

---

# swift-browser-ui

*Release 2.0.1*

**Jun 02, 2023**



---

## Contents:

---

<b>1</b>	<b>Setup Instructions</b>	<b>3</b>
1.1	Environment Setup . . . . .	3
1.2	Scaling up the service . . . . .	4
1.3	Further reading and citations . . . . .	5
<b>2</b>	<b>Getting started</b>	<b>7</b>
2.1	Command line interface . . . . .	7
<b>3</b>	<b>Deployment</b>	<b>9</b>
3.1	Dockerfile . . . . .	9
3.2	Database for sharing functionality . . . . .	9
3.3	Sharing functionality back-end . . . . .	9
3.4	Shared access request back-end . . . . .	10
3.5	Upload runner back-end . . . . .	10
3.6	Kubernetes Integration . . . . .	11
<b>4</b>	<b>Architecture</b>	<b>13</b>
4.1	About login process . . . . .	13
4.2	API . . . . .	13
<b>5</b>	<b>Web User Interface</b>	<b>15</b>
5.1	User information page . . . . .	15
5.2	Container page . . . . .	15
5.3	Object page . . . . .	16
5.4	Non-whitelisted mode . . . . .	16
<b>6</b>	<b>Sharing functionality</b>	<b>17</b>
6.1	Sharing a container . . . . .	17
6.2	Viewing containers shared from the project . . . . .	17
6.3	Viewing containers shared to the project . . . . .	18
<b>7</b>	<b>Shared access requests</b>	<b>19</b>
<b>8</b>	<b>Upload runner</b>	<b>21</b>
8.1	File Upload . . . . .	21
8.2	File Download . . . . .	22
8.3	Container Download . . . . .	22

8.4	Copying a container . . . . .	22
<b>9</b>	<b>Python Modules</b>	<b>23</b>
<b>10</b>	<b>Testing</b>	<b>25</b>
10.1	Unit Testing . . . . .	25
10.2	User Interface Testing . . . . .	25
<b>11</b>	<b>Tools used in project</b>	<b>27</b>
11.1	Backend . . . . .	27
11.2	Frontend . . . . .	27
11.3	Tests . . . . .	28
11.4	Documentation . . . . .	28
<b>12</b>	<b>Indices and tables</b>	<b>29</b>

A Web UI object browser for object storage back-ends using Openstack Keystone for authentication (e.g. [CSC Pouta](#)). It uses federated login via [HAKA](#), via the endpoints provided by [OpenStack Keystone](#).

Out of the box the `swift-browser-ui` offers:

- UI for browsing [SWIFT](#) objects;
- support for additional features like uploading files >5GiB in size;
- support for federated authentication of an user with their HAKA credentials using OpenStack Keystone;
- UI based on [Vue.js](#) with [Buefy](#) framework;
- asynchronous web server.



# CHAPTER 1

---

## Setup Instructions

---

The program can be installed with pip from the git repository:

```
# Requires python >= 3.10
git clone git@github.com:CSCfi/swift-browser-ui.git
# Frontend files need to be separately built
cd swift_browser_ui_frontend && npm install -g pnpm@7 && pnpm install && pnpm run_
↪build && cd ..
pip install .
```

---

**Note:** The program uses external services that need to be present in order to enable all functionality, like sharing. These additional services can be found from the git repositories. The instructions for getting the services up and running can be found in their respective repositories, and partly under the *Deployment* section.

- <https://github.com/cscfi/swift-x-account-sharing>
  - <https://github.com/cscfi/swift-sharing-request>
  - <https://github.com/cscfi/swiftui-upload-runner>
- 

## 1.1 Environment Setup

---

**Hint:** The command line arguments can also be configured as environment variables, the environment variable syntax is documented in the python click documentation<sup>1</sup>, the shape of a variable could take the following forms:

- BROWSER\_\$ARGUMENT - affects every command;
  - BROWSER\_\$SUBCOMMAND\_\$ARGUMENT - affects subcommands e.g. start, install.
- 

Variables are depicted in the table below:

---

<sup>1</sup> <https://click.palletsprojects.com/en/7.x/options/#values-from-environment-variables>

