

HSE Setup and Usage Guide

Robert Jans

January 5, 2021

Contents

1	(Temporary) Source Download and Deployed Application	3
2	Employed Technologies	3
3	Configuration, Build, and Deployment	3
3.1	Dependencies	3
3.2	Configuration Files	3
3.2.1	Maven configuration: pom.xml	3
3.2.2	Specific configurations in <code>.properties</code> files	4
3.2.3	docker-compose.yml	4
3.3	Creating a local build	4
3.3.1	Preparing the database	4
3.3.2	Issuing the build command	4
3.3.3	Running the application	4
3.4	Example deployment on <i>Ubuntu Server</i> with <i>Apache2</i>	5
3.4.1	Create and transfer the <i>Docker</i> image	5
3.4.2	Load the image and start the application	5
3.4.3	Apache2 configuration	5
4	Usage	6
4.1	Users, Roles, and their Definition (/admin/ui)	6
4.2	URL lists and Document collections (/indexing/ui)	6
4.3	Experiment Definition (/experiments/ui)	7
4.4	Experiment Configuration (/experiments/setup/ui)	7
4.4.1	Configuration in stand-alone mode	7
4.4.2	Configuration in Qualtrics mode	8
4.5	Experiment Execution (/experiments/run/ui)	9
4.6	Setting up a linked Qualtrics survey	9
4.6.1	Survey Questions	9
4.6.2	Survey Flow	10
4.7	Experiment Evaluation and Data Export (/experiments/eval/ui)	12
4.7.1	Data Representation	12

1 (Temporary) Source Download and Deployed Application

The source codes and related documentation files are available on *GitHub* under

https://github.com/robix82/bsc_project/.

The repository can be cloned by issuing the following command

```
git clone https://github.com/robix82/bsc_project.git
```

For testing the application and its interactions with Qualtrics, I temporarily deployed it on

<http://www.robix-projects.org/hse>.

You can log in using the following credentials;

- username: admin
- password: admin

2 Employed Technologies

The project is constructed as a web application using the *SpringBoot* framework. It consists of a back-end written in java, HTML pages (served via the *Thymeleaf* templating engine), some (JavaScript) to be run on client-side, and *CSS* files for the user interface styling. Data is stored in part in a *MySQL* database and in part as text files on the server's file system. The *JQuery* and *Bootstrap* frameworks are used for keeping the front-end code as simple as possible.

Dependency management and build configurations are handled by the *Maven* project management tool. In order to allow a simple deployment procedure, the application is packaged into a *Docker* image at build time. The application can be started and stopped using *Docker Compose* which automatically downloads, initializes, and links the required *MySQL* database. For a detailed deployment description, see section 3. For inspecting and/or modifying the source code, I suggest using the *Eclipse* IDE.

3 Configuration, Build, and Deployment

3.1 Dependencies

The system on which you build the application must have the following software installed:

- Java JDK 11
- Apache Maven 3.6.3
- Docker version 20.10.1

If you need to run the application locally without using docker, also *MySQL* is required.

The server on which the application is to be deployed only needs *Docker* and *Docker Compose*.

3.2 Configuration Files

3.2.1 Maven configuration: pom.xml

The build settings used by *Maven* are defined in `/hse/pom.xml`. This file contains some general information about the project, such as name and version, as well as a list of java packages and *Maven* plugins that are downloaded and set up at build time. The File also declares two profiles (“dev” and “prod”). The “dev” profile is intended for creating a local build to be used during development, while the “prod” profile is to be used for building the deployment version. The profiles are linked to specific configuration files

which contain various settings such as server ports and global constants: when the profile “dev” is selected, the application uses the file `/hse/src/main/resources/application-dev.properties`; when “prod” is selected, `/hse/src/main/resources/application-prod.properties`. By default the “dev” profile is selected; for using the “prod” profile, the flag `-Pprod` is to be included in the *Maven* command (e.g. `mvn -Pprod clean install`).

3.2.2 Specific configurations in .properties files

The directory `/hse/src/main/resources/` contains three files with extension `.properties`: `application.properties`, `application-dev.properties` and `application-prod.properties`. The first one contains settings that are applied independently of the selected profile, while the other two contain profile-specific settings. The crucial settings to be considered at build time are the `spring.datasource.xxx` properties, which indicates the database which the application is going to connect to, and the `baseUrl` parameter, which indicates the prefix used at server-level. For instance, in the `application-prod.properties` as I have set it up, the data source is pointing to a *MySQL* Docker container, and the base URL is set to `/hse/` since I deploy it on `http://www.robix-projects.org/hse/`. In the `application-dev.properties` file the data source points to a local *MySQL* instance and the `baseUrl` is `/`. The other properties indicate directory paths and should not need to be modified.

