Software Technical Design Specification

for

SBAC-11 Test Item Bank

Version 1.1

Revision History

Date	Version	Description	Author
09-07-2012	1.0	Initial Draft	Manohar Sunkum Mike Stern
01-31-2013	1.1	Updated persistence and deployment models	Manohar Sunkum Mike Stern

Table of Contents

Software Technical Design – Test Item Bank

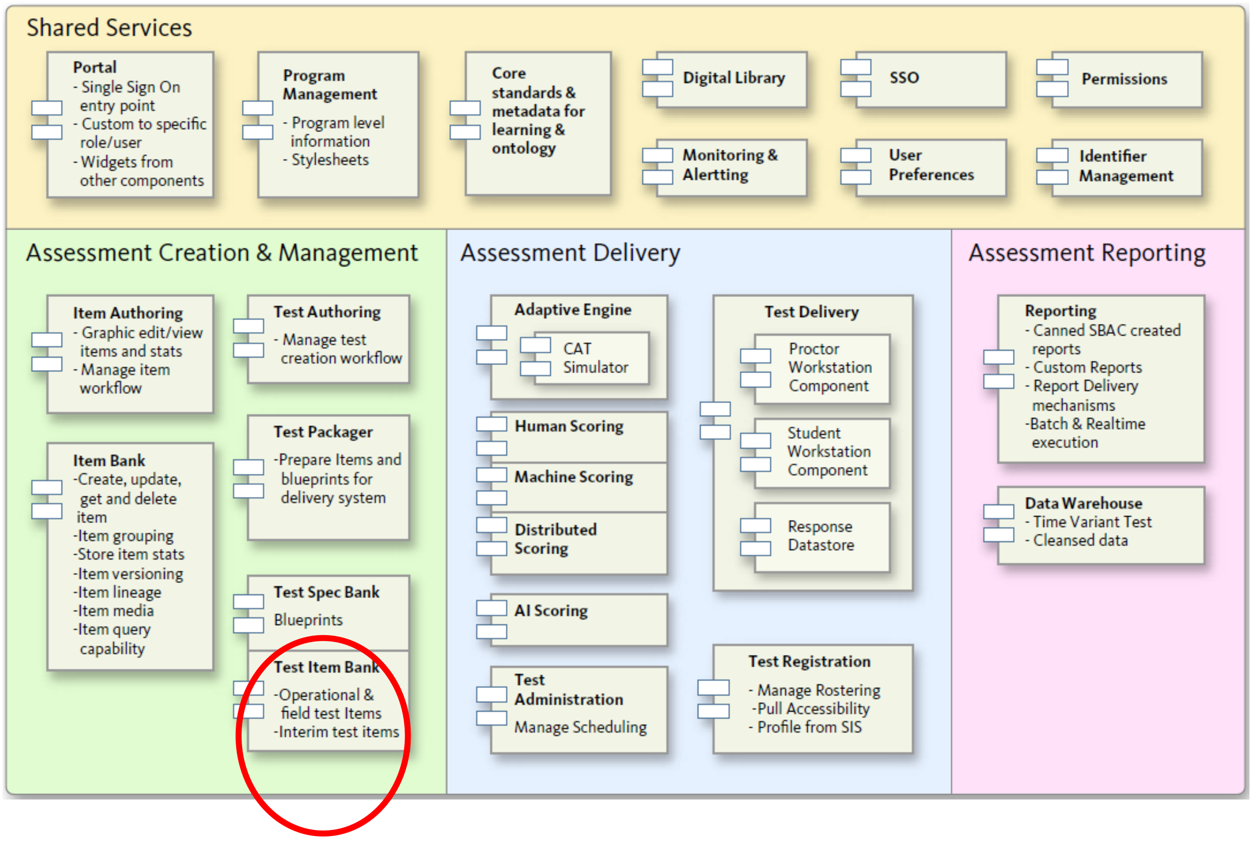
1. Introduction

1.1 Purpose

The objective of this document is to create SBAC-11 Test Item Bank 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 Test Item Bank 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 Item Bank' which is highlighted in red.


Shared Services
Portal - Single Sign On entry point - Custom to specific role/user - Widgets from other components
Program Management - Program level information - Stylesheets
Core standards & metadata for learning & ontology
Digital Library
SSO
Permissions
Monitoring & Alertting
User Preferences
Identifier Management
Assessment Creation & Management
Item Authoring - Graphic edit/view items and stats - Manage item workflow
Test Authoring - Manage test creation workflow
Test Packager -Prepare Items and blueprints for delivery system
Item Bank -Create, update, get and delete item -Item grouping -Store item stats -Item versioning -Item lineage -Item media -Item query capability
Test Spec Bank Blueprints
Test Item Bank -Operational & field test Items -Interim test items
Assessment Delivery
Adaptive Engine
CAT Simulator
Human Scoring
Machine Scoring
Distributed Scoring
AI Scoring
Test Administration Manage Scheduling
Test Delivery
Proctor Workstation Component
Student Workstation Component
Response Datastore
Test Registration - Manage Rostering -Pull Accessibility - Profile from SIS
Assessment Reporting
Reporting - Canned SBAC created reports - Custom Reports - Report Delivery mechanisms -Batch & Realtime execution
Data Warehouse - Time Variant Test - Cleansed data

