Shibboleth IdP Metadata Management GUI

Ready for testing!

Tell us what you think about Shibboleth IdP UI on the feedback form.

Click to see the overview fact sheet

See the recorded webinar here!

Project Overview

A GUI (Graphical User Interface) is being developed for the Shibboleth IdP, funded by Internet2/Trust and Identity and Unicon and with the guidance of community architects including Scott Cantor, whose initial focus is on managing metadata and metadata filters (using entity attributes). This Shibboleth Metadata Management GUI will be referred to below as the Shibboleth UI for short.

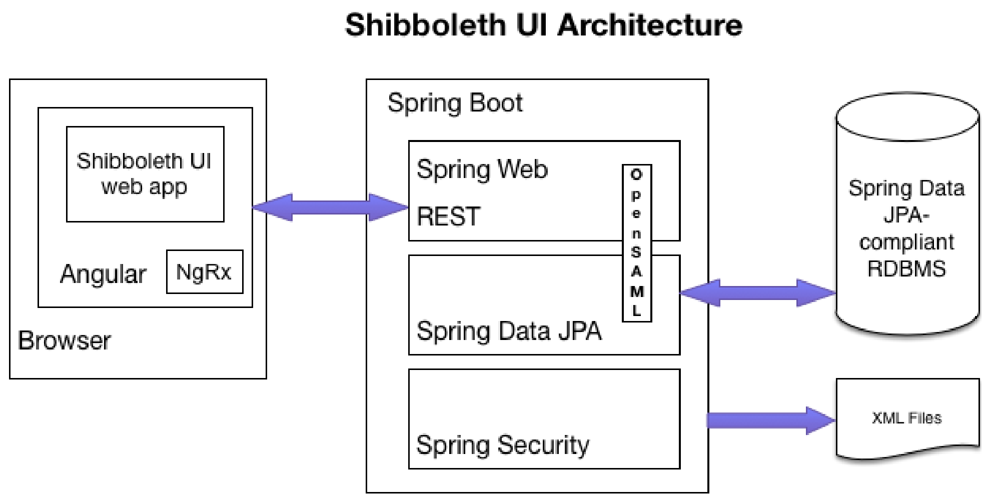
Why metadata? One of the primary reasons that the Shibboleth IdP is so widely adopted is its ability to leverage metadata, including large metadata aggregates found in many of the eduGAIN member federations, to support large-scale multilateral federated identity. The Shibboleth development team, in accordance with increasing community needs, is expanding the range of behaviors/settings of the Shibboleth IdP that can be controlled through metadata. In particular, as part of the next release of the Shibboleth IdP version 3.4, support is added for entity attributes that can trigger various relying party override settings.

The Shibboleth IdP already supports the ability to craft attribute release (filter) rules that are triggered by entity attribute settings, and other settings (such as NameID ones) can be similarly "activated". This GUI begins to make it much easier to take advantage of the metadata-driven capabilities of the Shibboleth IdP.

This new Shibboleth UI enables the Identity Administrator to create Service Provider (SP) metadata files from "scratch", or import metadata for an SP from a file or URL, and add entity attributes to that metadata that can impact relying party settings such as required authentication context, what is signed, signature algorithm, encryption, forced authentication, etc. Moreover, if the Identity Administrator adds the right template to the attribute-filter.xml file, the UI also enables the management of which attributes may be released to that SP. The Shibboleth UI also enables the Identity Administrator to similarly add entity attributes to specific SPs from a file-backed HTTP metadata provider such as the InCommon metadata aggregate. SP entries in such a metadata provider aggregate are modified by creating metadata filters that will "annotate" the (e.g. InCommon) aggregate-provided SP metadata. Through these filters, aspects of the IdP's behavior and how and what is contained within the SAML response to a given SP, can be managed thru the UI, just as it can for individually managed metadata files (sources).

Further iterations of the Shibboleth UI are planned, which will continue to expand on the range of metadata provider and filter types supported thru the UI, and allow defining and managing custom entity attributes. How the UI expands beyond that is a conversation for the community to have as this work moves forward.