3.2.3 docker-compose.yml

The simplest way to run the application on a server is by using *Docker Compose*. The way in which the containers are created from the images and the internal ports used are defined in `/hse/docker-compose.yml`.

3.3 Creating a local build

3.3.1 Preparing the database

In order to run the application locally, a *MySQL* database service running on port 3306 is required. The service must contain a database named `hse_db` and should be accessible via username `root` and password `root`. The tables are created automatically at application startup. If you need to use other login credentials, or the service is running on another port, these parameters can be set in `application-dev.properties`.

3.3.2 Issuing the build command

The command

```
mvn clean install
```

initiates the build process. The process involves executing several test suites, which should work without failure. In case the tests fail (e.g. due to path incompatibilities or missing files) the tests can be skipped using the `-DskipTests` flag:

```
mvn -DskipTests clean install
```

3.3.3 Running the application

Once the application is built, it can be run in several ways. During development, it is convenient to run it from the IDE (in *Eclipse* package explorer, right-click on project → Run As → Spring Boot App). Alternatives are to run it from command-line using *Maven*:

```
cd hse/  
mvn spring-boot:run
```

or using *Java*:

```
cd hse/target/  
java -jar hse-0.1.jar
```

3.4 Example deployment on *Ubuntu Server with Apache2*

3.4.1 Create and transfer the *Docker* image

The first step consists of creating the application's image by issuing

```
cd hse/  
mvn -Pprod clean install
```

At this point, the output of `docker image ls` should contain a line similar to:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
robix82/usi.ch-hse	0.1	c2137e7cc110	37 seconds ago	758MB

Once the image is created it can be exported as a `.tar` file by issuing

```
docker save robix82/usi.ch-hse:0.1 > hse.tar
```

Finally the `.tar` file and `/hse/docker-compose.yml` must be copied to the server, e.g. using `scp`.

3.4.2 Load the image and start the application

On the server, the image from the `.tar` file can be loaded with

```
docker load < hse.tar
```

With the image loaded, the application can be started by issuing

```
docker-compose up &
```

from the directory containing the `docker-compose.yml` file. The required *MySQL* image will be downloaded and initialized automatically.

At this point, the application is reachable on port 8081 (the port can be configured in `docker-compose.yml`).

3.4.3 Apache2 configuration

In order to make the application reachable on the server's external address with a custom prefix, it is necessary to configure a virtual host using Apache's `mod_proxy` module. For details on `mod_proxy`, please refer to <https://www.digitalocean.com/community/tutorials/...>

The virtual host configuration is done by placing a file (in this example `hse.conf`) in `/etc/apache2/sites-available/` and creating a symlink to it in `/etc/apache2/sites-enabled/`:

```
ln -s /etc/apache2/sites-available/hse.conf /etc/apache2/sites-enabled/
```

The content of the `.conf` file should look similar to

```
<VirtualHost *:80>  
  
    ServerName www.robix-projects.org  
    ProxyPreserveHost On  
  
    ProxyPass /hse/ http://127.0.0.1:8081/  
    ProxyPassReverse /hse/ http://127.0.0.1:8081/  
  
</VirtualHost>
```

This configuration makes the application available under `http://www.robix-projects.org/hse`. Notice that the prefix `/hse/` must match the `baseUrl` property in `application-prod.properties` and the port (8081 in this example) must correspond to the port defined in `docker-compose.yml`.

4 Usage

4.1 Users, Roles, and their Definition (/admin/ui)

The application distinguishes three types (roles) of users: *Administrators*, *experimenters*, and *participants*. Depending on a user's role, different UI elements are visible or accessible: participants can access only the search interface; experimenters have access to the indexing and the experiment setup interfaces; administrators have access to all interfaces, including a page for creating, updating, and removing administrators and experimenters. Participants can be defined by administrators or experimenters when an experiment is set up, or automatically if the experiment is run within a *Qualtrics* survey.

If the application is newly installed, a default user with *Administrator* role is created. This user can log in using the user name “*admin*” and password “*admin*”.