2. Design Goals, Constraints and Assumptions

2.1 Design Goals

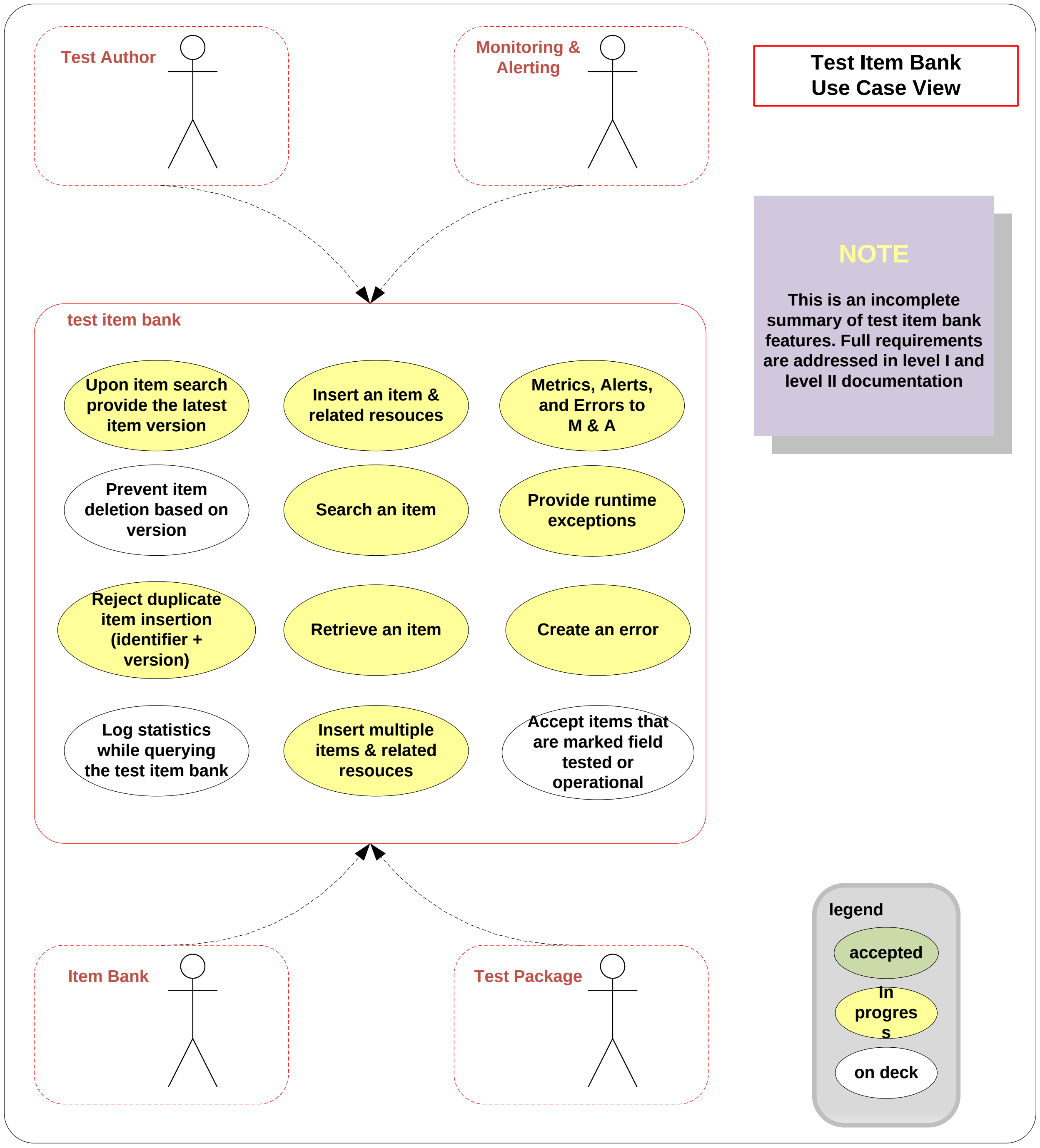
- Provide the capability for storing and retrieving assessment items.
- Provide the capability for storing and retrieving assets and metadata related to the assessment items.
- Provide the capability for item version tracking
- Providing a robust search and query capability that allows searching on 12 metadata fields. For more details please refer to level-II requirements.