Architecture



The Shibboleth UI is written as a Spring Boot application, with Angular used for the front-end (UI). And it relies on a persistent relational database for storing the configuration data created through the UI, and then writes out XML files to disk. The XML files include the individual metadata sources (files) being managed through the UI, which by default will be written into a generated/ directory with filenames of the SHA1 hash of the entityId. And it will write out a metadata-providers.xml file that contains all metadata providers defined through the UI, along with any filters that have been created for of those metadata providers.

Assumptions

The Shibboleth UI is specifically written to work with new metadata-driven configuration support that Shibboleth Identity Provider 3.4 has introduced. So the assumption is that you will use the Shibboleth UI in conjunction with IdP 3.4. And that the underlying configuration of your Shibboleth IdP 3.4 instance (s) will have been modified to **activate** metadata-driven configuration support, include the needed LocalDynamic metadata provider definition and metadata /generated directory (for the individual metadata sources (files) managed through the UI) etc. If you use the TIER-provided Docker Shibboleth Identity Provider 3.4 package, these IdP configuration changes will have already been made.

If you are not going to use the TIER Shibboleth IdP 3.4.x Docker distribution, then you would need to make these IdP 3.4.x configuration changes yourself. The following page discusses the range of changes that are needed to take full advantage of this Shibboleth UI.

Deployment

There is a TIER Docker Shibboleth UI testbed deployment one can use, that provides the full environment one needs to explore and gain experience with the Shibboleth UI. This Dockerized testbed version is available at https://github.internet2.edu/docker/shib-ui . That testbed follows the TIER packaging guidelines. It relies on supervisord, and includes the Shibboleth UI, a Shibboleth IdP (with a shared filesystem between the Shibboleth UI and Shibboleth IdP), an LDAP server as the base credential/attribute store for the IdP, the TIER Maria DB image for the UI's persistent database, and the TIER Beacon configuration.

Beyond becoming familiar with the Shibboleth UI by using the testbed, note that the Shibboleth UI has been built to be deployed as its own web app. It would run in the same servlet engine as the IdP, or it could run in a separate one. The key link is that files generated by the UI need to end up making it into the IdP's configuration (conf/) directory, however you want to make that happen. That could be a NFS/SMB file sharing approach, it could be rsync or scp or something similar, or you could put the files into a repo and then pull out of the repo into the IdP.

Documentation

The latest documentation, including the top-level README included below, can be found in the TIER Shibboleth Ui code repository, primarily in its *docs/* directory. And for deployment using the TIER Packaging (Docker container) and a testbed environment (as noted in the above Deployment section), see the documentation in the TIER Docker Container Packaging for Shibboleth UI repository,

A version of each of the primary documents has been copied into this wiki as child pages to this page (with the top-level README included below.)

- Installation and Basic Configuration (also includes commands to start as a quick start demo)
- Default Properties
- Database Configuration
- Metadata Sources (individual one-entityID-per files)
- Metadata Providers
- Customizations and Custom Development
- Internationalization Guide

Internet2 Shibboleth UI Repositories

- TIER Shibboleth Ui code repository

- TIER Docker Container Packaging for Shibboleth UI
- Here is the top-level README from the TIER Shibboleth Ui code repository. Note that to be sure you are viewing the latest such version of it, you should go to the TIER Shibboleth Ui code repository linked to above.

shibui

For more information, see `docs`

Requirements

- Java 8 (note that ONLY Java 8 is supported at this time)

Running

There are currently 2 ways to run the application:

1. As an executable
2. deployed in a Java Servlet 3.0 container

Note that some features require encoded slashes in the URL. In tomcat (which is embedded in the war), this can be allowed with:

```
-Dorg.apache.tomcat.util.buf.UDecoder.ALLOW_ENCODED_SLASH=true
```

In Apache HTTPD, you'll need something like:

```
<VirtualHost *:80>
    AllowEncodedSlashes NoDecode
    ServerName shibui.unicon.net
    ProxyPass / http://localhost:8080/ nocanon
    ProxyPassReverse / http://localhost:8080/
</VirtualHost>
```

Running as an executable

`java -jar shibui.war`

For complete information on overriding default configuration, see [https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-external-config.html].

Deploying as a WAR

The application can be deployed as a WAR file in a Java Servlet 3.0 container. Currently, the application must be run in the root context.