4.2 URL lists and Document collections (/indexing/ui)

Before an experiment can be set up, at least one document collection must be available. These can be created on the *indexing* UI page available to experimenters and administrators (see **Figure 1**). Creating a document collection involves the following steps:





- Upload a URL list, i.e. a text file (.txt) containing one URL per line.
- Define a doc collection by setting its name, the URL list to be used, and the language (*IT* or *EN*) of the web pages.
- Start the indexing process. This may take a long time since it involves downloading all the pages over the network; while testing I observed times in the order of one second per URL.


Document collections will later be assigned to test groups, so the search engine returns different results depending on which test group a participant belongs to. Optionally a document collection can be set as a fixed result for the first query; in this case, the first query submitted by a participant returns this entire document collection, in the order in which the URLs are set in the list used for generating it.

[Search](#) [Experiments](#) [Indexing](#) [Admin](#) [Logout](#)





Indexing

URL lists

urls_mixed.txt		
urls_good.txt		



Document collections

24	mixed	urls_mixed.txt	IT	indexed			index
26	good	urls_good.txt	IT	indexed			index




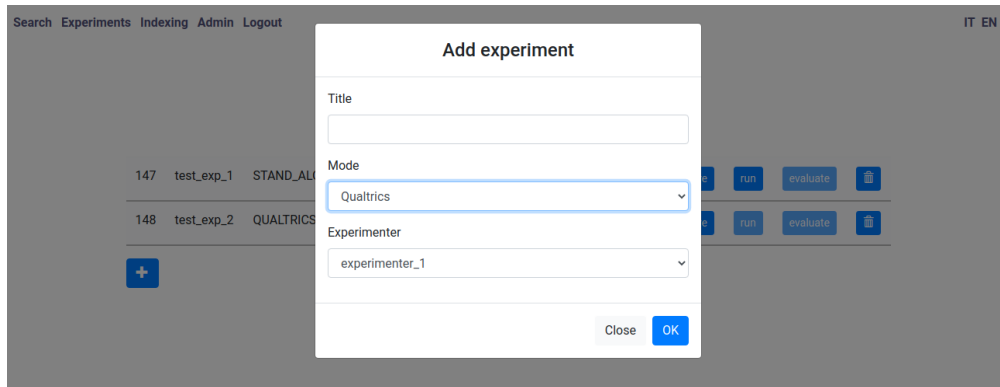
Figure 1: User interface for creating document collections

4.3 Experiment Definition (/experiments/ui)

The `/experiments/ui` page lists all defined experiments. Each line shows the experiment's unique id (needed for running with a *Qualtrics* survey), its title, mode (*stand-alone* or *Qualtrics*), the assigned experimenter, the date on which it was defined, its status (one of *created*, *ready*, *running*, or *complete*), and buttons leading to the UI for configuration, execution, and evaluation. The buttons are enabled or disabled depending on the experiment's status. **Figure 2** shows the experiments interface.



(a) Experiments list



(b) Popup for defining a new experiment

Figure 2: Interface for defining experiments

4.4 Experiment Configuration (/experiments/setup/ui)

The interface for experiment configuration is reachable by clicking on the *configure* button in the experiments list UI. If the experiment's mode is *stand-alone* the configuration involves creating test groups and assigning document collections and participants to them. Moreover, a document collection can be set as a predefined result list for the first query. Participants and groups can be defined manually or loaded from a configuration file. The same applies to *Qualtrics* mode; the only difference is that participants don't need to be defined, since they are created while they take part in the survey.

4.4.1 Configuration in stand-alone mode

The setup UI for stand-alone experiments includes a section for uploading, inspecting, or deleting configuration files (which can be used for defining test groups and participants), as well as a section for editing the test groups. **Figure 3a** shows the page after two test groups have been added and edited; **Figure 3b** shows an example of a configuration file. For the configuration to be complete, i.e. its status being set to *complete* and the *run* UI being available, there must be at least one test group defined, and each test group must have some participants and at least one document collection.

set up test_exp_1

The screenshot shows the 'set up test_exp_1' interface. On the left, the 'Experiment details' section includes 'Available configuration files' with a search bar and a list of files. Below this, the 'test groups' section shows two groups: 'testGroup2 [173]' and 'testGroup1 [168]'. Each group has a 'Document collections' list, a 'First query list' (fqr1), and a 'Participants' list. On the right, an example configuration file is shown with comments and participant definitions for two groups.

