Software Technical Design Specification

for

SBAC-11 Test Authoring

Version 1.0

Revision History

Date	Version	Description	Author
04/14/2014	1.0	Initial	Paul Krumrei
04/14/2014	1.1	Final	Paul Krumrei/Mike Stern

Table of Contents

Software Technical Design – Test Authoring

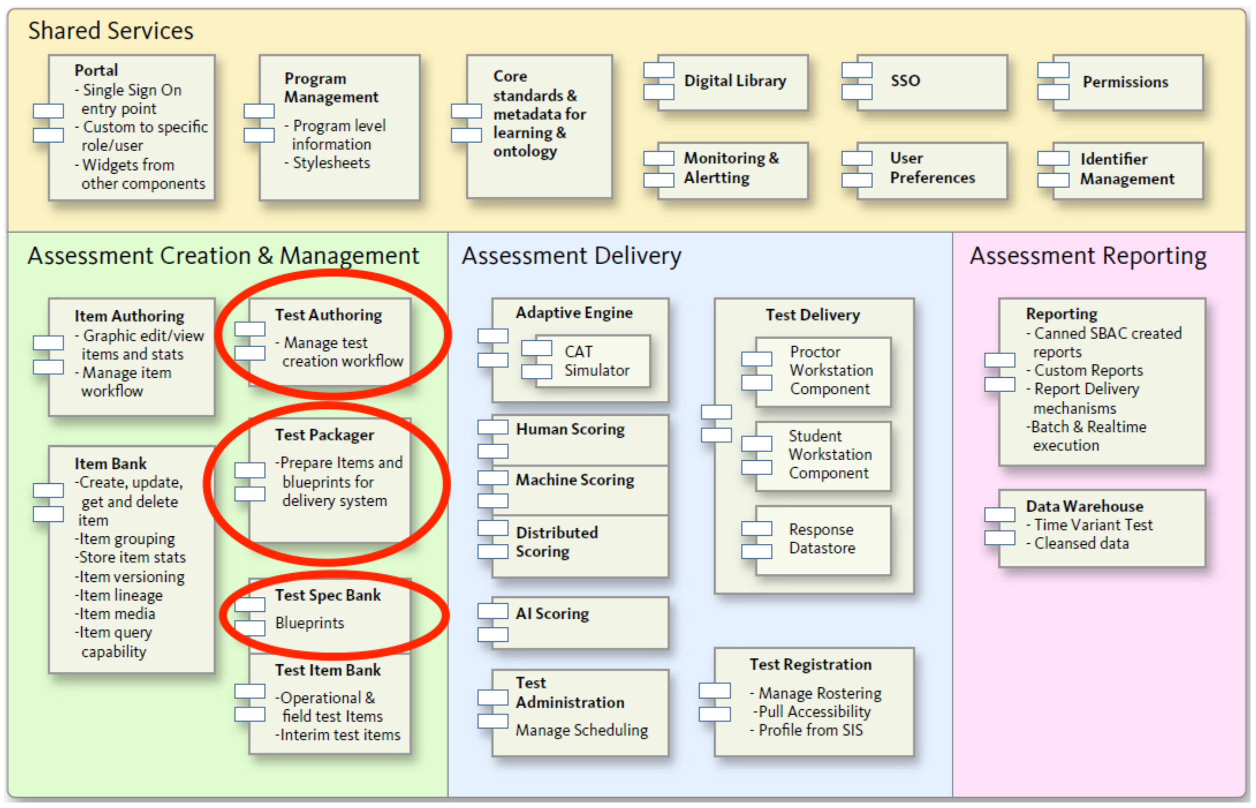
1. Introduction

1.1 Purpose

The objective of this document is to create SBAC-11 Test Authoring software technical design specifications. All architectural information related to the software within the scope of this document will be included and is intended to be a living document updated as needed throughout the life-cycle of the project. On project close the design document will be included in the project close documentation and ownership of the document will fall to the Production support of the application(s) described within. Production support teams are then responsible for updating documentation upon completion of architecturally significant changes.

1.2 Smarter Balanced Assessment Consortium logical components overview

The diagram below depicts the components of the assessment system. The focus of this document is 'Test Authoring' which is highlighted in red.



2. Design Goals, Constraints and Assumptions

2.1 Test Authoring

The Test Authoring component is responsible for the creation, management and workflow of test specification data. It uses the Test Spec Bank component to store and retrieve test specification data and queries the Test Item Bank for item data based on item metadata for assignment to adaptive item pools and test forms.

Test Packager functionality is downstream of Test Authoring, and uses test specifications that have advanced to a given level in the workflow to be combined with item data from the Test Item Bank to create test packages for various purposes including registration, simulation, scoring, reporting and administration.

The Test Authoring and Test Packager components reference the same components and must be aware of the same test specification workflow. If they were separate components, there would be a large amount of shared interface and workflow code. For this reason, the two components can be merged and Test Packaging can be considered to a function of Test Authoring.

The Core Standards component provides standards publications as a source of standards information by Test Authoring for the creation of blueprints and other uses.