To override default configuration, see [https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-external-config.html]. The easiest way to do this in a servlet container is through the use of system properties

Authentication

Currently, the application is wired with very simple authentication. A password for the user `user` can be set with the `shibui.default-password` property. If none is set, a default password will be generated and logged:

```
Using default security password: a3d9ab96-9c63-414f-b199-26fcf59e1ffa
```

Default Properties

This is a reflection of the default `application.properties` file included in the distribution. Note that lines beginning with # are commented out.

# shibui

For more information, see `docs`

## Requirements

* Java 8 (note that ONLY Java 8 is supported at this time)

## Running

There are currently 2 ways to run the application:

1. As an executable
1. deployed in a Java Servlet 3.0 container

Note that some features require encoded slashes in the URL. In tomcat (which is embedded in the war), this can be
allowed with:

```
-Dorg.apache.tomcat.util.buf.UDecoder.ALLOW_ENCODED_SLASH=true
```

In Apache HTTPD, you'll need something like:

```
<VirtualHost *:80>
    AllowEncodedSlashes NoDecode
    ServerName shibui.unicon.net
    ProxyPass / http://localhost:8080/ nocanon
    ProxyPassReverse / http://localhost:8080/
</VirtualHost>
```

### Running as an executable

`java -jar shibui.war`

For complete information on overriding default configuration, see [https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-external-config.html].

### Deploying as a WAR

The application can be deployed as a WAR file in a Java Servlet 3.0 container. Currently, the
application must be run in the root context.

To override default configuration, see [https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-external-config.html].
The easiest way to do this in a servlet container is through the use of system properties

## Authentication

Currently, the application is wired with very simple authentication. A password for the user `user`
can be set with the `shibui.default-password` property. If none is set, a default password
will be generated and logged:

```
Using default security password: a3d9ab96-9c63-414f-b199-26fcf59e1ffa
```

## Default Properties

This is a reflection of the default `application.properties` file included in the distribution. Note that lines
beginning with `#` are commented out.

```
# Server Configuration
#server.port=8080
```

```
# Logging Configuration
#logging.config=classpath:log4j2.xml
#logging.level.org.springframework.web=ERROR

# Database Credentials
spring.datasource.username=shibui
spring.datasource.password=shibui

# Database Configuration H2
spring.datasource.url=jdbc:h2:mem:shibui;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE
spring.datasource.platform=h2
spring.datasource.driverClassName=org.h2.Driver
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.h2.console.enabled=true


# Database Configuration PostgreSQL
#spring.datasource.url=jdbc:postgresql://localhost:5432/shibui
#spring.datasource.driverClassName=org.postgresql.Driver
#spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect

#Maria/MySQL DB
#spring.datasource.url=jdbc:mariadb://localhost:3306/shibui
#spring.datasource.driverClassName=org.mariadb.jdbc.Driver
#spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MariaDBDialect

#Tomcat specific DataSource props. Do we need these?
#spring.datasource.tomcat.maxActive=100
#spring.datasource.tomcat.minIdle=10
#spring.datasource.tomcat.maxIdle=10
#spring.datasource.tomcat.initialSize=50
#spring.datasource.tomcat.validationQuery=select 1

# Liquibase properties
liquibase.enabled=false
#liquibase.change-log=classpath:edu/internet2/tier/shibboleth/admin/ui/database/masterchangelog.xml

# Hibernate properties
# for production never ever use create, create-drop. It's BEST to use validate
spring.jpa.hibernate.ddl-auto=create
spring.jpa.hibernate.naming.implicit-strategy=org.hibernate.boot.model.naming.
ImplicitNamingStrategyJpaCompliantImpl
spring.jpa.show-sql=false
spring.jpa.properties.hibernate.format_sql=false

spring.jpa.hibernate.use-new-id-generator-mappings=true

shibui.metadata-dir=/opt/shibboleth-idp/metadata/generated
shibui.logout-url=/dashboard

spring.profiles.active=default

# Password for the default user. If not set, a password will be generated at startup
#shibui.default-password=
```
```

## Additional resources

Shibboleth Metadata Management GUI - Requirements
```

Recordings of Demonstrations

Design & Architecture

SHIBUI Jira Project

tier-shib-ui slack channel