Environment variable	De- fault	Description		
BROWSER_START_AUTH_ENDPOINT_URL		URL to use as the Openstack authentication backend		
BROWSER_START_PORT	8080	Port that the service will listen		
BROWSER_START_SET_ORIGIN_ADDRESS		Authentication return address to which WebSSO should redirect		
BROWSER_START_HAS_TRUST		Flag if the program is listed on the trusted_dashboards		
BROWSER_START_SHARING_ENDPOINT_URL		external URL for the container sharing backend		
BROWSER_START_REQUEST_ENDPOINT_URL		external URL for the shared access request backend		
BROWSER_START_RUNNER_ENDPOINT		internal URL for the upload, copy, download runner		
SWIFT_UI_SHARING_REQUEST_TOKEN		Token for signing the internal sharing & request API requests		
BROWSER_START_RUNNER_EXT_ENDPOINT		external URL for the upload runner service		
BROWSER_START_SHARING_INT_ENDPOINT_URL		internal URL / hostname of the sharing API		
BROWSER_START_REQUEST_INT_ENDPOINT_URL		internal URL / hostname of the request API		
LOG_LEVEL		set logging level e.g. INFO, DEBUG		

**Hint:** Authentication endpoint can also be specified with any openrc file, which can be usually downloaded from Openstack. The setup script from Openstack might ask for your password, but this isn't a required input and can be left empty.

### 1.1.1 Example environment variable files

For the Pouta test environment with NGINX TLS termination proxy in use:

```
export BROWSER_START_AUTH_ENDPOINT_URL="https://pouta-test.csc.fi:5001/v3"
export BROWSER_START_PORT="8081"
export BROWSER_START_SET_ORIGIN_ADDRESS="https://vm1950.kaj.pouta.csc.fi:8080/login/
↪webssso"
```

For the Pouta production environment for testing unsecurely without trust:

```
export BROWSER_START_AUTH_ENDPOINT_URL="https://pouta.csc.fi:5001/v3"
```

### 1.1.2 Setting up TLS termination proxy

The backend can be run in secure mode, i.e. with HTTPS enabled, but for scaling up a TLS termination proxy is recommended. Setting up TLS termination is outside the scope of this documentation, but a few useful links are provided along with the necessary configs regarding this service.<sup>2,3</sup>

## 1.2 Scaling up the service

The service runs in a single-threaded mode, since the library that's used for providing the back-end isn't multi-threaded. Therefore to completely use up a server's resources a multi-processed approach must be chosen. The easiest way to do this is to set up a reverse proxy, which can be run in the same server that acts as the TLS endpoint.

<sup>2</sup> <https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/>

<sup>3</sup> <https://docs.nginx.com/nginx/admin-guide/security-controls/terminating-ssl-http/>



The aiohttp documentation already gives us directions for the set-up<sup>4</sup> so they won't be provided here. In its current state the project should be configured to use TCP sockets in NGINX, so they're the directions to use in the aforementioned link. Also change the server run command to enable running the project as follows:

```
command=swift-browser-ui start --port=808%(process_num)s
```

## 1.3 Further reading and citations

---

<sup>4</sup> <http://docs.aiohttp.org/en/stable/deployment.html>



## CHAPTER 2

---

### Getting started

---

---

**Note:** Please note that the things related to project development aren't documented here, and everything on this page is only related to the running of the program

---

After the setup has been completed as illustrated in *Setup Instructions* the server can be used with the `swift-browser-ui` command, the command line options can be found below.

### 2.1 Command line interface

The project has a command line interface, that can be used to quickly test the frontend for different endpoints and usage cases. It provides basic functionality e.g. starting the server and specify a variety of different settings, detailed below:

```
swift-browser-ui --help
Usage: swift-browser-ui [OPTIONS] COMMAND [ARGS]...

Command line interface for managing swift-browser-ui.

Options:
  --version          Show the version and exit.
  -v, --verbose      Increase program verbosity.
  -D, --debug        Enable debug level logging.
  --logfile TEXT     Write program logs to a file.
  --help            Show this message and exit.

Commands:
  start             Start the browser backend and server.
```

### 2.1.1 Global arguments

The following command line arguments affect all of the commands in the application:

<b>--verbose</b>	Flag to increase program verbosity.
<b>--debug</b>	Enable program debug messages.
<b>--logfile FILE</b>	Save all program output to a file.
<b>--help</b>	Help on the CLI usage.
<b>--version</b>	Display the program version

### 2.1.2 The server startup

The following command line arguments are available for server startup.

```
swift-browser-ui start --help
Usage: swift-browser-ui start [OPTIONS]

Start the browser backend and server.

Options:
-p, --port INTEGER          Set the port the server is run on.
--auth-endpoint-url TEXT    Endpoint for the Openstack keystone API in use.
--has-trust                 Flag if the program is listed on the
                           trusted_dashboards in the specified address.
--set-origin-address TEXT   Set the address that the program will be
                           redirected to from WebSSO
--secure                   Enable secure running, i.e. enable HTTPS.
--ssl-cert-file TEXT        Specify the certificate to use with SSL.
--ssl-cert-key TEXT         Specify the certificate key to use with SSL.
--help                     Show this message and exit.
```

<b>--port PORT</b>	Set the port that the server will use.
<b>--auth-endpoint-url URL REQUIRED</b>	Set the endpoint that the program uses for authentication. The program cannot work without this.
<b>--set-origin-address TEXT</b>	Set the address that the program will be redirected to from WebSSO.
<b>--has-trust</b>	Toggle if the program has trust on the specified authentication endpoint, i.e. if the program has been listed on the respective Openstack keystone trusted_dashboard list. <sup>1</sup>
<b>--secure</b>	Enable HTTPS on the server, to enable secure requests if there's no TLS termination proxy.
<b>--ssl-cert-file TEXT</b>	Specify SSL certificate file. Required when running in secure mode.
<b>--ssl-cert-key TEXT</b>	Specify SSL certificate key. Required when running in secure mode.

---

<sup>1</sup> <https://docs.openstack.org/keystone/pike/advanced-topics/federation/webssso.html>

## CHAPTER 3

---

### Deployment

---

The recommended means of deployment for a production web server via a container image (e.g. Docker image). In this section we illustrate several means of building and running a the `swift-browser-ui` application via a Docker container image.

### 3.1 Dockerfile

Using vanilla docker in order to build the image - the tag can be customised:

```
$ git clone https://github.com/CSCfi/swift-browser-ui/
$ docker build -t cscfi/swift-ui .
$ docker run -p 8080:8080 cscfi/swift-ui
$ # or with environment variables
$ docker run -p 8080:8080 \
    -e BROWSER_START_AUTH_ENDPOINT_URL=https://pouta.csc.fi:5001/v3 \
    cscfi/swift-ui
```

### 3.2 Database for sharing functionality

Both `swift-x-account-sharing` and `swift-sharing-request` services need access to a PostgreSQL database in order to work. In a usual deployment this is done within a containerized stack. Necessary files to build a database container for testing can be found in the [deployment example repository](#). The file `init-project-db.sh` contains the necessary input to build the DB schema, and the same commands can be used to build the schema into an existing database server (as is the case when running on openshift using a base image for the database)

### 3.3 Sharing functionality back-end

Sharing functionality should be run by running it in a container. Easiest way to do this is to use the docker-compose fields provided in the [deployment example repository](#). The sharing functionality requires the following environment

variables to be present in order to work:

Environment variable	Default	Re- quired	Description
SWIFT_UI_API_AUTH_TOKENS	TOKENS	True	Comma separated list of master tokens that can be used for signing the API requests
SHARING_DB_NAME	swift-sharing		Name for the sharing functionality database
SHARING_DB_USER	sharing		User used in connecting to the sharing functionality database
SHARING_DB_HOST		True	Sharing functionality database address/hostname
SHARING_DB_PORT	5432		Sharing functionality database port
SHARING_DB_SSL	prefer		Sharing functionality database SSL MODE (production should be require)
SHARING_DB_PASSWORD		True	Sharing functionality database password

### 3.4 Shared access request back-end

Shared access request functionality should be run by running it in a container. Easiest way to do this is to use the docker-compose files provided in the [deployment example repository](#). The shared access request functionality requires the following environment variables to be present in order to work:

Environment variable	Default	Re- quired	Description
SWIFT_UI_API_AUTH_TOKENS	TOKENS	True	Comma separated list of master tokens that can be used for signing the API requests
REQUEST_DB_NAME	swiftrequest		Name for the shared access request functionality database
REQUEST_DB_USER	request		User used in connecting to the shared access request functionality database
REQUEST_DB_HOST		True	Shared access request functionality database address/hostname
REQUEST_DB_PORT	5432		Shared access request functionality database port
REQUEST_DB_SSL	prefer		Shared access request functionality database SSL MODE (production should be require)
REQUEST_DB_PASSWORD		True	Shared access request functionality database password

### 3.5 Upload runner back-end

SwiftUI upload runner should be run by running it in a container. Easiest way to do this is to use the docker-compose files provided in the [deployment example repository](#). The upload runner requires the following environment variables to be present in order to work:

Environment variable	De- fault	Re- quired	Description
SWIFT_UI_API_AUTH_TOKENS	TOKENS	True	Comma separated list of master tokens that can be used for signing the API requests
BROWSER_START_AUTH_ENDPOINT_URL			Openstack keystone endpoint for authentication – can also be specified with OS_AUTH_URL
OS_AUTH_URL			Openstack keystone endpoint for authentication – can also be specified with BROWSER_START_AUTH_ENDPOINT_URL

The authentication information can also be gotten through sourcing any Openstack credential v3 file, the password is not necessary as only the authentication endpoint information will be used.