Available configuration files

experimentConfigExample.txt

Experiment details

test groups

testGroup2 [173]

Document collections

mixed IT

good IT

First query list

fqr1

Participants

participant7

participant6

testGroup1 [168]

Document collections

good IT

First query list

Participants

participant3

participant2

participant1

```
# Example experiment configuration file
# lines starting with '#' are ignored

# a test group is defined by setting its name; the group name must NOT contain white spaces
# the following lines, up to the next test group definition or the end of the file
# define the participants (username + space + password). Usernames must be unique.

group: testGroup1
participant1 pwd1
participant2 pwd2
participant3 pwd3
participant4 pwd4

group: testGroup2
participant5 pwd5
participant6 pwd6
participant7 pwd7
```

(a) Experiment setup UI for stand-alone mode

(b) Example of a configuration file

Figure 3

4.4.2 Configuration in Qualtrics mode

In Qualtrics mode the setup options are similar, but the participants don't need to be specified, as they are defined during the survey. So there is no need for configuration files. For the configuration to be complete, there must be at least one test group, and each test group must have at least one document collection.

Figure 4 shows the configuration interface after creating and editing two test groups. Notice the test group's id number: this will be needed when setting up the related Qualtrics survey.

set up test_exp_2

The screenshot shows the 'set up test_exp_2' interface for Qualtrics mode. The 'Experiment details' section shows two test groups: 'testGroup1 [199]' and 'testGroup2 [200]'. Each group has a 'Document collections' list and a 'First query list' (fqr1). A red handwritten note 'id to be used in Qualtrics' points to the ID number '199' of testGroup1.

Experiment details

test groups

testGroup1 [199]

Document collections

mixed IT

good IT

First query list

fqr1

testGroup2 [200]

Document collections

mixed IT

First query list

fqr1

id to be used in Qualtrics

Figure 4: Experiment setup UI for Qualtrics mode

4.5 Experiment Execution (/experiments/run/ui)

If an experiment is configured, the related execution UI becomes available. The interface is quite simple: there is a button for starting, stopping, and resetting the experiment, as well as live updated information on participant's activities. In stand-alone mode, all participants are listed from the beginning, while in Qualtrics mode they appear as they log in.

The purpose of the start / stop mechanism is to enable participant access and to measure the experiment's duration. As soon as the experiment is started, participants can log in; when the experiment is stopped they are automatically logged out. In case the experiment is defined in Qualtrics mode, the participants are redirected back to the survey. After an experiment is started, the page can be left and returned to later. **Figure 5** shows the interface for running an experiment.

run test_exp_1

[stop](#)

01:02:40

experiment running: participants are enabled to log in

testGroup2					testGroup1				
id	user name	status	queries	clicks	id	user name	status	queries	clicks
197	participant7	offline	0	0	194	participant3	offline	0	0
198	participant6	offline	0	0	195	participant2	offline	0	0
					196	participant1	online	2	3

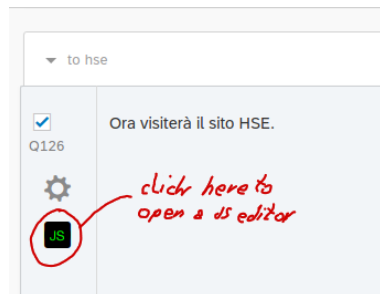
Figure 5: Experiments run UI

4.6 Setting up a linked Qualtrics survey

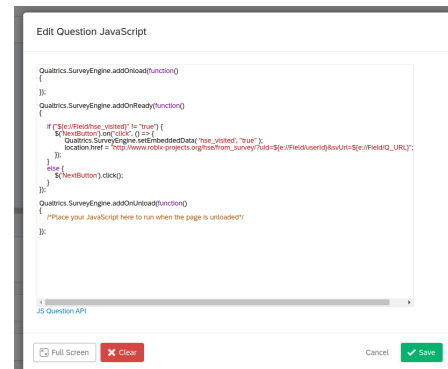
For linking a survey to an experiment, the survey must contain a block whose “next” button redirects to HSE, a failure block to be displayed in case accidentally the experiment is not running, or there is some connection error, and some settings at the beginning of the *Survey flow*.

4.6.1 Survey Questions

The redirection is set via *JavaScript* in the survey question (see **Figure 6**). The failure block needs no specific configuration.



