Software Technical Design Specification

for

SBAC-11 Program Management

Version 1.2

Revision History

Date	Version	Description	Author
09/26/2013	1.0	Initial Draft	Paul Krumrei
09/26/2013	1.1	Added content, diagrams	Mike Stern
09/30/2013	1.2	Added Content, Diagrams	Paul Krumrei

Table of Contents

Software Technical Design – Program Management

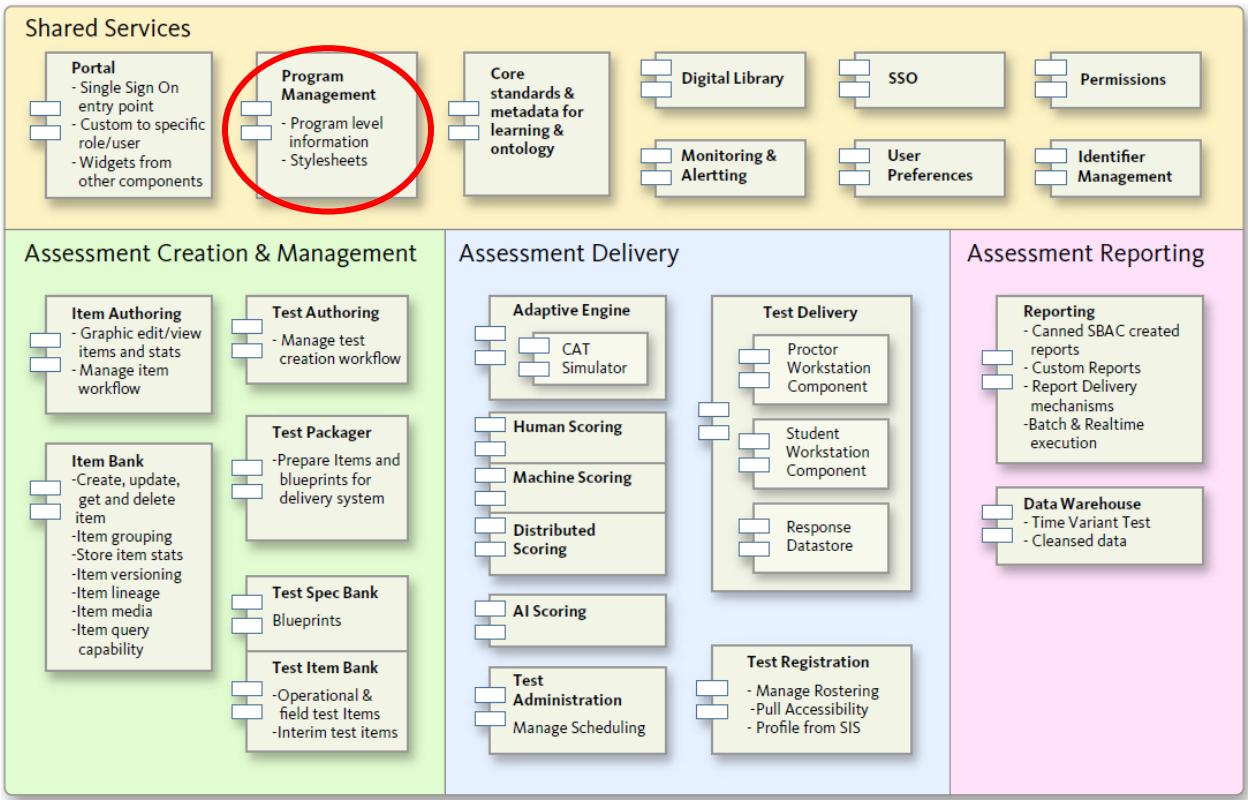
1. Introduction

1.1 **Purpose**

The objective of this document is to create SBAC-11 Project Management 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 'Program Management' which is highlighted in red.