## 3.6 Kubernetes Integration

For use with Kubernetes we provide YAML configuration. Further configuration files are provided in [deployment example repository](#)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    role: swiftui
  name: swiftui
  namespace: swiftui
spec:
  selector:
    matchLabels:
      app: swiftui
  template:
    metadata:
      labels:
        app: swiftui
        role: swiftui
    spec:
      containers:
        - image: cscfi/swift-ui
          imagePullPolicy: Always
          name: swiftui
          ports:
            - containerPort: 8080
              name: swiftui
              protocol: TCP
---
apiVersion: v1
kind: Service
metadata:
  name: swiftui
  labels:
    app: swiftui
spec:
  type: NodePort
  ports:
    - port: 8080
      targetPort: 8080
      protocol: TCP
      name: web
  selector:
    app: swiftui
```





In this section we would like to emphasize some of the core parts of the application and describe their inner-workings.

### 4.1 About login process

The program uses the WebSSO support provided by [Openstack](#), whenever the support has been implemented. At minimum the program requires the Openstack instance that it's supposed to be used with to implement the federated authentication, in which the non-WebSSO token delivery method can be used.

A collection of links to provide a recap of the things necessary to know about Openstack WebSSO implementation:

- [Federated authentication API in Openstack](#)
- [WebSSO details conveniently explained \(or how it works in the OS Horizon dashboard\)](#)
- [Openstack Horizon dashboard WebSSO guide](#)

The login process follows an *almost ordinary* process of federated authentication with SAML, but that is something we don't need to concern ourselves with – Openstack identity API takes care of that. The manual version works in much the same way, but the user is required to copy and paste the token themselves, since Openstack refuses to redirect to untrusted platforms.

Fig. 1: Sequence diagram of the login process.

### 4.2 API

---

**Hint:** The API has several endpoints, which are documented [here](#). The API can also be used with the convenience functions, located in the `api.js` file.

---

---

**Note:** All API queries expect an open session to Openstack, which means the queries towards Openstack are correctly scoped to the open project without further information in the API query. This of course requires a valid session token to be present with every API call.

---

The following flowchart gives a generic image of the possible responses from the API during normal usage.

Fig. 2: Flowchart of the simplified API execute routes upon query.

The api is documented in the api.yml file, that conforms to the OpenAPI specification (the file can be rendered with the [swagger editor](#)):

The user interface defaults to a container listing, showing all the containers for the default active project of the user. The localization can be changed from the button in the up-right corner.

### 5.1 User information page

Behind the **User information** button in the front page, a user information dashboard is displayed. The dashboard displays statistics about the current resource usage, e.g.

- Current billing unit consumption
- Amount of containers and objects in a project
- Total project data usage.

Additional information on different billing details is also provided, in the links contained in the dashboard bottom tile.

Fig. 1: Image of the user information dashboard in an example project.

### 5.2 Container page

The default front-page for the browser is the container listing, which will default to the first project that Openstack proposes. This page shows the containers available to be browsed, as well as general information about them. The container can be opened with a double-click, or if the table row's active, enter.

Fig. 2: Image of the container listing for an example project.

## 5.3 Object page

Any container can be opened, and the contents viewed. The object page shows information on the objects, e.g.

- The object name
- The object ETag
- A download link for the object
- Content type
- Last date of modification

Fig. 3: Image of the object listing for an example container.

Additional information is not shown by default, but can be opened with the chevron located in the beginning of each row.

Fig. 4: Image of the object listing, showing the additional details.

## 5.4 Non-whitelisted mode

When running a development environment that is not whitelisted to use the WebSSO for logins, the following login page will be displayed. The page is there to enable manual token delivery, since the server refuses to deliver it automatically to untrusted platforms. (i.e. copying and pasting the token)

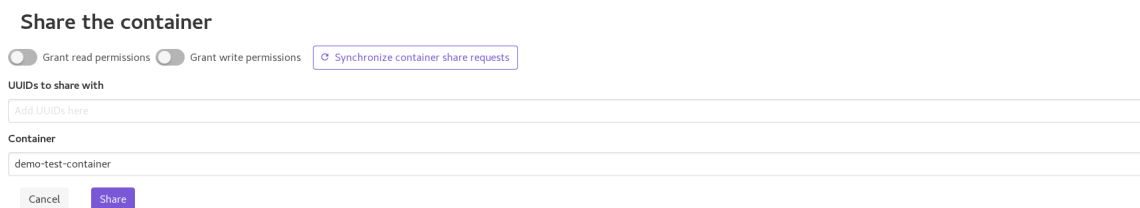
Fig. 5: Image of the manual token delivery login page.

## Sharing functionality

The UI provides a simple way of sharing containers between different projects, provided you know the project that requests sharing. (If not, see the documentation on sharing requests)

### 6.1 Sharing a container

A container can be shared from the “share” button in the UI, on the row of the container in the container listing. Clicking the button takes one to the container sharing view, in which the user needs to specify the project/projects the container is going to be shared to, and what rights to give. The view also contains a button to synchronize any requests for accessing the container, if any are present. In case the user doesn’t want these requests fulfilled, they can be removed from the tags that are inputted into the sharing view.



The screenshot shows a web form titled "Share the container". At the top, there are three controls: a toggle switch for "Grant read permissions" (which is turned on), a toggle switch for "Grant write permissions" (which is turned off), and a button with a circular arrow icon labeled "Synchronize container share requests". Below these is a section labeled "UIDs to share with" containing a text input field with the placeholder "Add UIDs here". Underneath that is a section labeled "Container" with a text input field containing the value "demo-test-container". At the bottom of the form are two buttons: a grey "Cancel" button and a purple "Share" button.

Fig. 1: Image of the container sharing view for an example container