2.2 Test Spec Bank

The Architecture Report describes the Test Spec Bank component as follows:

> *This component is a repository for test specifications, blueprints and other data about tests such as the adaptive algorithm to be used during the test.*

Regarding the Test Authoring component, the Architecture Report describes the following:

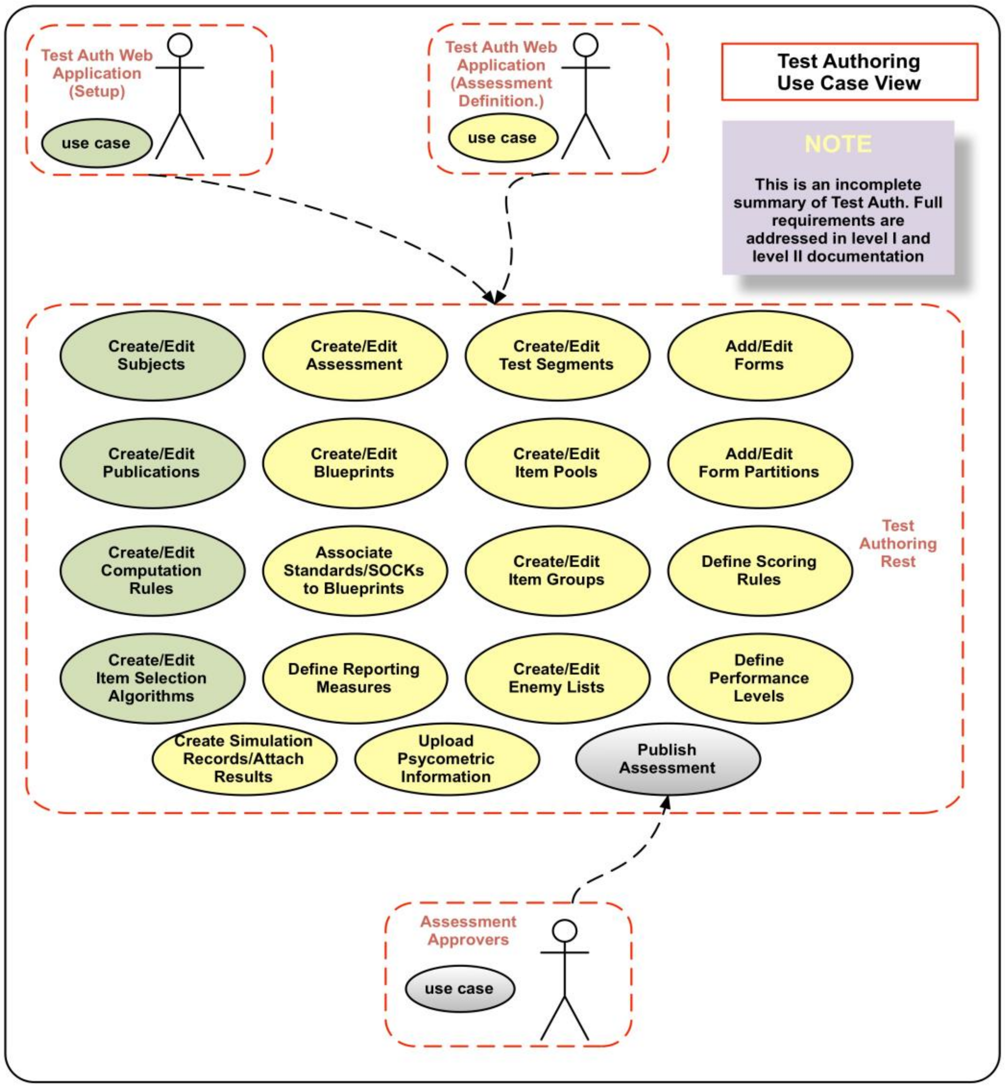
> *This component is a graphical interface used for creating test blueprints and specifications and manage the workflow. It will interact with the Test Item Bank component and the Test Spec Bank component.*

2.3 Test Packager

Test Packager is the combination of a test specification in XML format that conforms to a test specification DTD, and optionally (depending on the purpose of the test package) item metadata, or item content, metadata and item assets. Not all types of test package require item content, metadata or assets, only those intended for simulation and administration.