2. Design Goals, Constraints and Assumptions

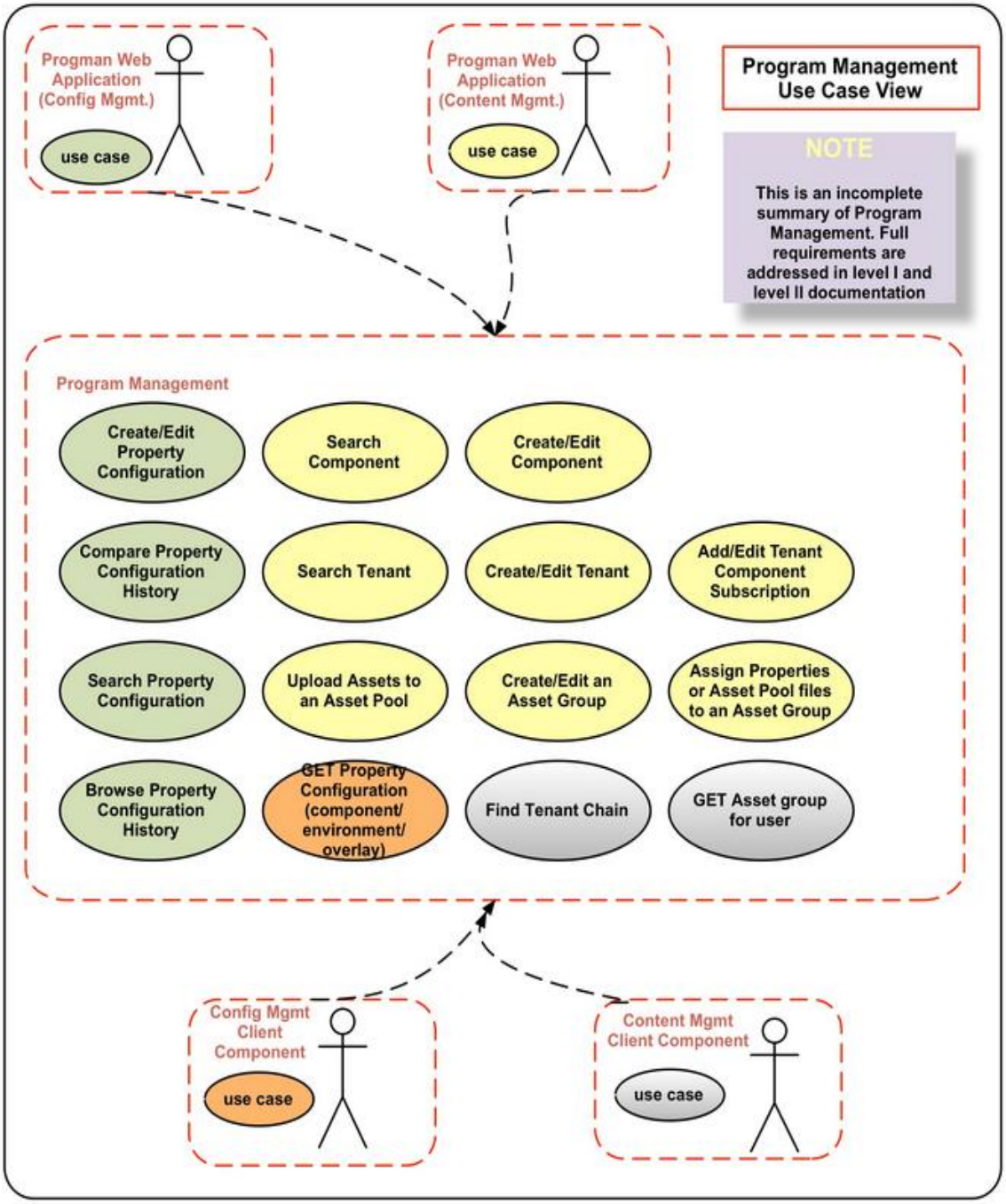
2.1 Design Goals

- Allow CSS, logos and other information needed for branding Smarter Balanced components for individual tenants
- Registration of components so that components know how to find endpoints for RESTful communications.
- Provide the ability to store system configuration
- associate specific content to a component and program

The Smarter Balanced architecture contains a segregation of where data is modified. The general rule is that data is only modified by the component that owns the data. For example: Item bank owns all Item related data and is the only component that can update this data. Other components consume parts of this data, but never update it. It is recommended that a simple custom solution be used instead of a commercial MDM product since the Smarter Balanced requirements do not demand a sophisticated system with features like "Single version of truth" [http://en.wikipedia.org/wiki/Single_version_of_the_truth] and data governance.