(a) Opening the JS editor



(b) Edited JS

Figure 6

In the JS editor there are three default functions: `Qualtrics.SurveyEngine.addOnload(...)`, `Qualtrics.SurveyEngine.addOnReady(...)`, and `Qualtrics.SurveyEngine.addOnUnload(...)`. The first and third functions don't need to be modified, while the `Qualtrics.SurveyEngine.addOnReady(...)` function needs to be edited as follows:

```
Qualtrics.SurveyEngine.addOnReady(function()
{
    if ("${e://Field/hse_visited}" != "true") {

        $('NextButton').on("click", () => {
            Qualtrics.SurveyEngine.setEmbeddedData( 'hse_visited', "true" );
            location.href = "http://[hse server url]/from_survey/
                            ?uid=${e://Field/userId}&svUrl=${e://Field/Q_URL}";

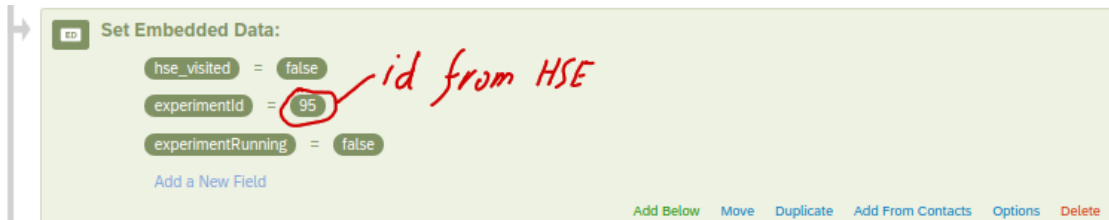
        });
    }
    else {

        $('NextButton').click();
    }
});
```

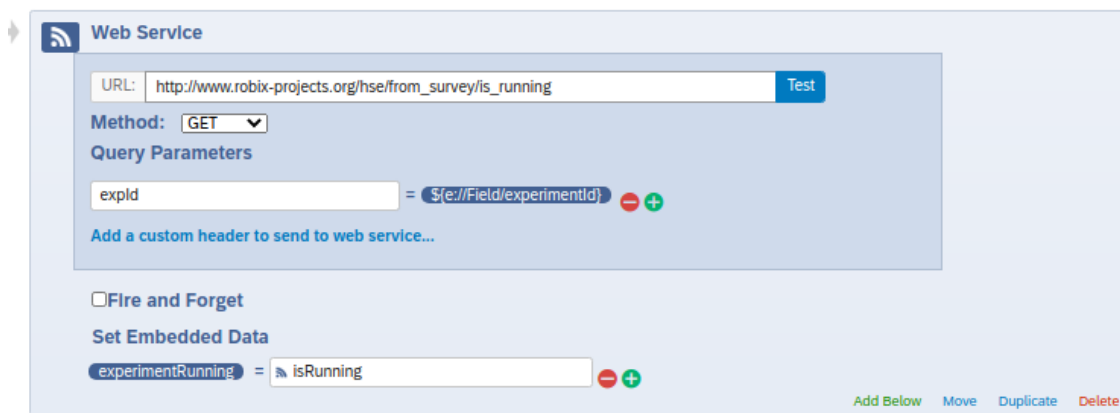
4.6.2 Survey Flow

The following elements are to be added at the beginning of the survey flow, in the order they are presented here:

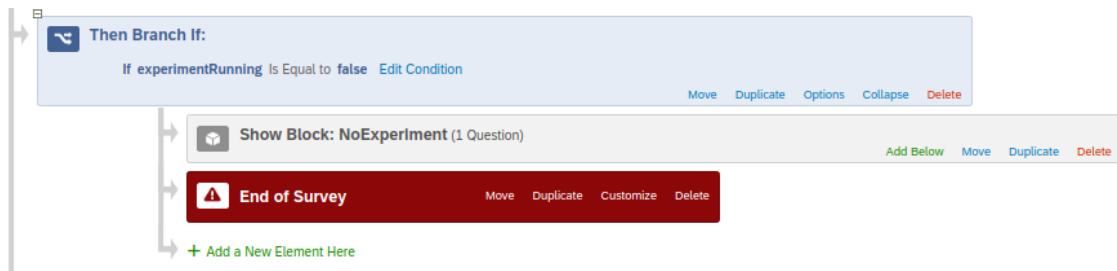
1. An *Embedded Data* element for initializing some variables:



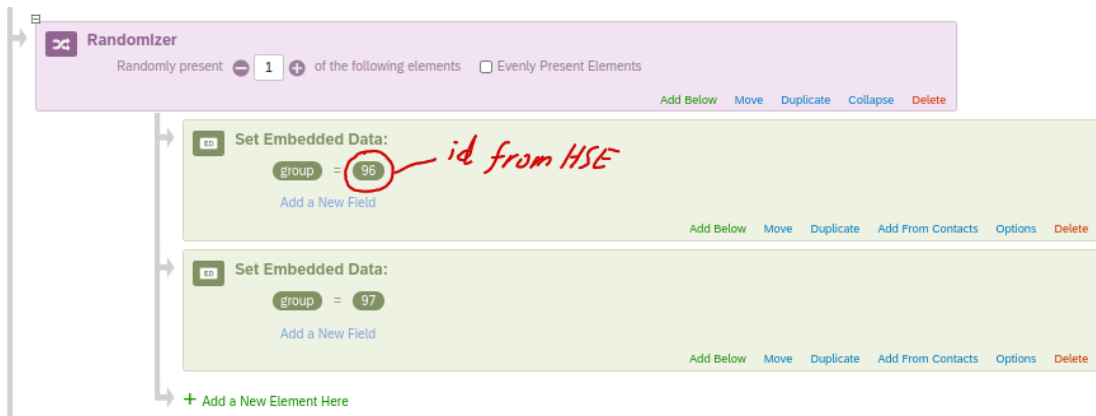
2. A *Web Service* element to do an API call for checking that the given experiment is actually running:



3. A *Branch* element to interrupt the survey if the experiment is not running:



4. A *Randomizer* element for assigning participants to test groups:



5. A *Web Service* element for initializing the participant in HSE:

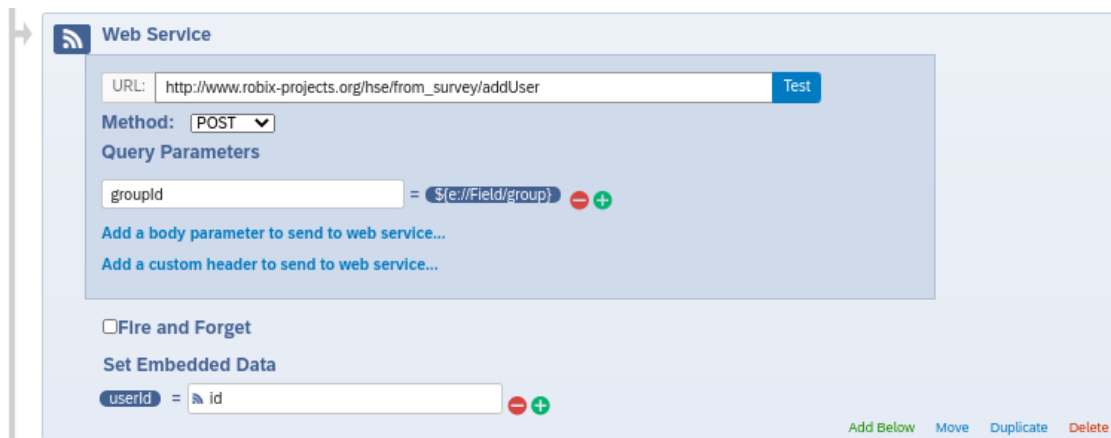


Figure 7 shows the entire edited survey flow.

4.7 Experiment Evaluation and Data Export (/experiments/eval/ui)

Once an experiment is completed, its evaluation page is accessible. The page contains several links for downloading the raw or partially pre-processed data, as well as a summary including per-user means and standard deviations for relevant derived data such as the number of documents visited, the time spent on a document, and the distribution of these measures over the different document collection.

4.7.1 Data Representation

The data collected during an experiment consists of a list of *Usage Events*. There are three kinds of such events: *Session Events* (login and logout), *Query Events* (generated when a participant submits a search query), and *Document Click Events* (generated when a participant clicks on a link from the results list and visits a page). All *Usage Events* contain the following data fields:

- A unique id (generated by the database system).
- A timestamp of the moment when the event was generated.
- The user id of the participant who generated the event.
- The id and name of the test group the participant belongs to
- The event type (one of `SESSION`, `QUERY`, or `DOC_CLICK`)

Session Events contain an additional field indicating whether it was a login or logout event; *Query Events* contain the query string submitted, the total number of retrieved results, and the proportions in which the document collections are represented in the results. *Document Click Events* contain the document's URL, an id, the name and id of the collection the document belongs to, and its rank (position within the query results).