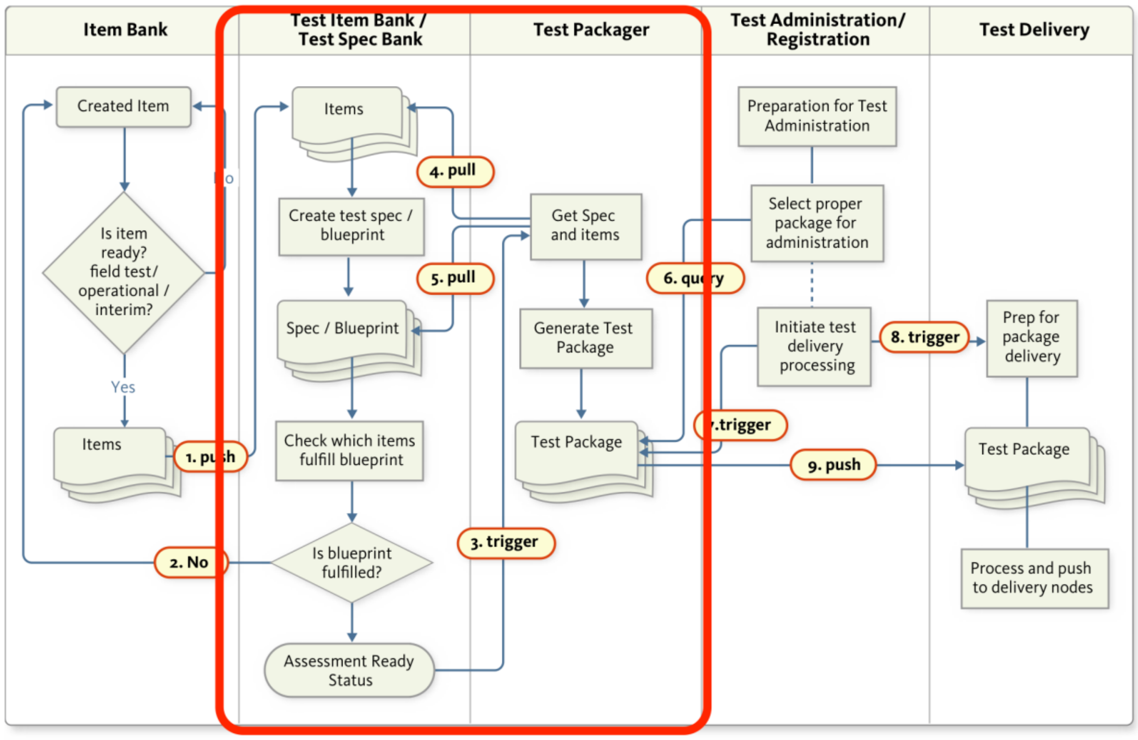
3. Use-Case View Figure

The following use cases are based on level-II requirements only. It is subject to change based on the review comments.



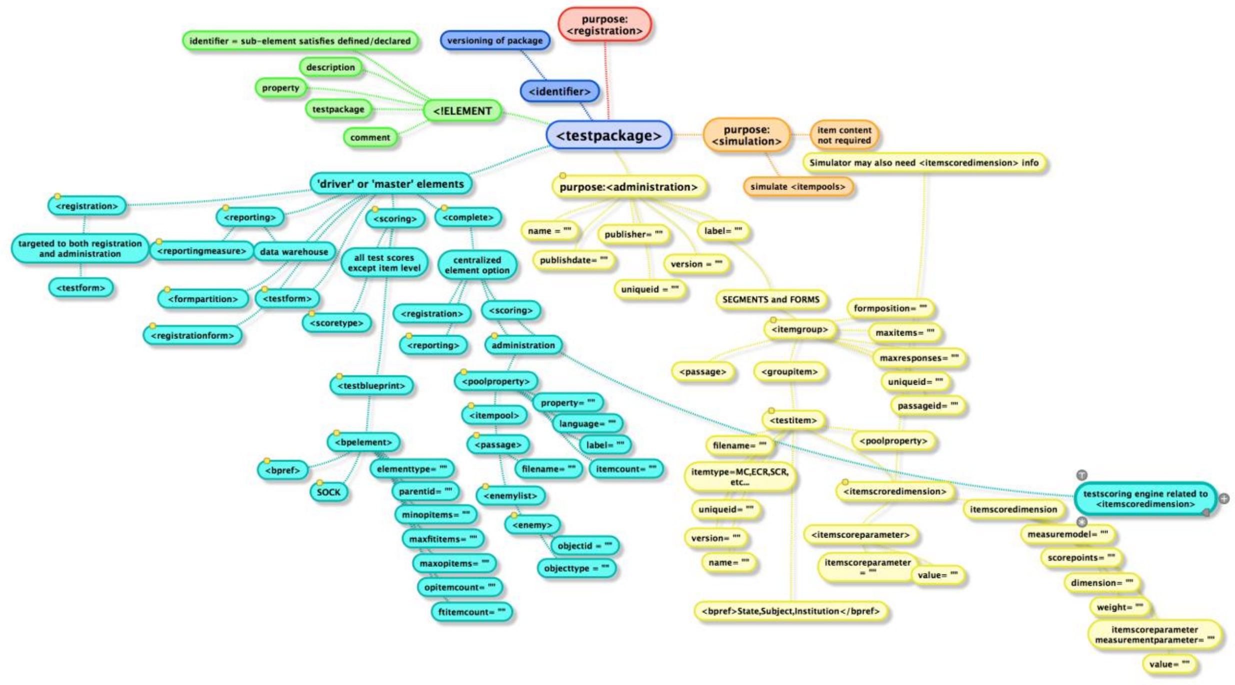
Test Auth Web Application (Setup)
use case
Test Auth Web Application (Assessment Definition.)
use case
Test Authoring Use Case View
NOTE
This is an incomplete summary of Test Auth. Full requirements are addressed in level I and level II documentation
Create/Edit Subjects
Create/Edit Assessment
Create/Edit Test Segments
Add/Edit Forms
Create/Edit Publications
Create/Edit Blueprints
Create/Edit Item Pools
Add/Edit Form Partitions
Create/Edit Computation Rules
Associate Standards/SOCKs to Blueprints
Create/Edit Item Groups
Define Scoring Rules
Test Authoring Rest
Create/Edit Item Selection Algorithms
Define Reporting Measures
Create/Edit Enemy Lists
Define Performance Levels
Create Simulation Records/Attach Results
Upload Psycometric Information
Publish Assessment
Assessment Approvers
use case