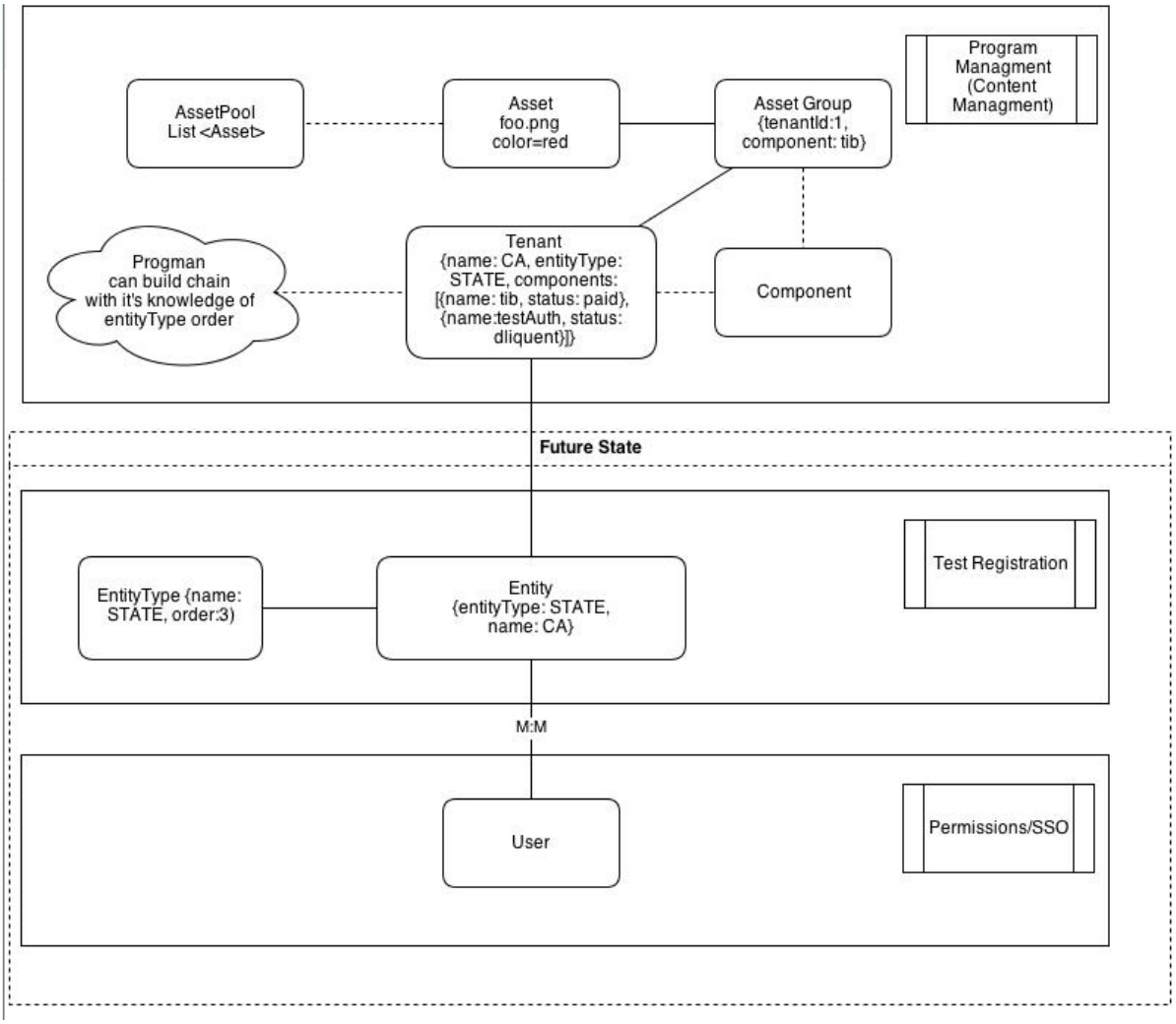
3. Use-Case View Figure

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


Progman Web Application (Config Mgmt.)
use case
Progman Web Application (Content Mgmt.)
use case
Program Management Use Case View
NOTE
This is an incomplete summary of Program Management. Full requirements are addressed in level I and level II documentation
Program Management
Create/Edit Property Configuration
Search Component
Create/Edit Component
Compare Property Configuration History
Search Tenant
Create/Edit Tenant
Add/Edit Tenant Component Subscription
Search Property Configuration
Upload Assets to an Asset Pool
Create/Edit an Asset Group
Assign Properties or Asset Pool files to an Asset Group
Browse Property Configuration History
GET Property Configuration (component/ environment/ overlay)
Find Tenant Chain
GET Asset group for user
Config Mgmt Client Component
use case
Content Mgmt Client Component
use case