### 6.2 Viewing containers shared from the project

Containers that have been shared from a particular project can be viewed by navigating to the “Shared” page in the application navbar. From this view the shared access can be revoked, a new share initiated, or existing access synchronized to the sharing back-end, thus enabling it to be queried from the back-end in the future.

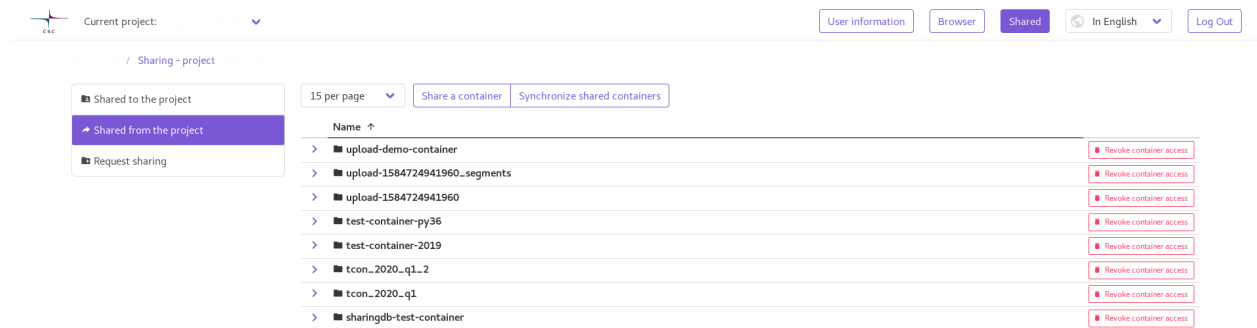


Fig. 2: Image of the view listing containers shared from the project

## 6.3 Viewing containers shared to the project

Containers that are shared to a particular project can be viewed by navigating to the “Shared” page in the application navbar. From this view the granted access can be viewed, and any container can be opened just like when using the normal container browsing view. All features available in the ordinary container view work, such as downloading, uploading (if write access is granted to the container) and copying the container.



Fig. 3: Image of the view listing containers shared to the project

## Shared access requests

The UI provides the possibility to request access to a container from a known project, which can be done via the shared access request page. This can be found under the “Shared” page in the navbar.

/ Sharing - project

- Shared to the project
- Shared from the project
- Request sharing**

Container / Identifier

The requested container name

Owner

The requested container owner **Request**

10 per page ▼

No shared containers have been requested for the project.

0-0 / 0 < >

Fig. 1: Image of the container access request page





---

## Upload runner

---

The UI provides an improved upload proxy / runner, that provides a possibility for more complicated operations not normally present on the object storage Web interface – these operations include e.g. uploading files into an object storage container, and downloading a whole container with API call.

### 8.1 File Upload

Files can be uploaded to an automatically generated container by drag’n’drop from the container listing page, or by using the upload button on top of the table. There’s no limit on how large the files uploaded can be, but browser performance puts a practical limit somewhere in the neighborhood of 10GiB.

Uploading files to a specific container can be done by opening the container, and uploading while the container is open. This can again be done either by drag’n’drop or using the upload button.

---

**Hint:** If complete relative file paths or folder structure is to be preserved, the only option for uploading is drag’n’drop. Only files can be uploaded using the upload button

---

---

**Hint:** Chrome is recommended as the browser of choice when uploading large files, as the File API on Chrome is better implemented. Firefox tends to have issues on files >5GiB, especially with multiple files. Safari is not supported, but should work without problems – same issues present with Firefox apply.

---

---

**Hint:** Mobile devices are not supported for file uploads, but can work. This is, however, not guaranteed.

---

## 8.2 File Download

The upload runner is used to provide file downloads from shared containers in a similar manner to the way the downloads work from containers owned by the project that is currently active. To the user the download continues to be a simple download link

## 8.3 Container Download

Full containers can be downloaded from the UI using the download button either on the table row in the container listing, or a download button on the top of the table when viewing an open container. Downloading whole containers works the same in both owned and shared containers. The runner archives the container while the download is taking place, in order to prevent additional waiting for an archiving operation to finish. This has the added benefit of not requiring any intermediary storage for the archiving operation on the server side.

---

**Hint:** Due to the fact that the archive size can't be precisely calculated when archiving on the fly, the server is unable to provide a progress bar for a container download. A rough estimate can be generated by calculating the time value from the container size visible in the container listing using the available speed of the connection.

---

## 8.4 Copying a container

Containers can be copied using the copy button, either on the row of the container in the container listing view, or on top of the table when viewing a specific container. The copy operation can only be performed to a fresh container, to prevent accidental data loss in case of an incomplete copy operation on an object. User can also copy a shared container to the currently activated project.

---

**Hint:** The copy operation takes a long time, and is run in the background. The UI navigates back to the normal view after copying is initiated. Thus, the copy operation is eventually consistent.

---

---

**Hint:** The runner validates every copied object against the file checksum present in the object storage backend – thus, if the object is present in the newly created container, it's guaranteed to have been successfully copied over.

---

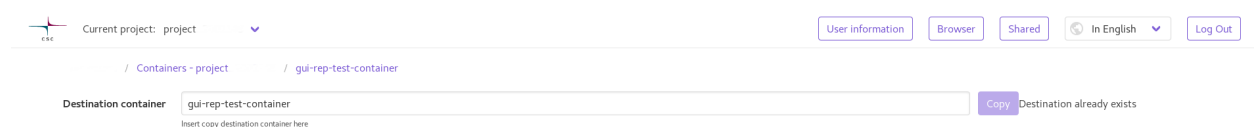


Fig. 1: Image of the container replication page when trying to copy over an existing container

## CHAPTER 9

---

### Python Modules

---

---

swift_browser_ui.ui._convenience
swift_browser_ui.ui.api
swift_browser_ui.ui.discover
swift_browser_ui.ui.exceptions
swift_browser_ui.ui.front
swift_browser_ui.ui.login
swift_browser_ui.ui.middlewares
swift_browser_ui.ui.misc_handlers
swift_browser_ui.ui.server
swift_browser_ui.ui.settings
swift_browser_ui.ui.shell
swift_browser_ui.ui.signature

---

---

swift_browser_ui.sharing.bindings.
bind
swift_browser_ui.sharing.api
swift_browser_ui.sharing.db
swift_browser_ui.sharing.server
swift_browser_ui.sharing.shared

---

---

swift_browser_ui.request.bindings.
bind
swift_browser_ui.request.api
swift_browser_ui.request.db
swift_browser_ui.request.server

---

---

<code>swift_browser_ui.upload.api</code>
<code>swift_browser_ui.upload.auth</code>
<code>swift_browser_ui.upload.common</code>
<code>swift_browser_ui.upload.download</code>
<code>swift_browser_ui.upload.replicate</code>
<code>swift_browser_ui.upload.server</code>
<code>swift_browser_ui.upload.upload</code>

---

[genindex](#) | [modindex](#)

**Note:** Unit tests and integration tests are automatically executed with every PR

### 10.1 Unit Testing

In order to run the unit tests, security checks with [bandit](#), Sphinx documentation check for links consistency and HTML output and [flake8](#) (coding style guide) [tox](#). To run the unit tests and UI tests in parallel use:

```
$ tox -p auto
```

To run environments separately use:

```
$ # list environments
$ tox -l
$ # run flake8
$ tox -e flake8
$ # run bandit
$ tox -e bandit
$ # run docs
$ tox -e docs
```

### 10.2 User Interface Testing

User Interface tests are developed using [cypress](#), and the tests are developed for both Firefox and Chrome web browsers.

```
$ cd swift_browser_ui_frontend/
$ npm install -g pnpm@7
```

(continues on next page)

(continued from previous page)

```
$ pnpm install
$ pnpm run build
$ cd ..
$ pnpm install cypress
$ pnpm exec cypress open
```

### 11.1 Backend

The backend is written in Python, requiring at minimum Python version 3.6.8, but is tested with 3.7 and 3.8 as well. Additionally the following libraries are used in the program development:

- aiohttp for the API server
- uvloop for increasing server performance
- keystoneauth1 for authenticating with Openstack
- python-swiftclient for communicating with Openstack Swift API
- cryptography for encrypting session cookies
- click for quickly providing a CLI for the server

### 11.2 Frontend

The frontend is written as an SPA (Single Page Application), in *ES6* Javascript. The frontend code is not tested for cross-compilation to ES5. The following libraries are used in writing the frontend:

- Vue.js for site functionality and DOM manipulation
- Vue router for site routing support
- Vue i18n for language support
- Buefy for site styling and components
- Buefy also comes with bulma for css framework
- Lodash for mostly debouncing functions to improve responsivity

## 11.3 Tests

Tests are written to be run with `Pytest`. The following libraries are used in writing the tests:

- `tox` for test automation
- `cypress` for UI test automation
- `pytest-timeout` for timing out UI tests, which can hang when failing

UI tests also require the WebDrivers for Chrome and Firefox, if tests are to be run locally.

- `WebDriver` for Chrome
- `WebDriver` for Firefox

## 11.4 Documentation

The documentation is automatically built with `sphinx`

### 11.4.1 Charts

The charts in documentation are made with `Dia`. The program is old fashioned, but versatile and can be installed without adding repositories, with the added benefit of not requiring the use of browser tools for making the charts. Charts are located in `docs/charts`, and the exported vector graphics file is linked into the documentation image directory.

---

**Note:** `swift-browser-ui` and all its sources are released under *MIT License*.

---



## CHAPTER 12

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`