4. Process View

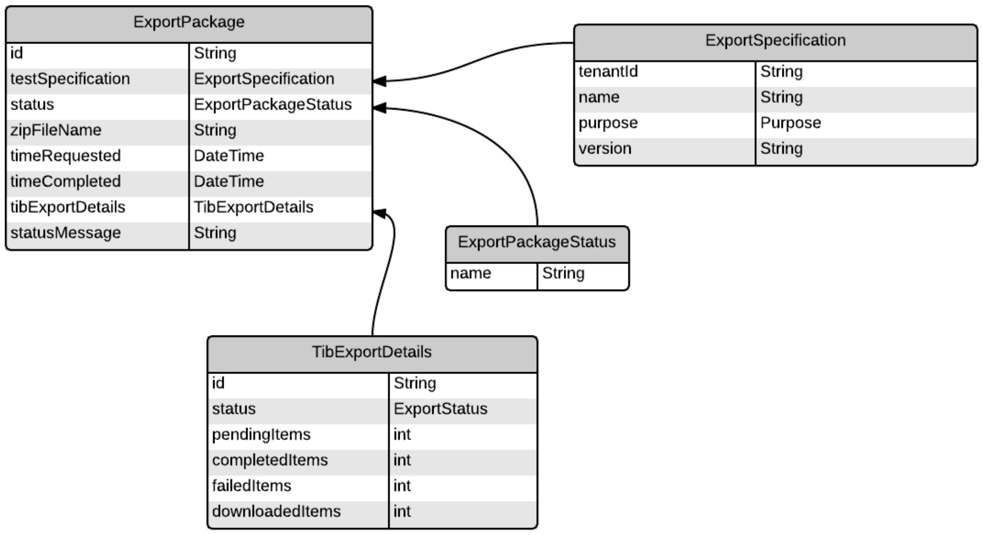


Item Bank
Test Item Bank / Test Spec Bank
Test Packager
Test Administration/ Registration
Test Delivery
Created Item
Is item ready? field test/ operational / interim?
No
Yes
Items
1. push
2. No
Items
4. pull
Create test spec / blueprint
5. pull
Spec / Blueprint
Check which items fulfill blueprint
Is blueprint fulfilled?
Assessment Ready Status
3. trigger
Get Spec and items
Generate Test Package
Test Package
6. query
7. trigger
9. push
Preparation for Test Administration
Select proper package for administration
Initiate test delivery processing
8. trigger
Prep for package delivery
Test Package
Process and push to delivery nodes

