sb11-test-registration

Test Registration enables users to upload and maintain various data pertinent to the SBAC system including:

- Organizational hierarchy
- Students
- Accommodations
- SBAC system users
- Assessments
- Eligibility

Other functionality includes importing assessments from Test Spec Bank, user provisioning and administration, and integrations with Test Delivery, SSO, and the Data Warehouse.

Module Overview

These are the different modules that comprise the Test Registration code.

Domain

Contains all of the domain objects used in the system. Also contains high level validation and search objects.

Persistence

All interaction with the database along with business logic and related validation.

REST

All REST endpoints and code relating to uploading data files.

Webapp

The Test Registration UI.

Setup

In general, building the code and deploying the two WAR files is a good first step. Test Registration, however, has a number of other steps that need to be performed in order to fully setup the system.

Tomcat

Like other SBAC applications, Test Registration must be setup with active profiles and program

management settings.

Typical profiles for the `-Dspring.profiles.active` include:

- `progman.client.impl.integration` Use the integrated program management
- `progman.client.impl.null` Use the program management null implementation
- `mna.client.integration` Use the integrated MnA component
- `mna.client.null` Use the null MnA component
- `rabbit` Connect to a running RabbitMq server
- `no-rabbit` Do not connect to a RabbitMq server
- `tsb.client.impl.integration` Integrate with Test Spec Bank
- `test-write-dw-gen-data` Write generated Data Warehouse files to /tmp

Active profiles should be comma separated.

Set `-Dprogman.baseUri` to the URI for the REST layer of the Program Management instance to connect to.

Set `-Dprogman.locator` to the locator string needed to find the Test Registration properties to load.

User Change / SSO SFTP Setup

The SSO integration periodically checks to see if any user changes have occurred. If so, it will generate an XML file which is then SFTP'd to an OpenAM server where it is consumed for user provisioning or maintenance.

There are a few properties that must be setup in program management to complete the setup for this integration.

- `testreg.user.export.frequency.milliseconds` Periodicity of XML generation in milliseconds
- `testreg.sftp.port` SFTP port of server to connect to
- `testreg.sftp.dir` Directory on SFTP server to copy file to
- `testreg.sftp.pass` Password for authentication
- `testreg.sftp.user` User for authentication
- `testreg.sftp.host` SFTP host to connect to
- `testreg.sso.filename.suffix` Suffix to append to file when SFTP'd

Data Warehouse Integration Setup

The data warehouse integration is a periodic export of data to a data warehouse landing zone. It incorporates pulling data out of the Test Registration database, compressing it, encrypting it, and then SFTPing the data. There are numerous items to setup in order for it to all work.

GPG Keys

All files sent to the data warehouse must be encrypted and signed using GPG. In order to do this, each Test Registration deployment must generate their own GPG key pair.

GPG is usually pre-installed on most UNIX systems. If generating keys on a Mac system install GPG Suite [https://gpgtools.org/]. If generating keys on a Windows system GPG4Win [http://www.gpg4win.org/] can be used.

Follow the directions in the software on your system to generate a key pair. In order to generate a key pair on a UNIX system the following can be used:

`gpg --gen-key`

You will be asked to choose a type:

```
Please select what kind of key you want:
    (1) RSA and RSA (default)
    (2) DSA and Elgamal
    (3) DSA (sign only)
    (4) RSA (sign only)
Your selection?
```

The default key will work fine.

The next question that is asked is to choose key size. The default key size is ok.

```
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048)
```

Now choose how long the key is valid for. For testing purposes it is ok to choose that it will never expire.

```
Requested keysize is 2048 bits
Please specify how long the key should be valid.
     0 = key does not expire
     <n>  = key expires in n days
     <n>w = key expires in n weeks
     <n>m = key expires in n months
     <n>y = key expires in n years
Key is valid for? (0)
```

A name and email address is then requested in order to build a user id to identify the key. For

testing purposes these can be fake values.

```
You need a user ID to identify your key; the software constructs the user ID
from the Real Name, Comment and Email Address in this form:
"Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"

Real name: Test Tester
Email address: test@test.com
Comment: This is a test account
You selected this USER-ID:
    "Test Tester (This is a test account) <test@test.com>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit?
```

Choose (O)kay to continue.