2.2 Design Constraints

- TBD

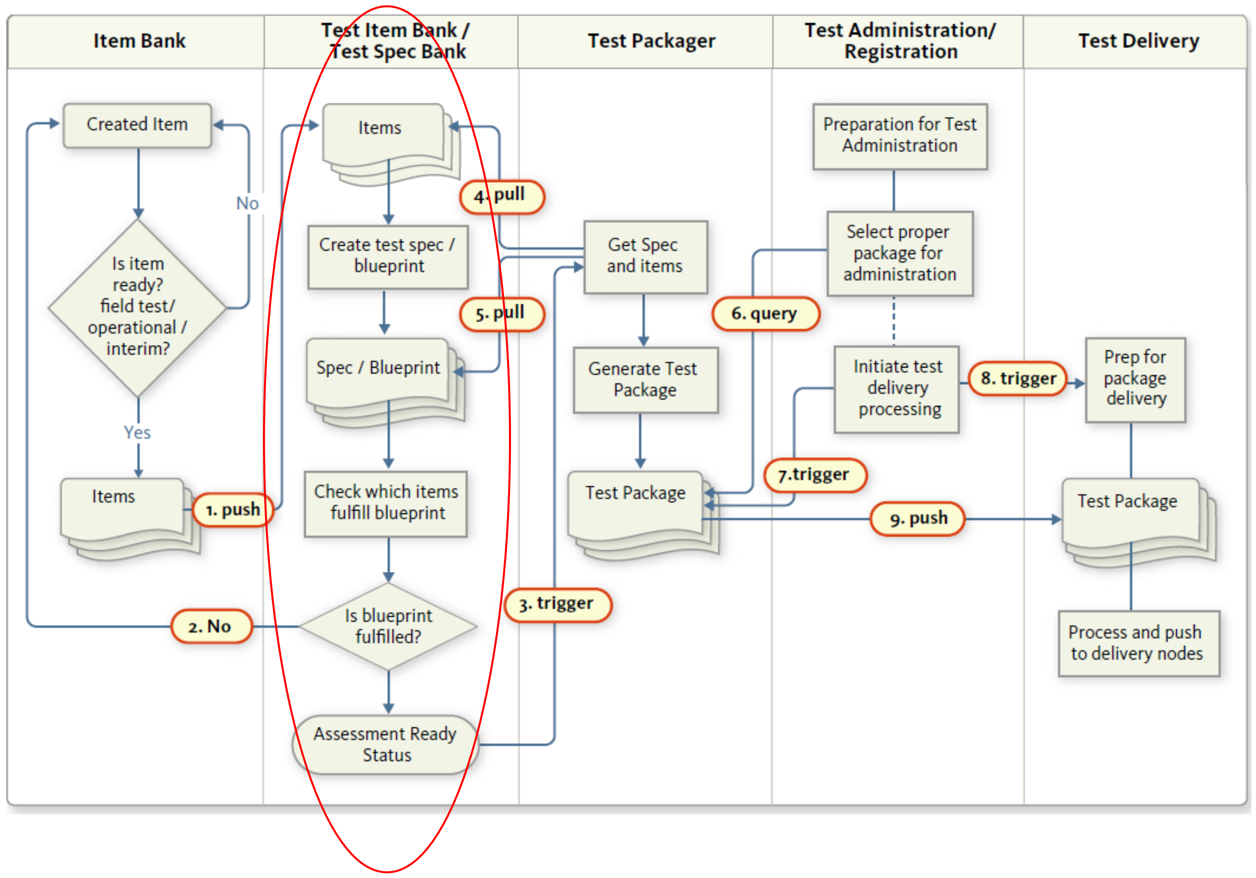
3. Use-Case View Figure

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


Test Author
Monitoring & Alerting
Test Item Bank Use Case View
NOTE
This is an incomplete summary of test item bank features. Full requirements are addressed in level I and level II documentation
test item bank
Upon item search provide the latest item version
Insert an item & related resouces
Metrics, Alerts, and Errors to M & A
Prevent item deletion based on version
Search an item
Provide runtime exceptions
Reject duplicate item insertion (identifier + version)
Retrieve an item
Create an error
Log statistics while querying the test item bank
Insert multiple items & related resouces
Accept items that are marked field tested or operational
Item Bank
Test Package
legend
accepted
In progres s
on deck