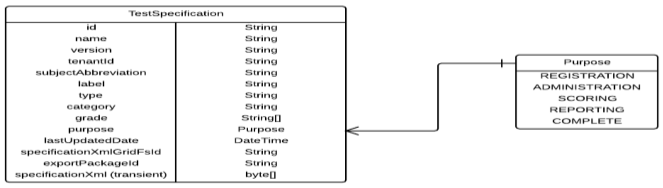
5. Data View



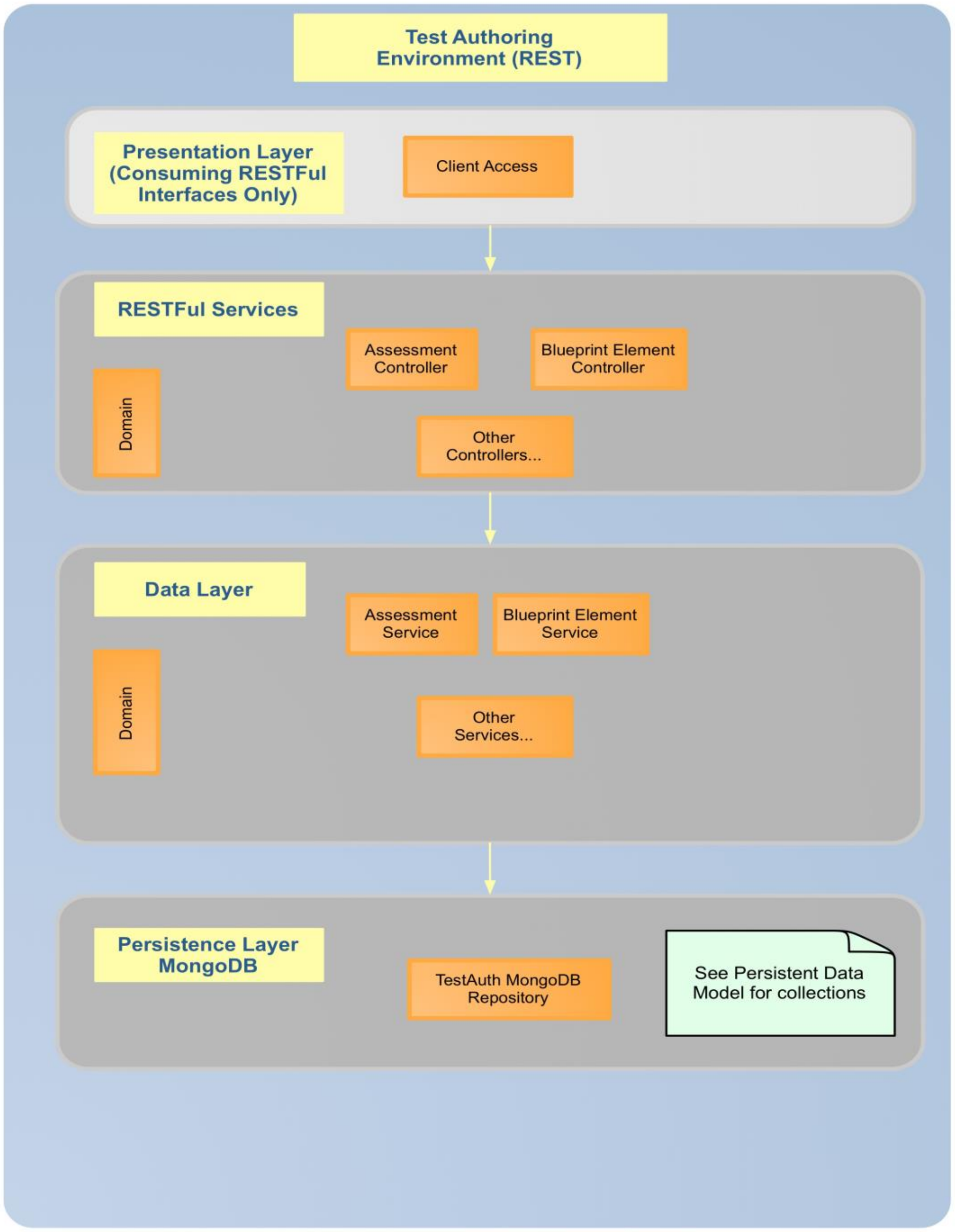
Test Packager Diagram



Test Spec Bank Diagram

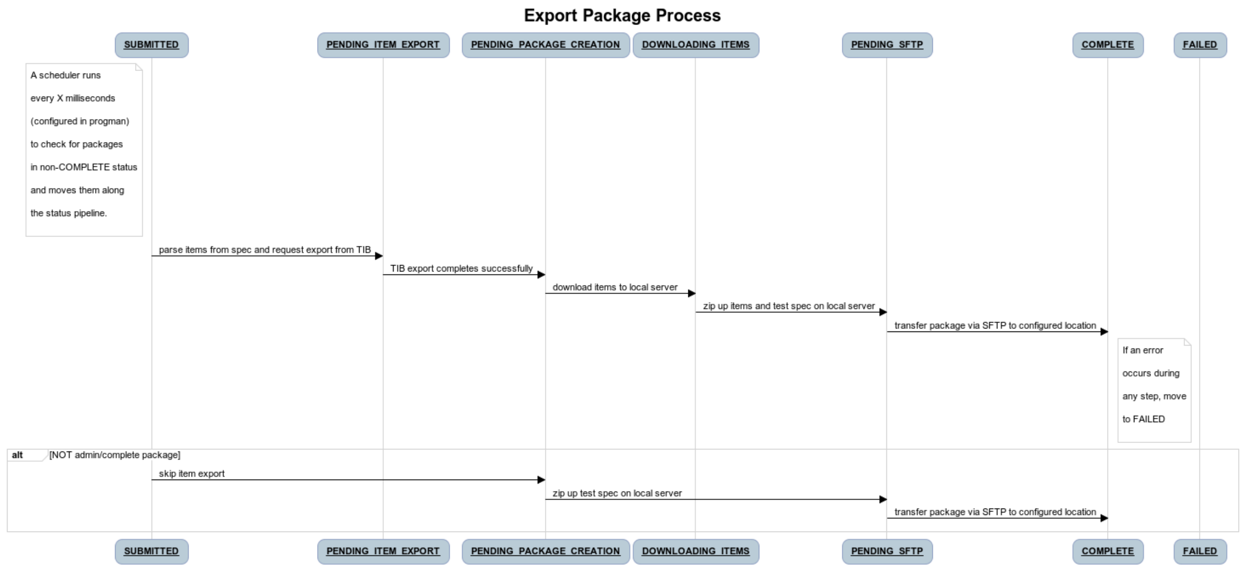


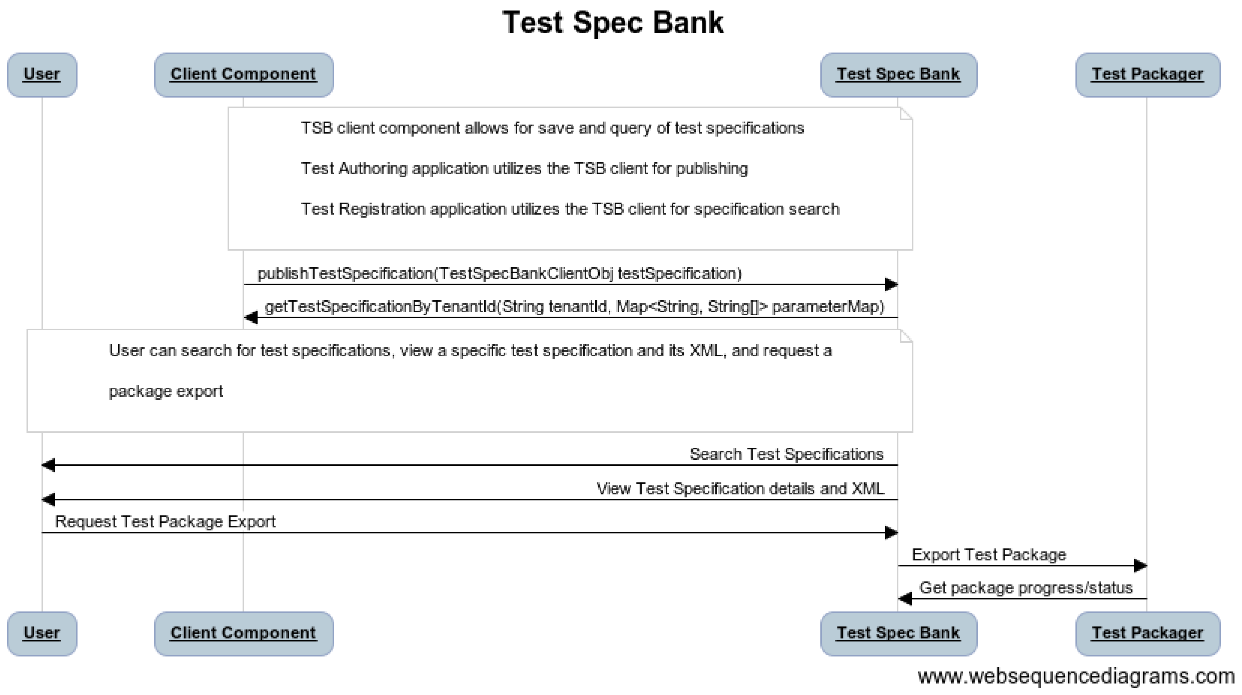
6. Component Layered View


Test Authoring Environment (REST)
Presentation Layer (Consuming RESTFul Interfaces Only)
Client Access
RESTFul Services
Domain
Assessment Controller
Blueprint Element Controller
Other Controllers...
Data Layer
Domain
Assessment Service
Blueprint Element Service
Other Services...
Persistence Layer MongoDB
TestAuth MongoDB Repository
See Persistent Data Model for collections

7. Dynamic View

The sequence diagrams have been created to be used as a guided tour through the code. The diagrams do not incorporate every endpoint or process flow, but rather highlight representative patterns within the application's design. The sequence diagrams are best consumed with the code base.