Next choose a passphrase to protect your secret key.

```
You need a Passphrase to protect your secret key.

Enter passphrase:
```

After this, GPG needs to gather entropy in order to generate a more random key. On a system with a GUI and mouse entropy is easy to generate just by moving the mouse around the screen. If logged in remotely to a UNIX system, generating entropy will be a little bit tougher. One way to generate the needed entropy is to open up another SSH session to the remote server and running one or both of the following:

`ls -R /`

`find / -type f`

After the key generates GPG will output something like this:

```
gpg: key 00168441 marked as ultimately trusted
public and secret key created and signed.

gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
pub   2048R/00168441 2014-03-05
      Key fingerprint = 573D 9B85 618A 1FA8 C61A  B07A 4D57 69AE 0016 8441
uid                  Test Tester (This is a test account) <test@test.com>
sub   2048R/F288019A 2014-03-05
```

The key pair is now created.

The public key that was created needs to be exported so that it can be given to the data warehouse. The export can be done by:

`gpg --export -a "test@test.com" > public.key`

The exported key is an ascii file.

The data warehouse should send a public key to Test Registration. This key should be imported into the keychain on the Test Registration server.

`gpg --import dw.public.key`

SSH Key

An SSH key pair will need to be generated for SFTP authentication. It is easiest to generate the SSH key on the server that the SFTP will take place on.

To generate a key pair log into the remote server and change the user to the correct account that will be SFTPing.

Now use:

`ssh-keygen`

Which will ask for a file to save the key in.

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ubuntu/.ssh/id_rsa):
```

The default is usually ok unless the user account has multiple ssh keys.

Now ssh-keygen asks for a passphrase. Usually a passphrase is strongly suggested. However, in the case that the application is performing a system to system automated file transfer it is

probably ok to just press enter to not have a passphrase.

`Enter passphrase (empty for no passphrase):`

You'll see some information about the key and it's now been generated.