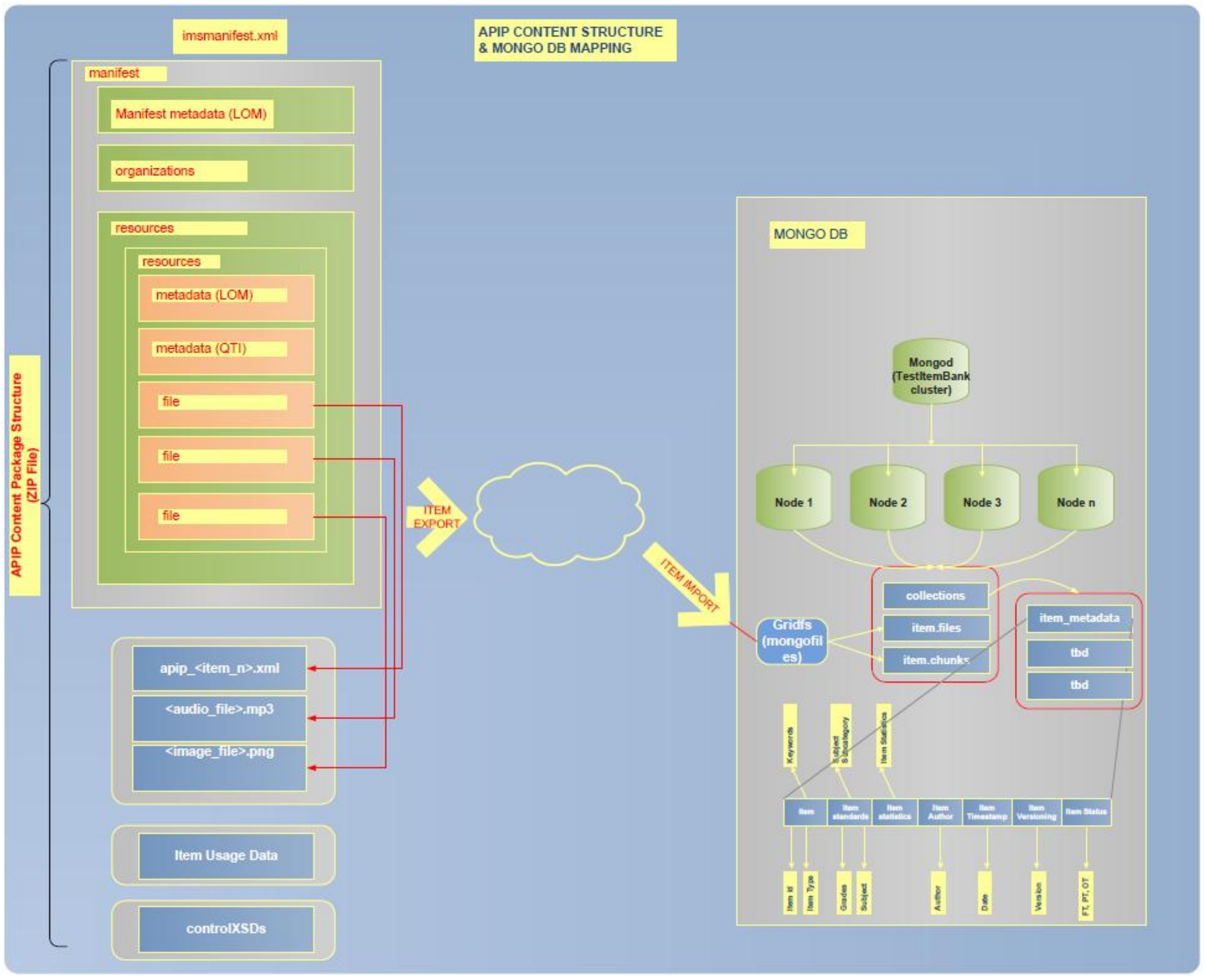
4. Process View

4.1 Test item bank process inter dependencies sequence flow (Includes business components only)


Item Bank
Test Item Bank / Test Spec Bank
Test Packager
Test Administration/ Registration
Test Delivery
Created Item
No
Is item ready? field test/ operational / interim?
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
Preparation for Test Administration
Select proper package for administration
Initiate test delivery processing
8. trigger
9. push
Prep for package delivery
Test Package
Process and push to delivery nodes

5. Data View

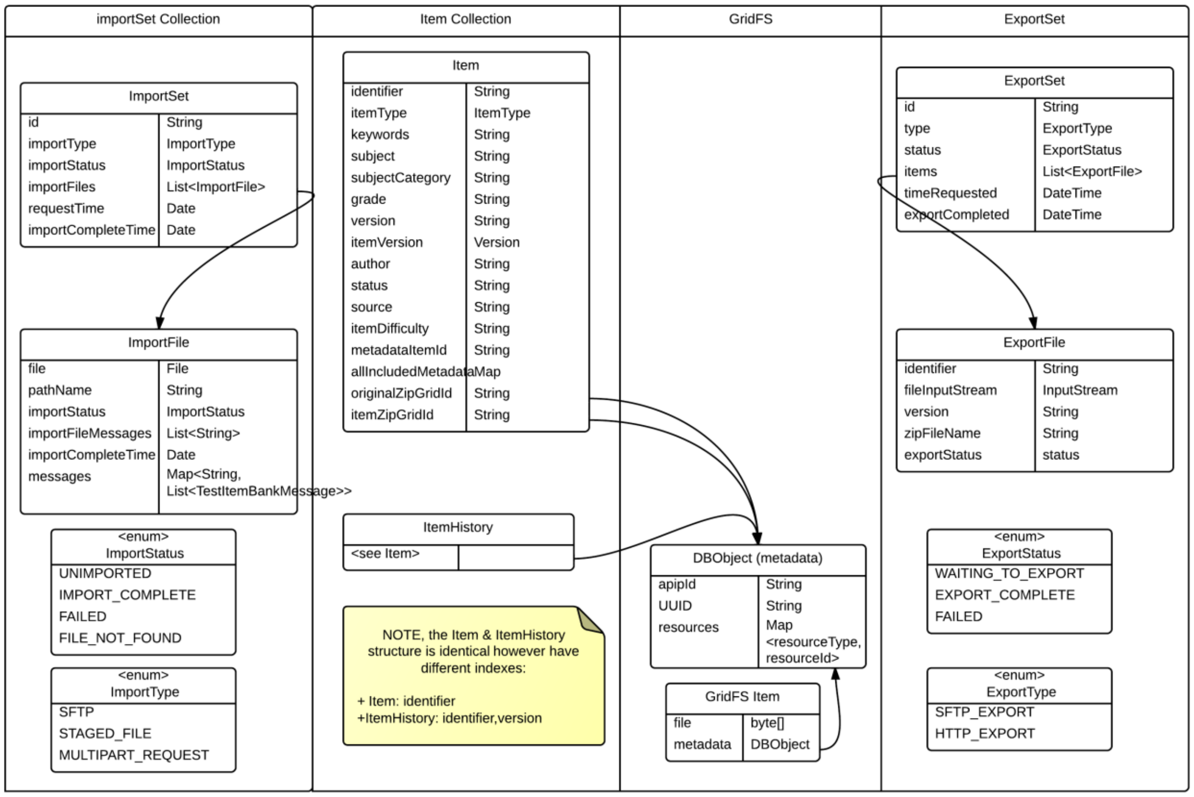
5.1 Conceptual data mapping (APIP → Mongo)