Export Package Process
SUBMITTED
PENDING_ITEM_EXPORT
PENDING_PACKAGE_CREATION
DOWNLOADING_ITEMS
PENDING_SFTP
COMPLETE
FAILED
A scheduler runs every X milliseconds (configured in progman) to check for packages in non-COMPLETE status and moves them along the status pipeline.
parse items from spec and request export from TIB
TIB export completes successfully
download items to local server
zip up items and test spec on local server
transfer package via SFTP to configured location
If an error occurs during any step, move to FAILED
alt
[NOT admin/complete package]
skip item export
zip up test spec on local server
transfer package via SFTP to configured location


Test Spec Bank
User
Client Component
Test Spec Bank
Test Packager
TSB client component allows for save and query of test specifications
Test Authoring application utilizes the TSB client for publishing
Test Registration application utilizes the TSB client for specification search
publishTestSpecification(TestSpecBankClientObj testSpecification)
getTestSpecificationByTenantId(String tenantId, Map<String, String[]> parameterMap)
User can search for test specifications, view a specific test specification and its XML, and request a package export
Search Test Specifications
View Test Specification details and XML
Request Test Package Export
Export Test Package
Get package progress/status
www.websequencediagrams.com

8. Implementation View

This view describes the organization of static software modules (source code, data files, executables, documentation etc.) in SBAC-11 Test Authoring component development in terms For Test Authoring, Test Spec Bank and Test Authoring.
The Code review documentation and installation instructions are located in Knowledge Tree and the external release doc directory of each Projects source repository.

8.1 Source Code

The source code for Test Authoring is implemented independent of other components. The component source repository is divided into sub-projects as follow:
The REST module is a deployable WAR file (test-auth.rest-VERSION.war) that provides REST endpoints that can be used to access and modify Test Authoring data. The REST module has an internal dependency to the SB11 Test Authoring Persistence module. ; following the same as the webapp module.
Webapp Module: The Webapp module is a deployable WAR file (test-auth.webapp-VERSION.war) that provides the administrative UI for Test Authoring functionality. The Webapp module uses the REST module for all data access, but this is a runtime dependency through a REST endpoint and not a direct code dependency.

Build Process:

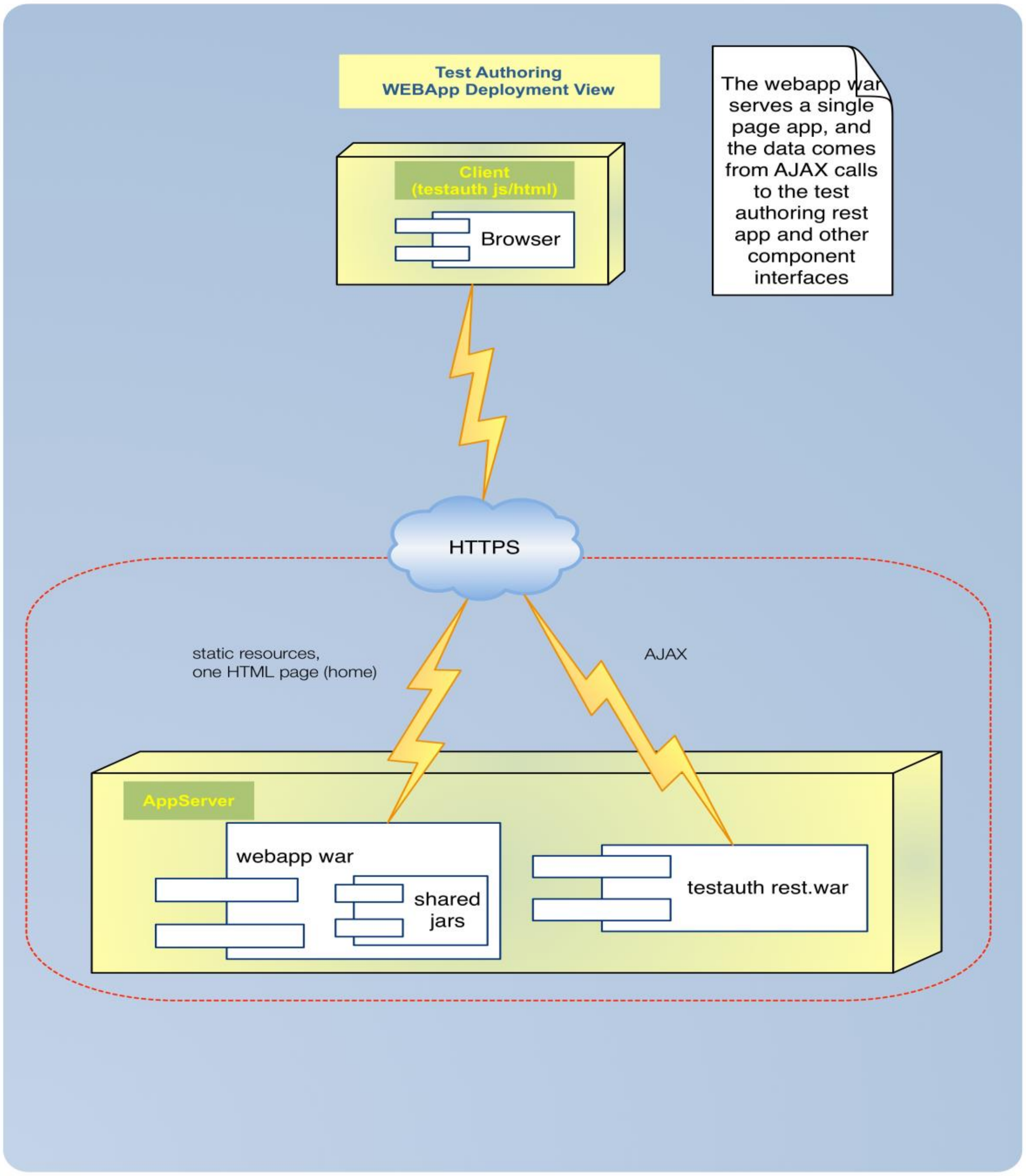
Test Authoring is using maven to automate the compile/test/package process. A parent pom.xml at the top level defines common dependencies for each component. The parent pom.xml files declaratively add dependencies unique to each sub- project. Additionally, common dependencies are resolved by maven using pom inheritance as well as transitive dependency resolution from child dependencies.