```
Your identification has been saved in /home/ubuntu/.ssh/id_rsa.
Your public key has been saved in /home/ubuntu/.ssh/id_rsa.pub.
The key fingerprint is:
1a:3c:74:09:65:39:8a:b1:c2:48:2e:a3:6b:8d:80:aa ubuntu@jenkinsmaster01
The key's randomart image is:
+--[ RSA 2048]----+
|      ..o.       |
| .  .  oo.       |
|oo   +..o.       |
|+.o oo..         |
|+. .  + S        |
|+      +         |
|o.o   .          |
|oo .             |
|E                |
+-----------------+
```

The public key that was generated needs to be given to the data warehouse so that they can correctly setup security for the landing zone that Test Registration will SFTP to. The path of the generated public key is displayed above.

Password File

A property file needs to be created that contains the passphrase for the secret GPG key that was generated above. This file should contain one line like:

`testreg.secret.passphrase=testkey`

The value of that passphrase should be whatever was typed in when the GPG key was generated. Place this file somewhere in the file system that is secure. The only system user that should have access to this file is the user that runs Tomcat.

Tomcat Parameter

Tomcat needs to know where the secret passphrase file is. In order to tell Tomcat about it, a -D parameter needs to be added to the Tomcat command line at startup. It should look something like:

`-Dtestreg.secret.passphrase.file=/pathto/testregkeypass.properties`

The parameter name must be testreg.secret.passphrase.file.

Program Management Properties

There are number of other properties that need to be added to Program Management in order to successfully run the data warehouse integration.

- `gpgKeyring.public.location` location of gpg public keyring, usually found in .gnupg/pubring.gpg
- `gpgKeyring.secret.location` location of gpg secret keyring, usually found in .gnupg/secring.gpg
- `testreg.secret.key.userid` email address for test reg key
- `landingzone.public.key.userid` email address for landing zone key
- `dw.export.callback.url` callback url for data warehouse
- `dw.host` sftp host for landing zone
- `dw.private.key.loc` file path of private key for authenticating to landing zone
- `dw.port` sftp port
- `dw.user` user to connect to landing zone as
- `dw.remote.dir` remote sftp directory to write to
- `dw.gpgfile.prefix` filename prefix for generated file

Application Settings

In the running Test Registration application there is a link at the top of the screen called Setting. There are two settings on that page that need to be set in order for the Data Warehouse integration to work correctly.

The first is `Test Reg System Id`. If nothing is set in the text field a button should be displayed next to the text field that will generate a random GUID to use as the system id. Press the button to generate the GUID.

The second setting is `Assessment Eligibility types for export to Data Warehouse`. This is a multi-select list that allows for the setup of only specific types of assessments to control the data warehouse export. Select the types of assessments that should influence the export: summative, formative, and/or interim.

Be sure to save any settings changes that are made.

Field Name Crosswalk Setup

Each field that is part of an upload file has an associated field name that is displayed in various locations on the UI. It is possible to change the display names of the fields.

The initial mapping of the field names is kept in the REST module in the `/src/main/resources/spring/entity-crosswalk-context.xml` file. This can be modified and the WAR container restarted to pickup changes.

Organizational Hierarchy Renaming (Entity)

The application allows for the renaming of the levels of the organizational (entity) hierarchy. For example, the lowest level of the hierarchy is an institution. Many deployments may wish to change this to always display "school" rather than "institution".

The renaming of the levels in the hierarchy is part of component branding in the program management component. Two steps are necessary: adding a tenant and then configuring the branding for the tenant and component.

If the deployment of Test Registration is a state level deployment, all users who log into the system should ultimately have tenancy that resolves to the state for which the system has been deployed. If, however, the Test Registration deployment truly is multi-tenanted, then this configuration must be performed for each tenant. Component branding is resolved from lower to higher in the tenant hierarchy.

Add a Tenant

In the Program Management application, choose Manage Tenants to create a tenant if it has not been created. Make sure that the tenant has a subscription to the Test Registration component. If a Test Registration component has not yet been created go to Manage Components to create one.

Configure Component Branding

To setup the names go to Configure Component Branding. All of the entries are of type `Property`. The list of properties to configure are:

- Assessment
- Assessments
- User
- Users
- Institution
- GroupOfDistricts
- GroupOfInstitutions
- GroupOfStates
- Student
- Students
- Institutions
- StudentGroups
- District
- State
- Accommodations
- Eligibility

- Student Group

For each property the value reflects what will be seen in the UI.

Other Miscellaneous Setup

In the Test Registration UI there is a link at the top of the screen called Setting. The settings page has a few items that should be configured before users use the system:

- `Client Name` The name of the top level Client for this deployment. This is bootstrapped from Program Management properties, but can be changed here.
- `Time Zone` The effective time zone for this deployment. Choose from the drop down list.
- `Share Student Identity Data` This checkbox turns on and off the choice to share student identity data. If selected, then student names, birth dates, and SSID will be shared out through external interfaces. If turned off, then this data will not be shared.
- `Hide 'Group Of' Entity Levels` This must be configured before the system is used. Allows the system to completely hide the "Group of" levels of the organizational hierarchy. Once hidden, the system will not allow any functions to be performed for those levels of the hierarchy.