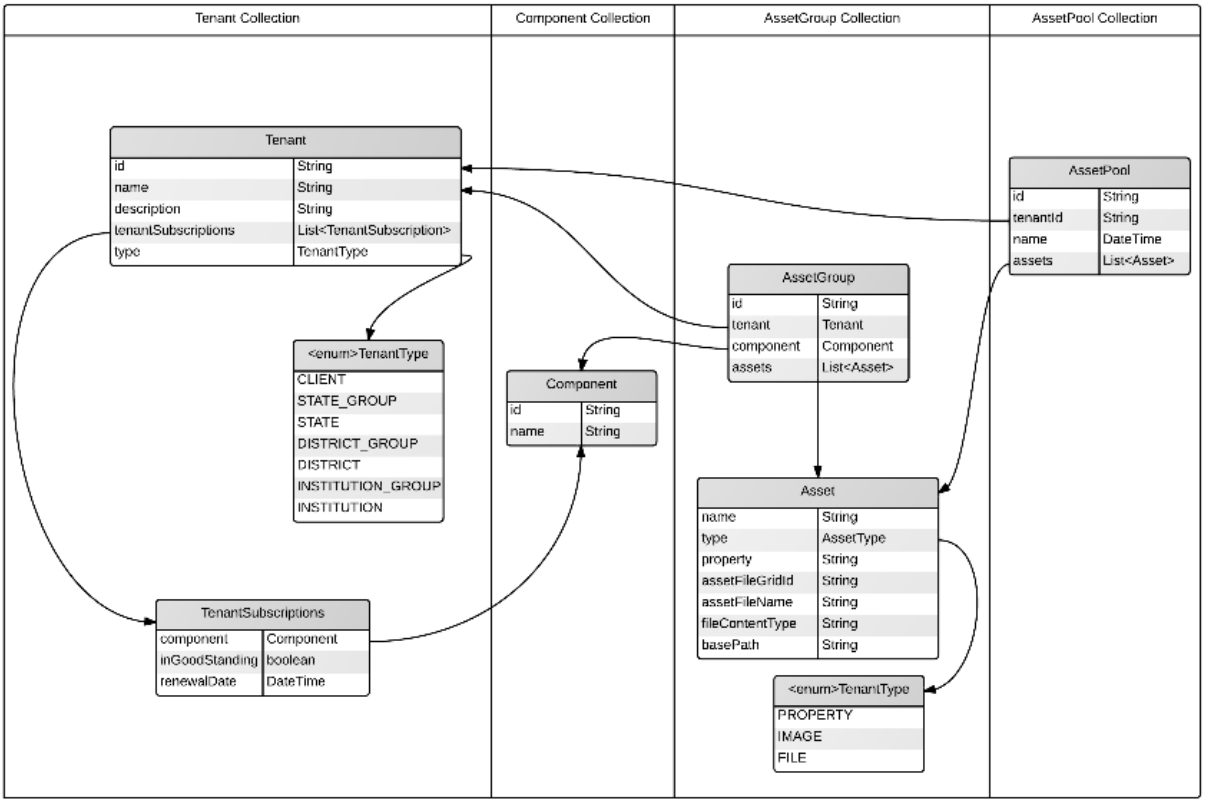
4. Process View

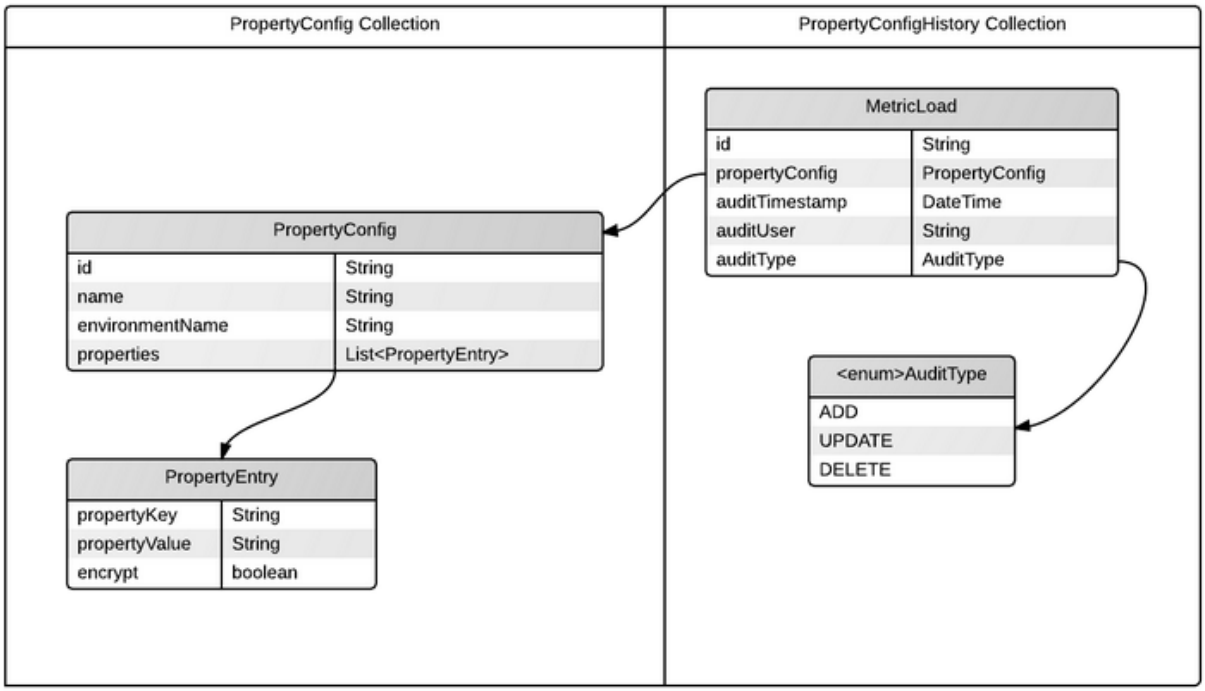
4.1 Project management component provides services to all components and is not part of a business process.