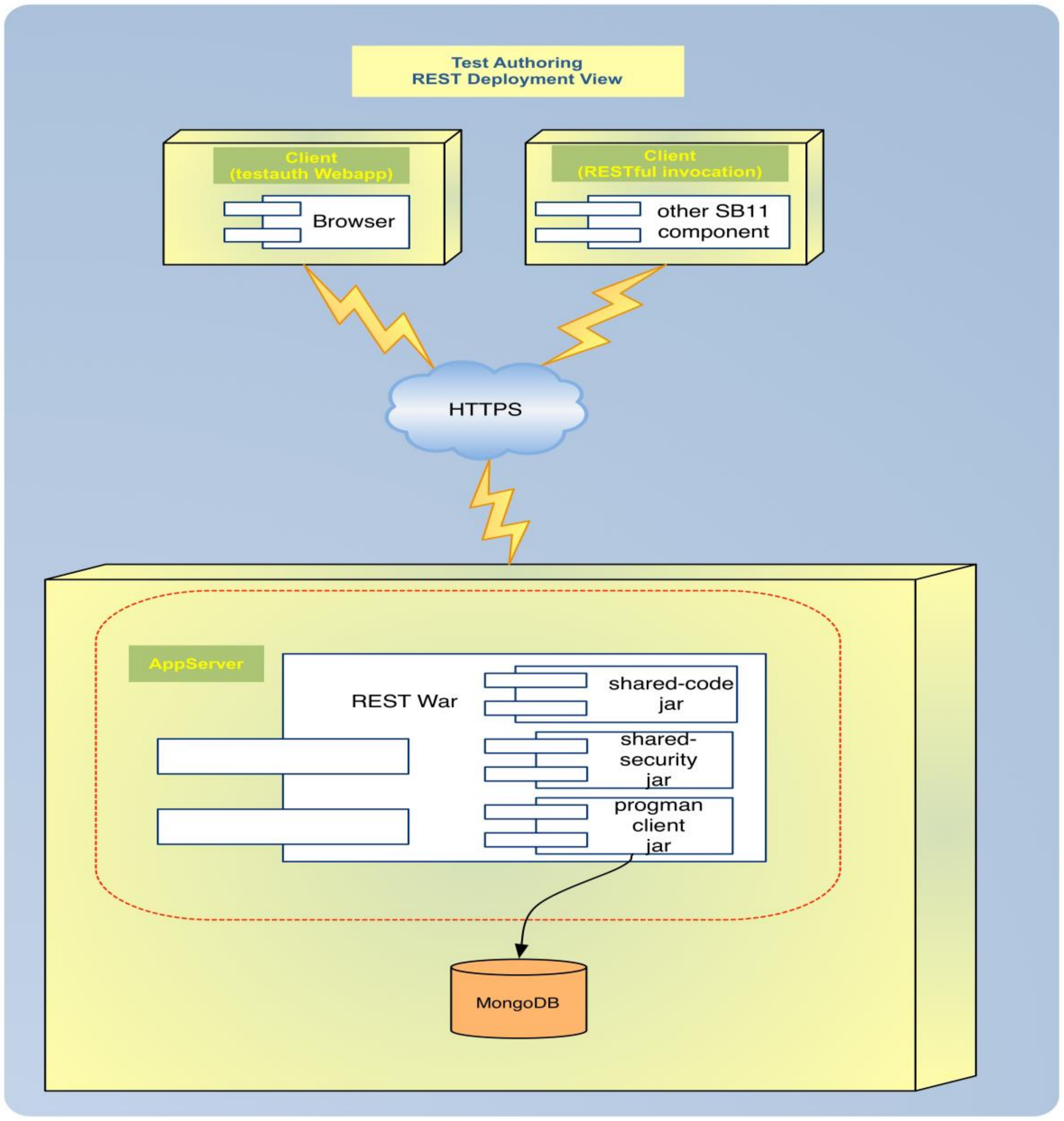
8.2 Third party dependencies

These dependencies are fully described in the respective installation instructions in the external release docs included in the repository as well as Knowledge Tree.