Other Dependencies and Progman Properties

There are many other properties that need to be set in Program Management so that Test Registration will function correctly.

MNA Properties

- `testreg.mna.description` Component name string to pass to M&A
- `mna.registrationUrl` The M&A registration URL (/jmx/registration)
- `mna.registrationDelayInSeconds` Number of seconds to wait before attempting to register Test Registration with M&A
- `testreg.mna.healthMetricIntervalInSeconds` How often to send heartbeat metric in seconds
- `mna.mnaUrl` The URL of the M&A REST api (/rest/)
- `mna.mnaDeleteComponentUrl` The URL for deleting a component from M&A (/jmx/registration/{server}/{node}/{component})
- `testreg.mna.availability.metric.email` Default email to send alerts to

RabbitMq Properties

- `rabbitmq.vhost` The name of the RabbitMq ghost to connect to. The default vhost is "/", but RabbitMq could be configured to use something different.
- `rabbitmq.password` Password to authenticate to RabbitMq with

- `rabbitmq.username` Username to connect to RabbitMq as
- `rabbitmq.host` RabbitMq server hostname

Security/SSO Properties

- `testreg.security.idp` The URL of the SAML-based identity provider (OpenAM)
- `testreg.security.profile` The name of the environment the application is running in. For a production deployment this will most likely be "prod". Must match the profile name used to name metadata files.
- `component.name` The name of the component that this Test Registration deployment represents. This must match the name of the component in Program Management and the name of the component in the Permissions application
- `permission.uri` The base URL of the REST api for the Permissions application

Mongo Properties

- `testreg.mongo.hostname` A comma delimited list of Mongo hostnames
- `testreg.mongo.port` The Mongo port
- `testreg.mongo.username` Username to connect to Mongo as
- `testreg.mongo.password` Password for Mongo authentication
- `testreg.mongo.dbname` The database name for the Test Registration database

Other Miscellaneous Properties

- `client` The name of the top-level client entity for this deployment
- `tsb.tsbUrl` The base URL for the REST api of Test Spec Bank
- `language.codes` Comma delimited list of valid language codes
- `testreg.minJs` Flag for JavaScript minification, true to minify
- `testreg.rest.context.root` The server relative context root for the Test Registration REST WAR. Should start and end with a slash.

RabbitMq Installation

RabbitMq is used as a message queueing solution to queue events relating to eligibility so that eligibility calculations can occur asynchronously in the background.

RabbitMq can be downloaded from the RabbitMq website or installed using a package manager.

Installation is straightforward. Be sure to install the RabbitMq Management plugin in order to use the front-end UI manager.

Other tasks to perform after installation:

- Create a new vhost. Vhosts can have separate security and permissions attached to them and provide a nice way to separate functionality on your RabbitMq server. It's not required to create a new vhost. Make sure that if a new vhost is created, that the value is put into the `rabbitmq.vhost` program management property. By default the system vhost is "/".
- Add a new system-wide administrator user. By default the user will be for the "/" vhost. Make sure that it has the "administrator" tag so that RabbitMq recognizes that the user is an administrator. Configure, read, and write privileges default to ".*", but can be set to whatever is needed.
- Add a user for the Test Registration application to connect as. Make sure it's configured for the correct vhost. Be careful with privileges. Configure privileges are necessary for the application to be able to dynamically create queues. Use this user and password for the program management properties `rabbitmq.username` and `rabbitmq.password`.
- Remove (or disable) the default guest user. Logins with the guest user should not be allowed in a production environment.

The management console web UI can be found at port 15672 on the host RabbitMq is installed to.

Build

These are the steps that should be taken in order to build all of the Test Registration related artifacts.

Pre-Dependencies

- Mongo 2.0 or higher
- Tomcat 6 or higher
- Maven (mvn) version 3.X or higher installed
- Java 7
- Access to sb11-shared-build repository
- Access to sb11-shared-code repository
- Access to sb11-security repository
- Access to sb11-rest-api-generator repository
- Access to sb11-program-management repository
- Access to sb11-monitoring-alerting-client repository
- Access to sb11-test-spec-bank repository

Build order

If building all components from scratch the following build order is needed:

- sb11-shared-build
- sb11-shared-code
- sb11-rest-api-generator
- sb11-program-management
- sb11-monitoring-alerting-client
- sb11-security
- sb11-test-spec-bank
- sb11-test-registration

Dependencies

Test Registration has a number of direct dependencies that are necessary for it to function. These dependencies are already built into the Maven POM files.

Compile Time Dependencies

- Apache Commons IO
- Apache Commons Collections
- Apache Commons Beanutils
- SB11 Shared Code
- SB11 Shared Security
- Spring Core
- Spring WebMVC
- Spring Expression
- Spring AOP
- Spring Aspects
- Spring Context
- Spring Beans
- Spring Tx
- Spring Web
- Jackson Datatype Joda
- Hibernate Validator
- Logback Core
- Logback Classic
- SLF4J
- JCL over SLF4J
- Apache Commons Jexl
- XStream
- Spring Data MongoDb
- Spring Integration SFTP
- Spring Integration Core
- Spring Integration AMQP

- Spring Integration File
- BouncyCastle Provider (bcprov-jdk15on)
- BouncyCastle GPG/PGP (bcpg-jdk15on)
- Apache Commons Compress
- Super CSV
- Apache Commons File Upload
- CGLib
- JSTL
- Apache POI
- Apache POI Excel (poi-excelant)
- Apache POI OOXML (poi-ooxml)
- Google Guava
- Open CSV

Other dependencies such as Mongo DB driver, Joda Time, and Jackson are pulled in through other dependencies.

Test Dependencies

- Spring Test
- Hamcrest
- JUnit 4
- Flapdoodle
- Mockito
- Podam
- Log4J over SLF4J

Runtime Dependencies

- Servlet API
- Persistence API