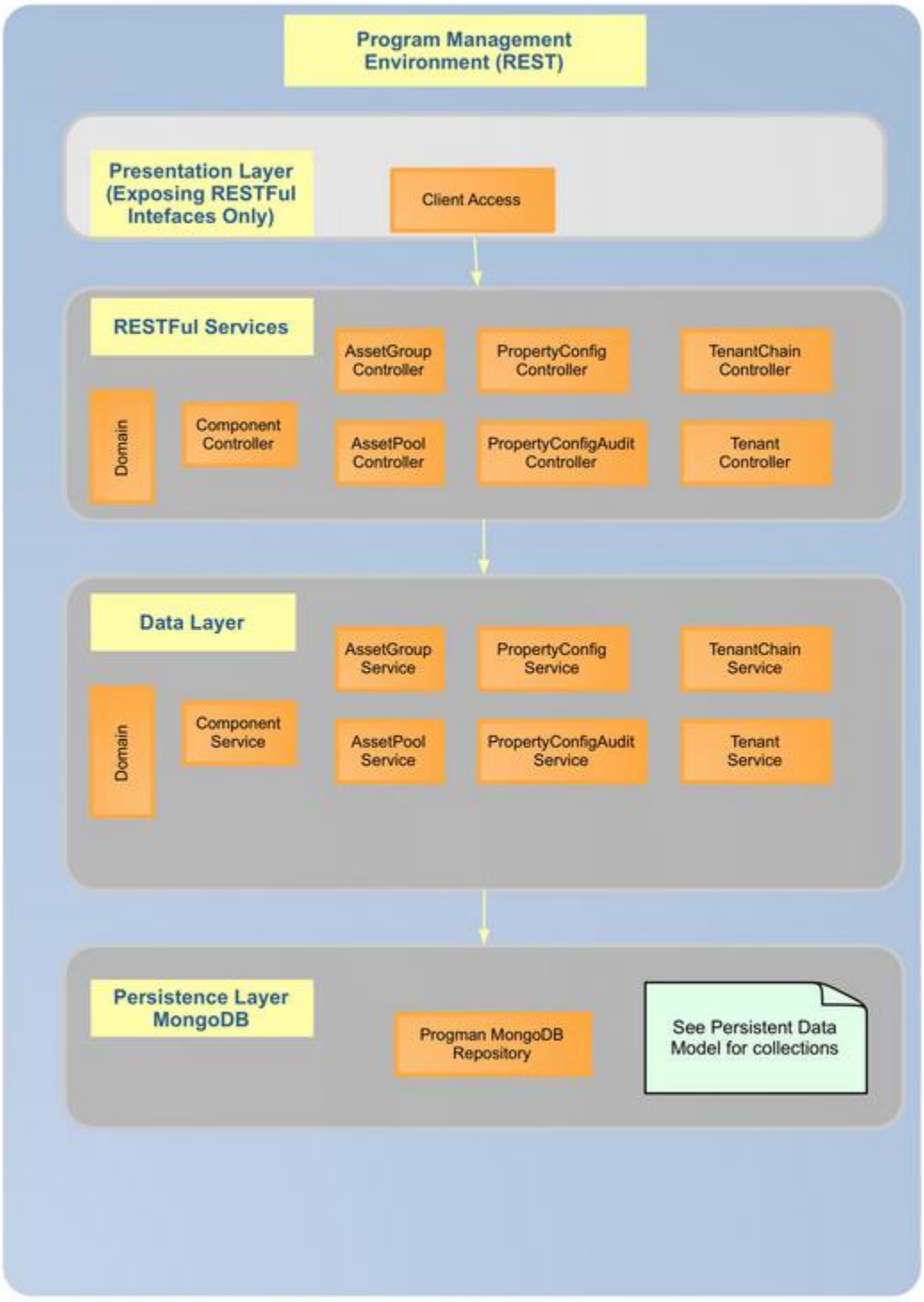
5. Data View

5.1 Program Management NoSQL physical schema consists of the following collections and nested documents





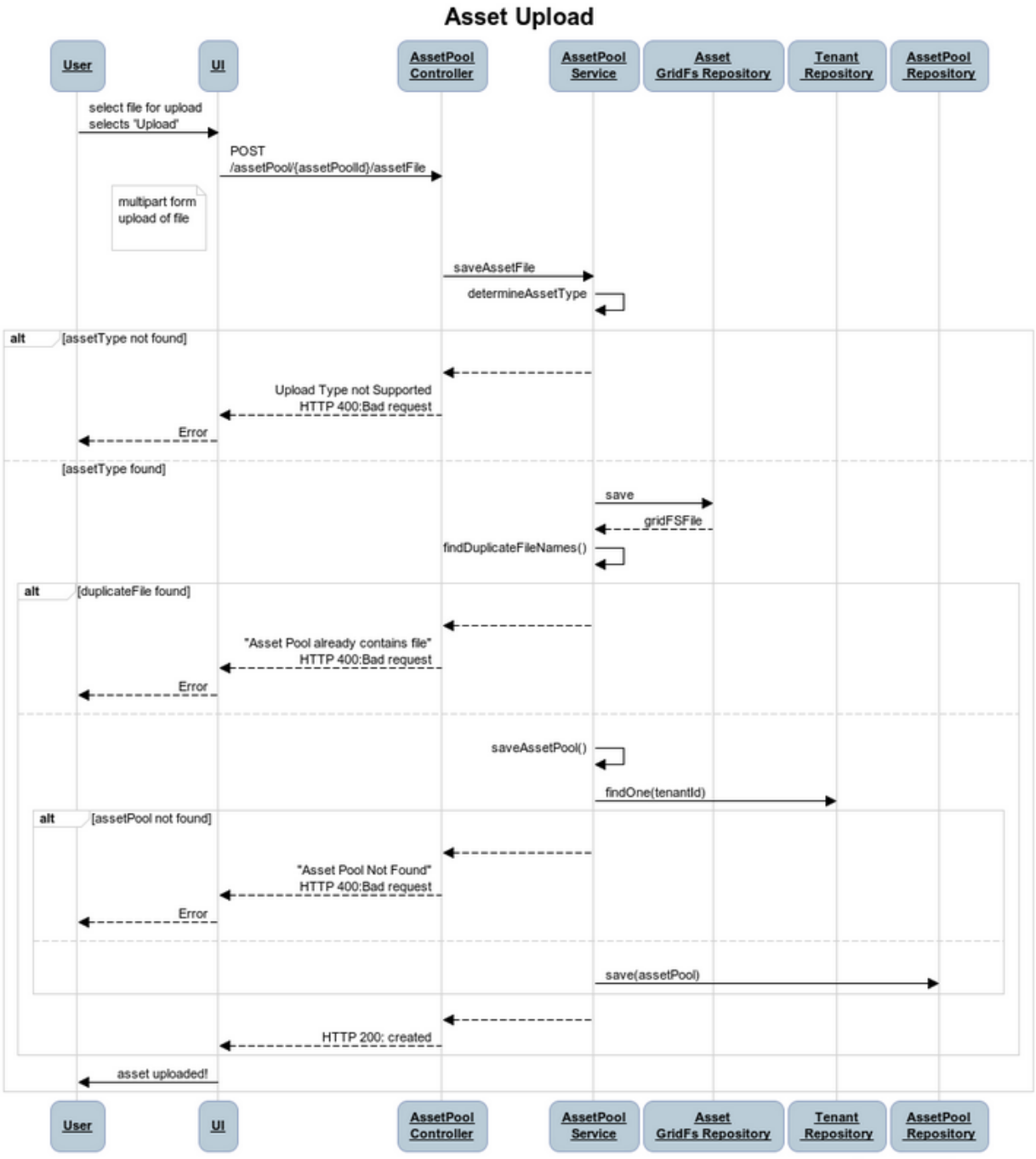
6. Component Layered View

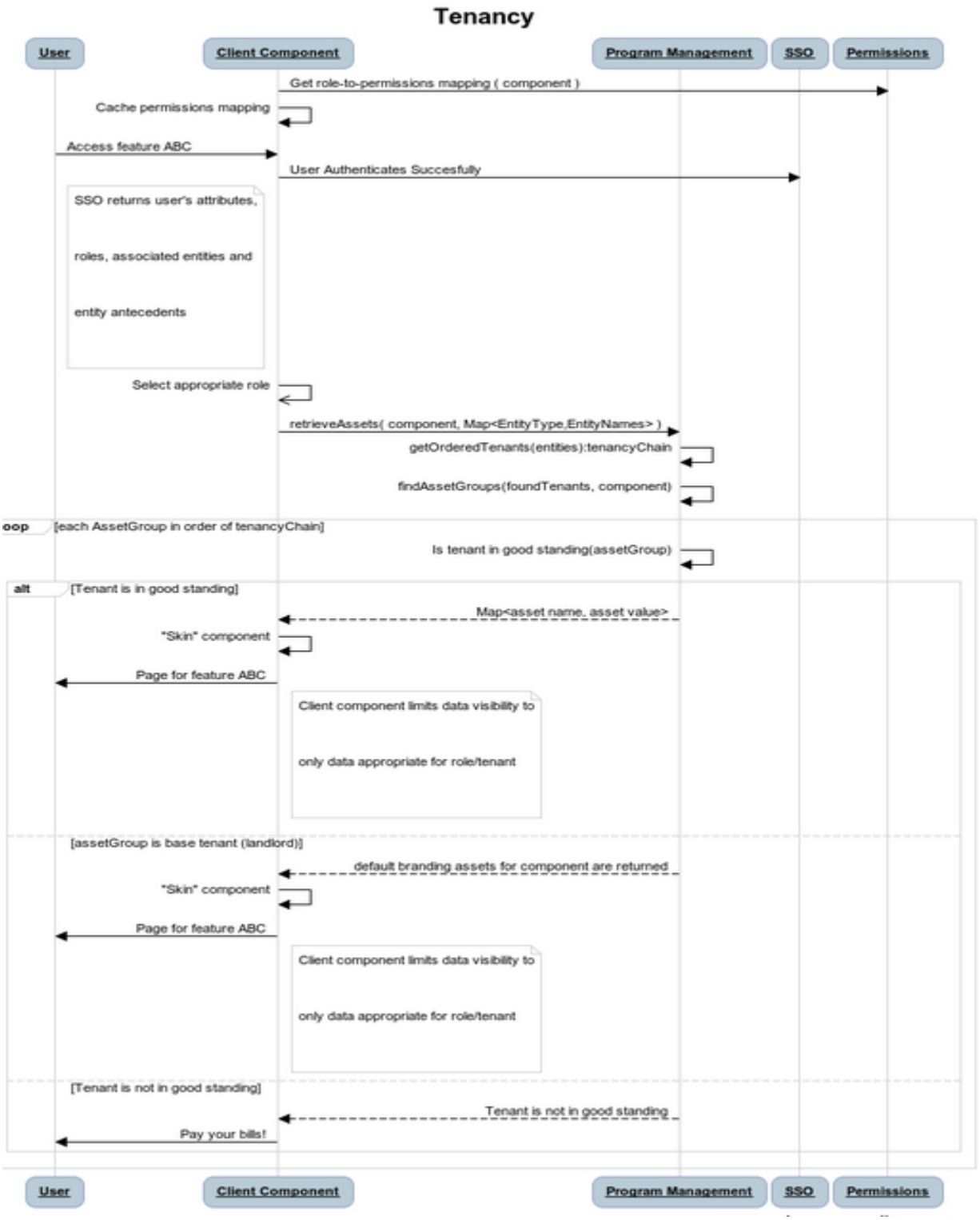


7. Dynamic View

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.

Update and Delete patterns have been omitted since the pattern is very similar to the Create with no significant design points of interest.




Tenancy
User
Client Component
Program Management
SSO
Permissions
Get role-to-permissions mapping (component)
Cache permissions mapping
Access feature ABC
User Authenticates Succesfully
SSO returns user's attributes,
roles, associated entities and
entity antecedents
Select appropriate role
retrieveAssets(component, Map<EntityType,EntityNames>)
getOrderedTenants(entities):tenancyChain
findAssetGroups(foundTenants, component)
oop