- Test Authoring: installation/TestAuth_Install.pdf
- Test Spec Bank: installation/testpackager_Install.pdf
- Test Packager: installation/testspecbank_Install.pdf

9. Deployment View


Test Authoring
WEBApp Deployment View
The webapp war serves a single page app, and the data comes from AJAX calls to the test authoring rest app and other component interfaces
Client (testauth js/html)
Browser
HTTPS
static resources, one HTML page (home)
AJAX
AppServer
webapp war
shared jars
testauth rest.war



10. Security

HTTPS will be provided in the hosted environment. This is not a concern of the application layer as all negotiation and encryption is handled between the network and application server container.

Authentication and Authorization is an orthogonal concern within the application. At a servlet container level, Spring Security allows for global and/or more specific security masking of web resources. For example, all web service endpoints can require clients to be authenticated while leaving other assets (such as images or static content) unsecured. At this time, a spring security web filter has been defined for both the web application as well as the rest application container. Selective exceptions have been added where necessary to allow access for unauthenticated requests.

All web service endpoints have been secured using Spring Security Annotations. These annotations allow for permission based security. Authentication is provided by the SB11 OpenAM installation via SAML. Component permissions, as well as user roles are defined independently in the SB11 permissions application. Additionally, the SB11 Permissions component administers Role to Permission mapping.

Authentication
Users are provisioned into Open AM via the SB11 Test Registration Component. User/role/entity mapping is provided by the SB11 Test registration component at the time of provisioning. As part of the provisioning process, the users credentials (password, security questions etc) are established. In order to access either the Web application or the rest interfaces, the user has to Authenticate to OpenAm using the established credentials.

Authorization
Within the functional components, such as Test Authoring and Test Spec Bank, the Spring Security annotations are configured to ensure the permissions defined by the shared permission component are in alignment. As part of the OpenAM authentication procedure, the user's Entity Role mappings are provided to the user context of the component. The component is then responsible to:

1. Ensure the entity associated with each role has valid tenancy for the given application

2. If valid tenancy is found to grant the associated permissions for the given role by matching them to the role to permission mapping from the SB11 permission component.

Tenanted Data
Additionally, the data (specifically the assessments, and test specifications) are logically partitioned for the Tenant the user is acting on behalf of. All appropriate data is then scoped to a given tenant. Additional Spring Security checks have been added to the appropriate rest endpoints to Pre or Post authorize the user has sufficient access to the tenanted data they are requesting or mutating.

If any of these conditions are not met, Spring security handles insufficient authorization by denying access with a HTTP 401 error returned.

11. Exception and error handling

11.1 Error handling

Known error conditions such as validation rules will be reported to the client in a consistent manner. The errors will be returned to the user with an HTTP 400: BAD REQUEST and the error message text included in the returned payload. Known errors are enumerated within a component and the dynamic portion of the message parameterized. This allows for static portions of the messages to have multiple translations (a development time concern as defined by the requirements). Spring Validation is being used as the base framework for error checking and validation logic is extended within the application as necessary. Some of the Test Authoring error handling examples are:

POST: the service will return HTTP status 201 (created) when a log entry is added

POST: the service will return HTTP status 201 (created) when a notificationRule is added, and an id will be returned which can be used to fetch the rule

GET with query params: return a list of results and HTTP 200 when valid query parameters are specified, and HTTP 400 Bad Request if the query parameters are not valid.

11.2 Exception handling

For all exceptions generated from unanticipated conditions, a fault barrier has been put in place to ensure that no details of the originating exception are exposed to the client of the web service. Instead of exposing the exception details (which may contain implementation details), a customizable error message including a unique reference to the logged exception is returned to the user. The unique reference allows the user to communicate with technical support in an unambiguous manner to quickly locate the original root cause for problem resolution. The logged exception will be logged to the local application server environment as well as to the Monitoring and Alerting component for centralized aggregation eventually.

12. Quality

12.1 Scalability

The design of Test Authoring RESTful services allows horizontal scaling. Test Authoring persistence layer uses MongoDB which has auto-sharding capability to scale from a single server deployment to large, complex multi-site architectures.

12.2 Reliability

Test Authoring logs messages on each local server as well as writing messages to a centralized data store for persistence. The local log can be used by system administrators if communication with the Test Authoring component is lost. The MongoDB data store has built-in replication with automated failover which provides enterprise-grade reliability and operational flexibility.

Each system deployment must provide at least two component servers to host the Test Authoring component to provide failover.

12.3 Availability

Each system deployment must provide at least two component servers to host the Test Authoring component to provide failover. Test Authoring leverages MongoDB's built-in replication with automated failover to provide high availability for data.

13. API

Once the Test Authoring component is installed and running, the API can be viewed by entering one of the below links A full API pdf is located in Knowledge Tree and the external release doc directory of each project's source repository.

Test Authoring API Context Roots

- api/blueprintElement
- api/item
- api/subject
- api/approval
- api/form
- api/progman
- api/publication
- api/user
- api/coreStandard
- api/itemSelectionAlgorithm

- api/blueprintSock
- api/segment
- api/itemGroup
- api/scoringRule
- api/assessment
- api/scoringFunction
- api/reportingMeasure
- api/enemy
- api/fileGroup
- api/performanceLevel
- api/tibitem
- api/publishingRecord
- api/formPartition

Test Spec Bank API

- api/exportPackage
- api/testSpecification
- api/user

Test Package API

- api/exportPackage