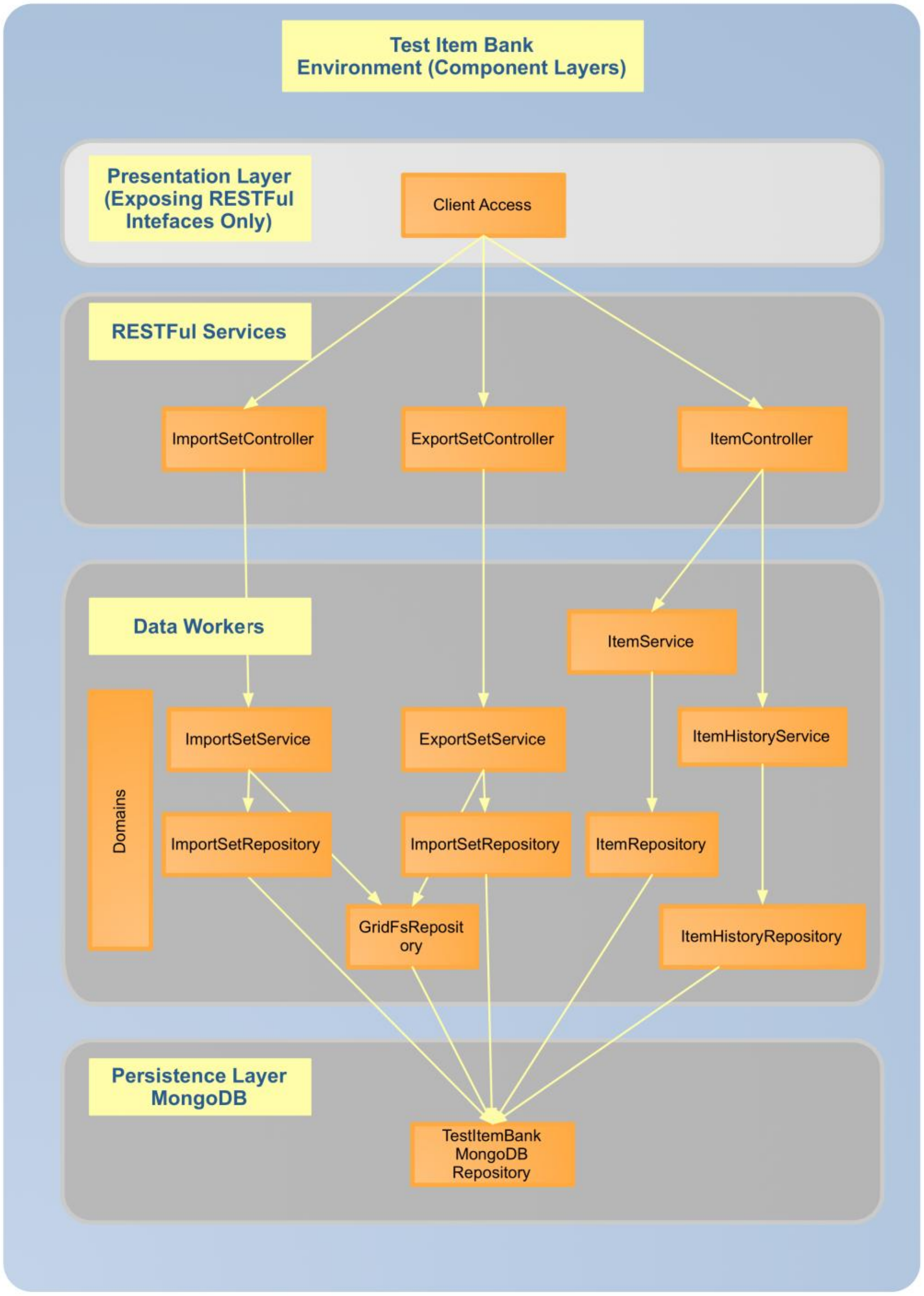
5.2 Physical Data Model

Test Item Bank NoSQL physical schema consists of the following collections and nested documents:

- ImportSet → {ImportSet → ImportFile }
- ImportStatus
- ImportType
- ItemCollection → {Item → Version → Stats & DBOject}
- GridFS →{ GridFSItem → DBOject }