The experiment's evaluation page allows to download the entire raw data in a single `.csv` or `.json` file, or the same data split into per-user histories (in a single `.json`, or in separate `.csv` files).

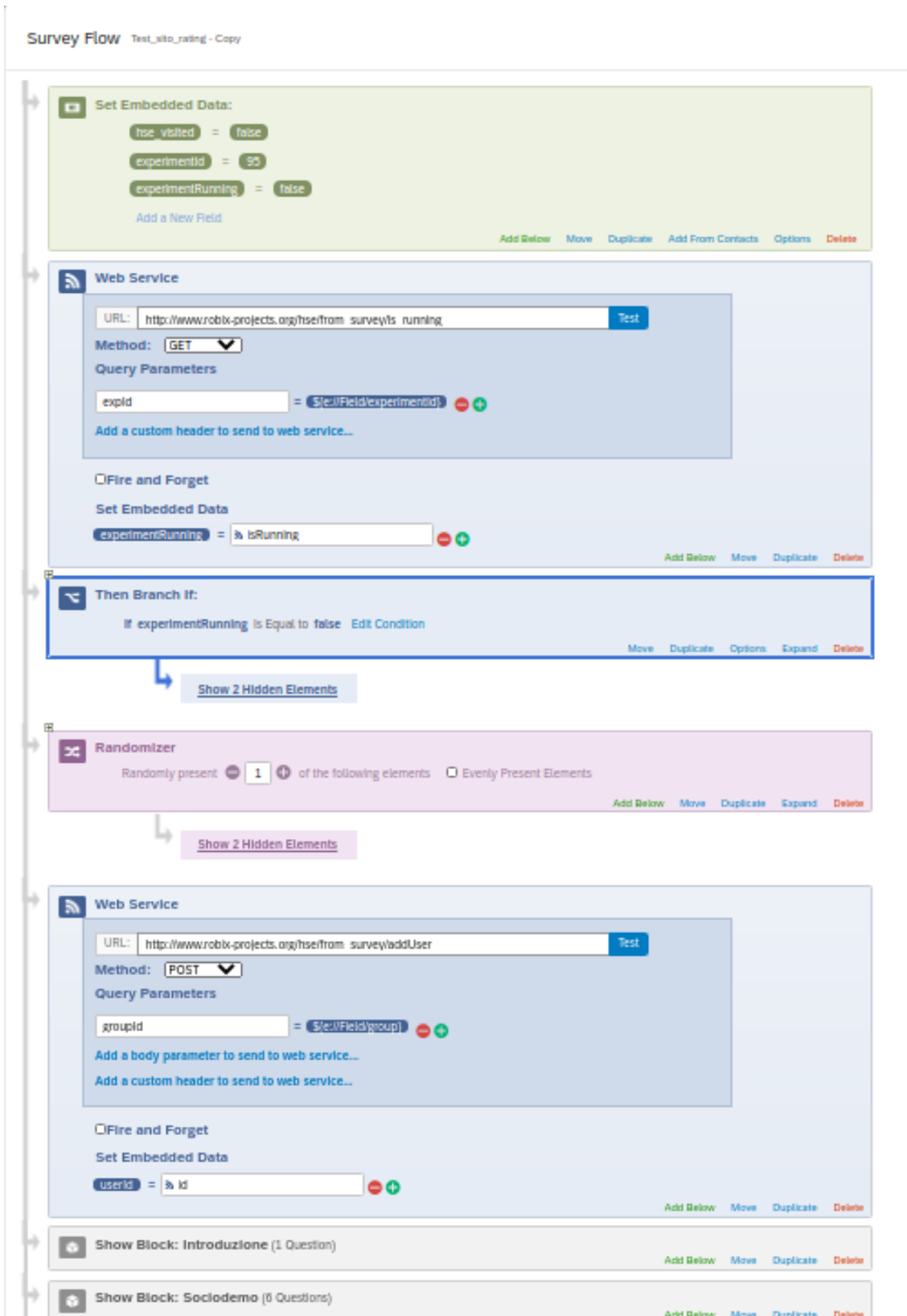


Figure 7: Edited Survey Flow