KM_PERM, KM_UPGRADE_PERM, KM_PUBLISH_PERM, PM_READ_PERM, PM_WRITE_PERM, QTN_PERM, DMP_PERM]
  researcher: [PM_READ_PERM, QTN_PERM, DMP_PERM]

mail:
  enabled: false
  name:
  email:
  host:
  port:
  ssl:
  username:
  password:

analytics:
  enabled: false
  email:

feedback:
  enabled: false
  token:
  owner:
  repo:
  issueurl:
```

Section aside *General* and *JWT* 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.

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 *Worker* component.

integrationConfig

Type String

Default `"integration.yml"`

Filename or whole path of integration configuration file (see *Integration*). In case of path, it is relative to the `config` folder.

registrationEnabled

Type Boolean

Default `true`

If registrations within the DS Wizard instance are enabled or disabled.

publicQuestionnaireEnabled

Type Boolean

Default `false`

If public questionnaire (i.e. questionnaire demo without registration) functionality within the DS Wizard instance is enabled or disabled.

levelsEnabled

Type Boolean

Default `true`

If questions can be related to certain desirability levels/phases.

itemTitleEnabled

Type Boolean

Default `false`

If list questions require specific title per item or are just groups of subquestions.

questionnaireAccessibilityEnabled

Type Boolean

Default `true`

If questionnaires can be set private, public read-only, or public. When disabled (i.e. value is set to `false`), all questionnaires are public.

Client

Configuration related to parts displayed in the client application.

`appTitle`

Type String

Default (nothing)

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

`appTitleShort`

Type String

Default (nothing)

Short name of the DS Wizard instance (displayed, for example, on the top of the navigation bar)

Tip: Use consistent *`appTitle`* and *`appTitleShort`*.

`welcomeWarning`

Type String

Default (nothing)

Warning text for users that displays after login (if any).

`welcomeInfo`

Type String

Default (nothing)

Info text for users that displays after login (if any).

`privacyUrl`

Type String (URL)

Default `"https://ds-wizard.org/privacy.html"`

URL to page with privacy policy of the service.

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`).

JWT

JSON Web Token configuration for communication.

secret

Type String

Default `""` (empty)

Secret string used for JWT signing and validation (we recommend to generate some randomly).

version

Type Integer

Default `1`

Our internal version of JWT Payload for DS Wizard (only `1` at the moment).

expiration

Type Integer

Default `14`

Number of days until a token expires.

Roles

Basic configuration of roles and privileges within the DS Wizard instance. All roles and permission use capitalized names. There are three roles: *Researcher*, *Data Steward*, and *Administrator*. We recommend to leave the permission as in example *application.yml*, otherwise our *Usage* documentation may not match your configuration.

`defaultRole`

Type Role (Enumeration [`RESEARCHER`, `DATASTEWARD`, `ADMIN`])

Default *DATASTEWARD*

Role that will be assigned to newly registered user.

`admin`

Type Permissions

Default `[ UM_PERM, ORG_PERM, KM_PERM, KM_UPGRADE_PERM, KM_PUBLISH_PERM, PM_READ_PERM, PM_WRITE_PERM, QTN_PERM, DMP_PERM]`

List of permissions for *Administrator*/`ADMIN`.

`dataSteward`

Type Permissions

Default `[KM_PERM, KM_UPGRADE_PERM, KM_PUBLISH_PERM, PM_READ_PERM, PM_WRITE_PERM, QTN_PERM, DMP_PERM]`

List of permissions for *Data Steward*/`DATASTEWARD`.

`researcher`

Type Permissions

Default `[PM_READ_PERM, QTN_PERM, DMP_PERM]`

List of permissions for *Researcher*/`RESEARCHER`.

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 `true`

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

`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.

Registry

`token`

Type String

Default `""` (empty)

Your organization token aquired by registration within the Registry service. More information can be found in *Registry* section of installation documentation.

Feedback

Configuration for feedback functionality within questionnaires via GitHub issues.

`enabled`

Type Boolean

Default `true`

If feedback functionality will be used.

`token`

Type String

Default `""` (empty)

GitHub personal access token with permission to create issues in the selected repository.

`owner`

Type String

Default `""` (empty)

GitHub username or organization under which is the repository for feedback created.

`repo`

Type String

Default `""` (empty)

Name of the repository (without owner name).

`issueurl`

Type URI

Default `"https://github.com/:owner/:repo/issues/:issueId"`

Template URL for feedback issue.

5.1.3 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
  baseUrl: 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.

5.2 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

```javascript
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 variables.scss file
  - /path/to/customizations.scss:/customizations/variables.scss
  # mount other assets, you can then refere them from scss using '/assets/...'
  - /path/to/assets:/usr/share/nginx/html/assets
```

For more information about variables and assets, visit Theming Bootstrap.

5.3 Worker

For running scheduled service tasks, there is a (optional) server worker component. Its configuration is done with `API_URL` which is the same as when configuring *Client* but also `SERVICE_TOKEN` that must correspond with server configuration *`serviceToken`*.

```
API_URL=https://api.dsw.example.org
SERVICE_TOKEN=<secret_token>
```

5.4 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 (we use its implementation called Ginger) is useful.

5.4.1 Template files

We provide currently basic `root` template but it is possible to get inpired and create more or edit it:

- `templates/dmp/root.json` = metadata about the template
- `templates/dmp/root.html.j2` = main template file
- `templates/dmp/root.css` = stylesheet file included in the main file

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 `dmp` 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:

```
{
  "uuid": "43a3fdd1-8535-42e0-81a7-5edbff296e65",
  "name": "Common DSW Template",
  "rootFile": "root.html.j2"
}
```

5.4.2 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>
```

5.4.3 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*:

```
dsw_server:
image: datastewardshipwizard/server
restart: always
ports:
  - 3000:3000
volumes:
  - /dsw/app-config.cfg:/dsw/config/app-config.cfg
  - /dsw/root.json:/dsw/templates/dmp/root.json
  - /dsw/root.html.j2:/dsw/templates/dmp/root.html.j2
  - /dsw/root.css:/dsw/templates/dmp/root.css
external_links:
  - mongo_mongo_1:mongo
networks:
  - default
  - mongo_default
```

5.5 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. 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).

5.5.1 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.

5.5.2 Templates variables

All templates are provided also with variables:

- `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`)

5.5.3 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

5.5.4 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):

```
dsw_server:
  image: datastewardshipwizard/server
  restart: always
  ports:
    - 3000:3000
  volumes:
    - /dsw/app-config.cfg:/dsw/config/app-config.cfg
    - /dsw/templates/mail:/dsw/templates/mail
# ... (continued)
```

CHAPTER

SIX

CONTRIBUTING

6.1 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.

6.2 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/dsw-server
- https://github.com/ds-wizard/dsw-client

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

SEVEN

ROADMAP

7.1 Planned

7.1.1 1.8.0

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

7.1.2 1.9.0

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

7.2 Released

7.2.1 1.7.0

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

7.2.2 1.6.0

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

7.2.3 1.5.0

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

7.2.4 1.4.0

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

7.2.5 1.3.0

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

7.2.6 1.2.1

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

7.2.7 1.2.0

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

7.2.8 1.1.0

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

7.2.9 1.0.0

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

INDEX