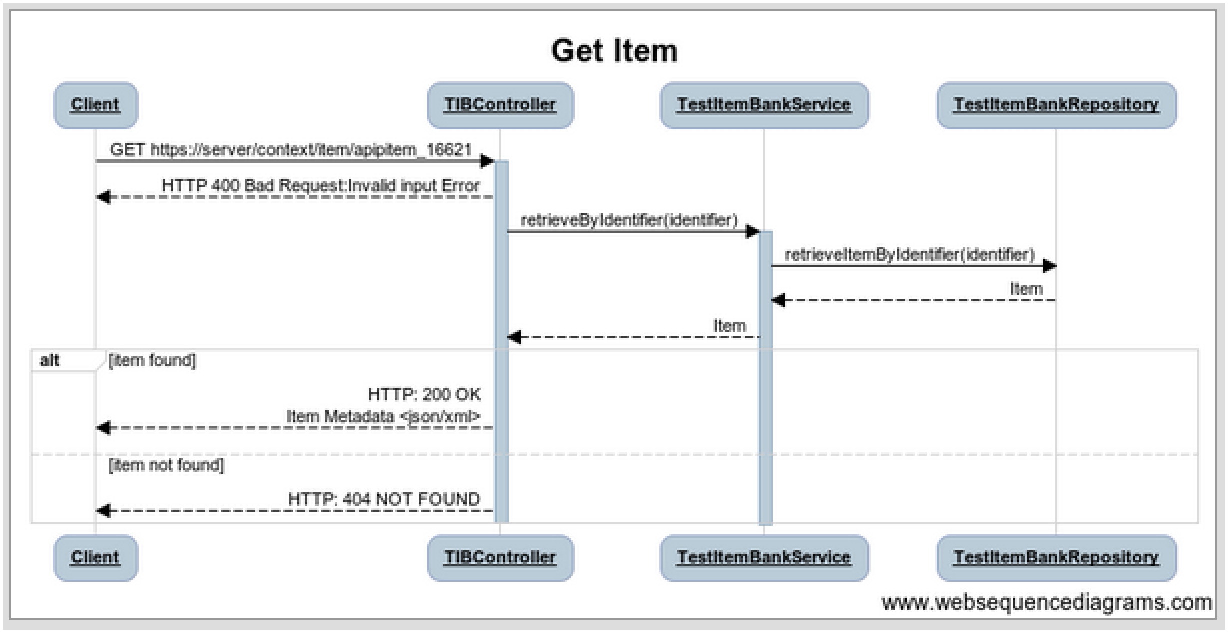


6. Component Layered View

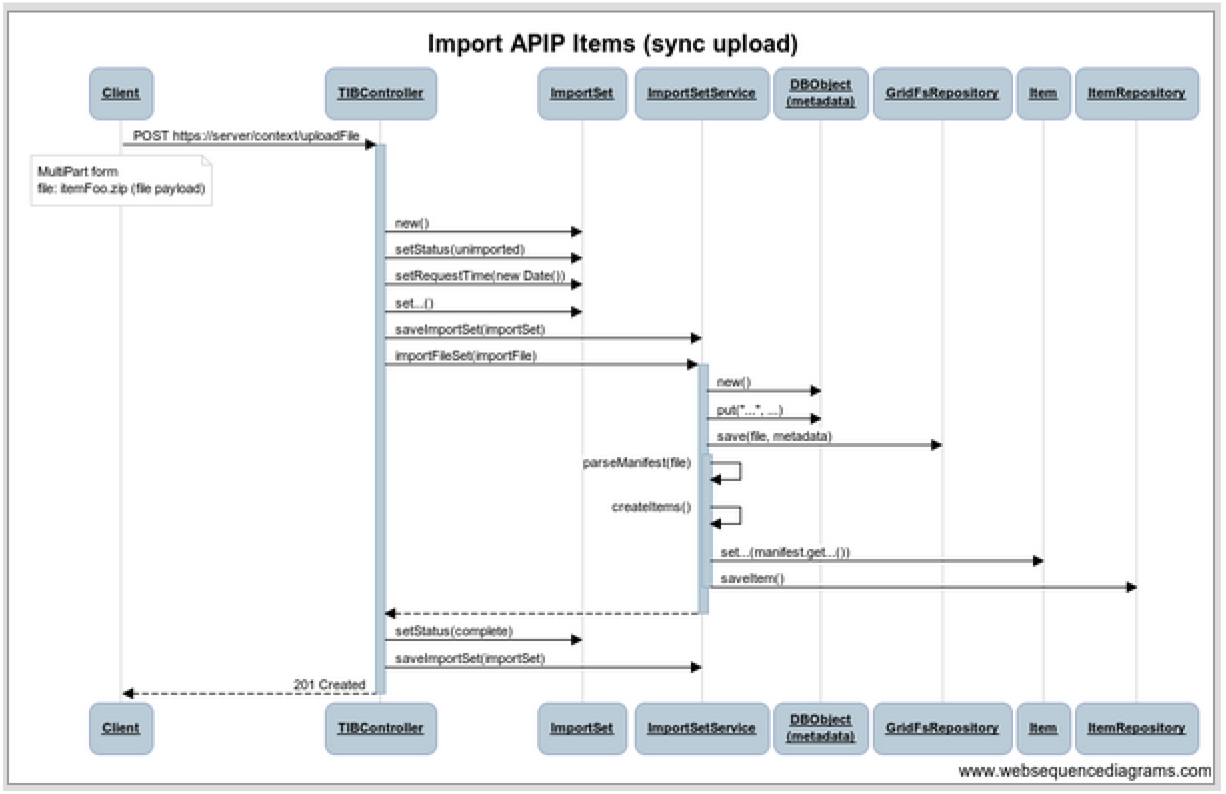


7. Dynamic View

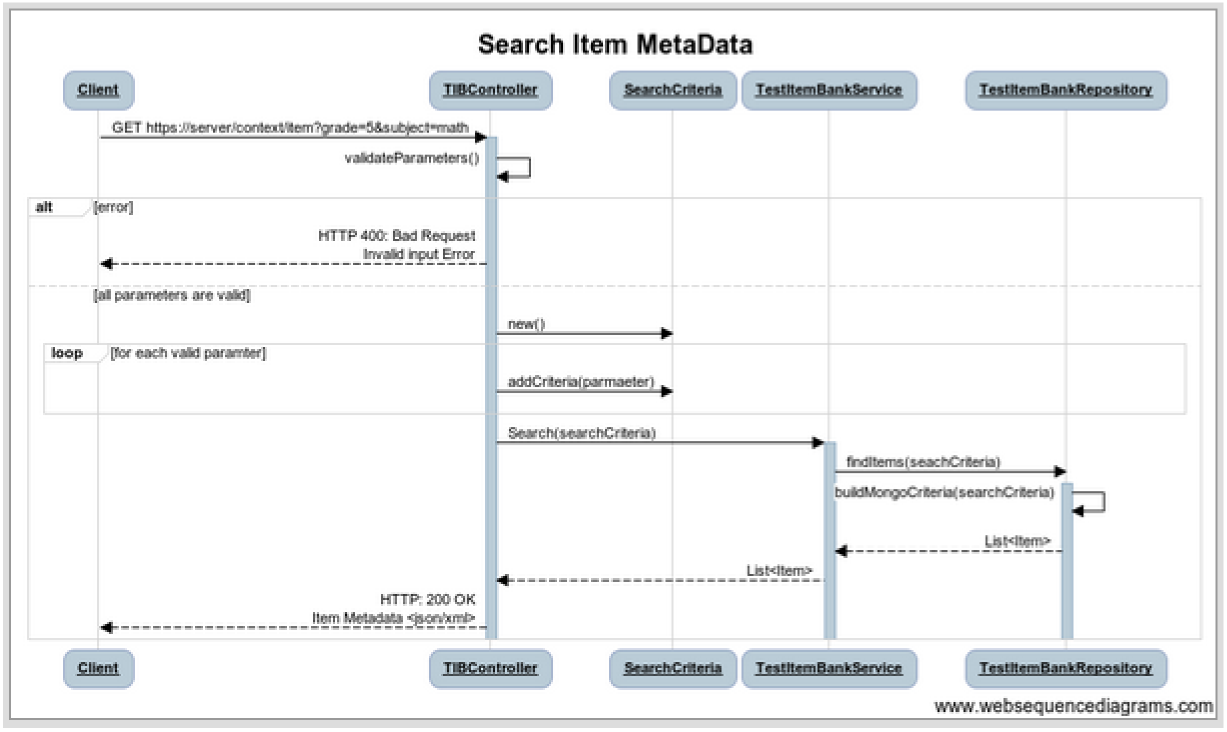
7.1 Sequence diagram – Get Items



7.2 Sequence diagram – Import Apip Items



7.3 Sequence diagram – Search Item MetaData



8. Implementation View

This view describes the organization of static software modules (source code, data files, executables, documentation etc.) in SBAC-11 test item bank component development in terms

- Source code, Packaging and layering
- Configuration Management (build and release strategy etc.)

8.1 Source Code

The source code for test item bank is implemented independent of other components. Within each component source repository is divided into three sub-projects as follow:

rest: controllers implementing the RESTful API
persistence: data workers and the mongoDb implementation
domain: objects shared by the data workers and controllers. The sb11-common project contains code that can be shared across multiple components. Dependencies on sb11-common are managed with maven.

8.2 Build Process

Test Item Bank is using maven to automate the compile/test/package process. A parent pom.xml at the top level defines common dependencies for each component. Child pom.xml files add dependencies unique to each sub-project.

