SB11-TEST-ITEM-BANK Backlog

(as of 01/31/2013)

The following captures the known outstanding functionality left for Test Item Bank as well as any known remaining technical tasks. The schedule for completion of the backlog is not determined. External dependencies have been noted where given as reason for being in the backlog. Please note that the component cannot be considered finished until this remaining back log is completed. Scheduling for completion of these tasks will need to be coordinated by all involved stakeholders.

Authentication/Authorization

sb11-test-item-bank components can be deployed on both windows and unix/linux flavors:
This task remains as an open issue until such time the SSO/Authorization mechanism for Smarter Balanced RFP#11 is decided upon and delivered. Based on the following assumptions, the level of effort remaining will be a matter of applying the authentication pattern to the web application that hosts the web services, as well as annotating the web service method with

```
@Secured
(Role="testItemBankRoleGoesHere")org.springframework.security.access.annotation.Secured.
```

Assumptions:

- Authentication will be handled by a separate component
- The authentication method will be supported by Spring Security
- Authentication will be on a component client basis. Example: the Test Authoring component will be authenticated and granted authorization (by the security component) in the form of a statically defined role.
- The web service endpoints will be secured (via the @Secured annotation) and configured to be protected by a statically defined role, and improper access will result in a HTTP 401 (unauthenticated) or a HTTP 403 (unauthorized)
- Additional business (non-role) based security rules have not been specified, and will require additional effort if required.

Monitoring and Alerting

Due diligence was done to capture the functionality described in the requirement (validation, auditing, logging, exception handling). Most defined alerts are currently being written to a separate log file, which represents what will be written to Monitoring and Alerting as Alerts after the component is delivered. Where reasonable, a stubbed out implementation, writing to a separate log file, was implemented for future refactoring.

Validations and Alert: RADTIB.1.2, RADTIB.1.6, RADTIB.1.7, RADTIB.1.8, RADTIB.1.9, RADTIB.2.1, RADTIB.3.3--Any of these conditions will be handled as validations. They are coded and running as part of the import process. Any errors are captured in the persisted importSet, and will be written to Monitoring and Alerting as an alert after the component is delivered.

Audit Alerts: RADTIB.1.4--Item creation audits are captured in the persisted importSet. These alerts are currently being written to a separate log file, which represents what will eventually be written to Monitoring and Alerting as Alerts after the component is delivered. Note that for non-SFTP import/exports, the processing will complete and results will be returned synchronously. For SFTP imports and multiple item export via SFTP, the processing will happen asynchronously, and a pollable URL (to the importSet or ExportSet respectively), will be returned synchronously. For these async processes, it is the intent that the client can check status in one of two manners:

1.) By polling the URL for a completed status
2.) Via the Monitoring and Alerting alert that will be created upon completion.

Errors (unexpected): RADTIB.1.5, PRTIB.1, PRTIB.1.1--Any unexpected exceptions are currently being gracefully handled and logged with a reference number that is returned to the web service client for support escalation. These exceptions will be written to Monitoring and Alerting as errors after the component is delivered.

Audit metrics: RADTIB.4.3, RADTIB.4.4, PRTIB.1—Currently Test item bank is auditing all web service requests, along with the performance metrics of each request Successful and unscuccessful searches will be logged as metrics to the Monitoring and Alerting component after the component is delivered.

Configuration Component/Service Locator

This is a Smarter Balanced RFP #11 level issue. At the time of authoring the Test Item Bank component, the direction was unclear as to the location and implementation of this service. It was the intent of the development team to design a functional solution for now, that can be refactored in the future. Currently there are two instances where this configuration component will be needed. Both of these instances have been mitigated for now, but will be required to be completed when the direction is selected.

SFTP: Currently there is the ability to have one SFTP site configured that will be used by the async Import & Export endpoints. Eventually, the design is intended to allow for the Import/Export to specify an SFTP site key that will match to a SFTP configuration that can be located using the Configuration Component. For now the 4 keys required to do an SFTP import or export can be configured in one of 3 ways further described in the dependency Configuration below.

Component lookup: When Test Item Bank eventually integrates with Monitoring and Alerting there will be a need to discover the component's URL via a service locator component.

Dependency Configuration

Test Item Bank has two external dependencies configurations (MongoDB, and SFTP) that allow for three flexible configuration options.

1. **Embedded Property File** Include the configurations on the class path (as resources) in the deployed application. This is least secure, but has the least moving parts. Currently the files are only included as test resources for execuritng junit tests. The file names are mongo.properties and sftp.properties respectively.

2. **Environment Variable Key/Values** Set the required keys (those that would be contained in the property files) as environment variables. A little more secure, but still not ideal.

3. **External Property file** Create a secured property file on the server's file system. Pass the location of this file to the application as environment variables. The names of these environment variables are, SB11_TIB_MONGO_PROPS, and SB11_TIB_SFTP_PROPS respectively. This is the most secure, allowing for restricted access to only the process ID the

application is running under. It does require a little configuration/coordination at deploy time, but avoids exposing credentials as part of the artifact/runtime environment configuration.

APIP format

At the time of authoring the Test Item Bank component, the development team was provided several examples of APIP items. These examples were used as a basis for test items to be imported, but required some amount of scrubbing (removal of actual item content), as well as some erroneous/superfluous data. Details of this effort can be provided upon request. The samples used for development will be provided as part of the test resources included in the source tree, as well as in the test plan/reports.

Test Item Bank was defined to be Item content agnostic, however, there are several places where a specific binding was required. The search content is based off of the metadata file provided, and the bindings for these fields follow. Additionally, the metadata was listed as a dependency to each item. Seeing as the metadata file is an APIP extension, and no resourceType is defined, the convention of resource name starts with "meta" was used to locate the metadata file. Please note this is considered a backlog item in the eventuality that these metadata bindings may change based on source of items and their interpretation/extension of the APIP/QTI standard.

- itemType = metadata file: SYSTEM_ItemType
- version = metadata file: SYSTEM_ItemVersion
- keywords = metadata file: SYSTEM_ItemKeywords
- grade = metadata file: SYSTEM_Grade
- subject = metadata file: SYSTEM_Subject
- subjectCategory = metadata file: SYSTEM_ItemSubcategory
- status = metadata file: SYSTEM_Status
- author = metadata file: SYSTEM_Author
- itemDifficulty = metadata file: SB_Difficulty
- source = metadata file: SYSTEM_Source

XML vs. JSON format

At this time the services were developed to accept and return JSON as the primary data interchange format. Most endpoints also serve XML, but JSON should be used when binding to the services. Additionally, the format is able to be modified when feedback is received from integrations with other components.

SFTP Site cleanup

At this time, Test Item Bank does not delete or remove any files from the SFTP import site. It is the responsibility of the owner of the SFTP site to clean up or maintain the files on SFTP sites. Test Item Bank does archive the imported file in whole cloth, to allow for auditing and/or re-processing as may be required.

Temporary file storage

At this time Test Item Bank relies upon the temp storage directory (FileUtil lookup) of the application server to temporarily store the files during processing. The files are deleted once processing is completed, but it should be noted that the user the application is running under requires write access to the temp directory of the underlying file system.