8.3 Third party dependencies

Test Item Bank is relying on the following third party components:

- org.springframework: spring-beans, spring-context, spring-core, spring-web

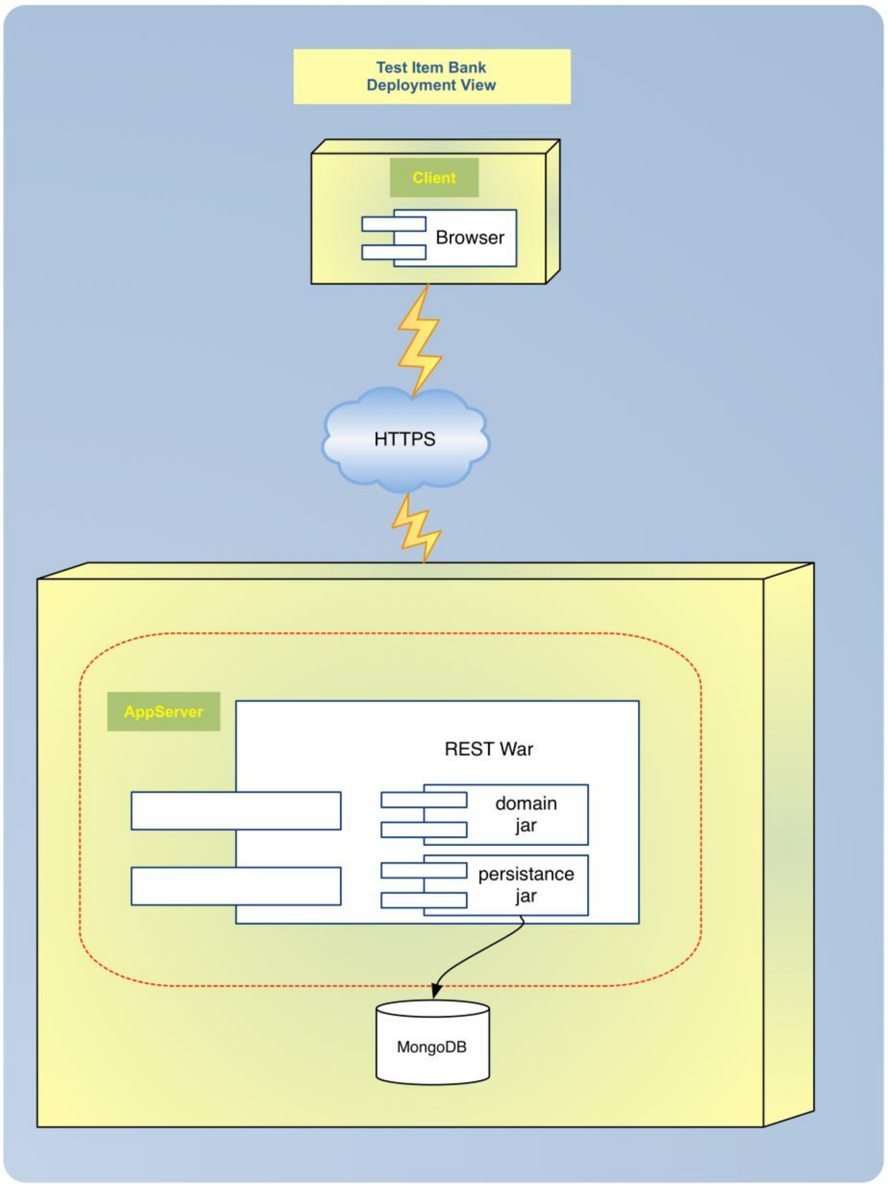
- org.springframework.data: spring-data-commons-core, spring-data-mongodb
- org.springframework.integration.spring-integration-core
- org.mongodb.mongo-java-driver
- javax.servlet: jstl, servlet-api, jsp.jsp-api
- slf4j & log4j
- cglib
- javax.inject
- org.aspectj.aspectjrt
- org.codehaus.jackson.jackson-mapper-asl
- org.hibernate.hibernate-validator

8.4 Test Automation Dependencies

Test Item Bank is leveraging following testing frameworks:

- Junit
- org.springframework spring-test
- org.springframework spring-test-mvc

9. Deployment View



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.

All web service endpoints will be secured using Spring Security Annotations. These annotations allow for role based security. Authentication and user/role mapping will be provided by a shared component,. Within the functional components, such as this one, the annotations will be configured to ensure the roles provided by the shared authentication component are in alignment. Spring Security handles insufficient authorization by denying access with a HTTP 403 error returned. Given that the shared component is currently not completed, the exact mechanism of security will remain uncompleted until the Authorization & Authentication can be integrated together.

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 item bank errors handling examples are:

POST: the service will return HTTP status 201 (created) with a LOCATION header that can be used to retrieve the document.

GET by id: return a single result. When an entity is found HTTP status is 200, otherwise 404.

GET with query params: return a list of results and always HTTP 200.

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.

12. Quality

12.1 Size & Performance

Test Item Bank is leveraging NoSQL mongoDB for storage which provides the following size & performance benefits:

- MongoDB eliminates object-relation-mapping work and the so-called “impedance mismatch by leveraging JSON object-style-data.
- Allows dynamic schemas or schemaless operations
- Easy to store and manipulate complex and polymorphic data
- Reduces the amount of work required to scale out the application and increase system speed
- GridFS for large file storage

12.2 Scalability

Test Item Bank’s persistence layer uses mongoDB which has auto-sharding capability to scale from a single server deployment to large, complex multi-site architectures.

12.3 Reliability

MongoDb has built-in replication with automated failover provides enterprise-grade reliability and operational flexibility.

12.4 Availability

Test Item Bank leverages monogDb’s built-in replication with automated failover to provide high availability