[each AssetGroup in order of tenancyChain]
Is tenant in good standing(assetGroup)
alt
[Tenant is in good standing]
Map<asset name, asset value>
"Skin" component
Page for feature ABC
Client component limits data visibility to
only data appropriate for role/tenant
[assetGroup is base tenant (landlord)]
default branding assets for component are returned
"Skin" component
Page for feature ABC
Client component limits data visibility to
only data appropriate for role/tenant
[Tenant is not in good standing]
Tenant is not in good standing
Pay your bills!
User
Client Component
Program Management
SSO
Permissions

8. Implementation View

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

8.1 Source Code

The source code for Program Management 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 (prog-mgmnt.rest-VERSION.war) that provides REST endpoints that can be used to access and modify Program Management data. The REST module has an internal dependency to the SB11 Program Management Persistence module. ; following the same as the webapp module.
- **Webapp Module:** The Webapp module is a deployable WAR file (prog-mgmnt.webapp-VERSION.war) that provides the administrative UI for Program Management 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.
- **Persistence Module:** The Persistence module is a JAR artifact that is used by the REST module. This module is responsible for persistence of application data. It also encrypts and decrypts sensitive data.
- **Domain Module:** The domain module contains all of the domain beans used to model the Program Management data as well as code used as search beans to create Mongo queries. It is a JAR artifact that is used by other modules.
- **Client Modules:** There are three modules that make up the client:
  - The Client Interfaces module contains the interface classes for the client, AOP advice and point cuts, bootstrap initialization
  - The Null Client module is a client implementation that sends all messages to the local logging subsystem (SLF4J)
  - The Integrated Client sends all messages to the advertised REST endpoints from the REST module

8.2 Build Process:

Program Management 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

- Compile Time Dependencies
- Apache Commons IO
- Apache Commons Beanutils
- Jackson Data type Joda
- Google Guava
- Hibernate Validator
- Apache Commons File Upload
- Jasypt
- SB11 Shared Code
- Logback

- SLF4J
- JCL over SLF4J
- Spring Core
- Spring Beans
- Spring Data MongoDb
- Mongo Data Driver
- Spring Context
- Spring WebMVC
- Spring Web
- Spring Aspects
- AspectJ RT
- AspectJ Weaver
- Javax Inject
- Apache HttpClient
- JSTL API
- Apache Commons Lang
- Joda Time
- Jackson Core
- Jackson Annotations
- Jackson Databind
- SB11 REST API Generator
- JSTL
- Apache Commons Lang

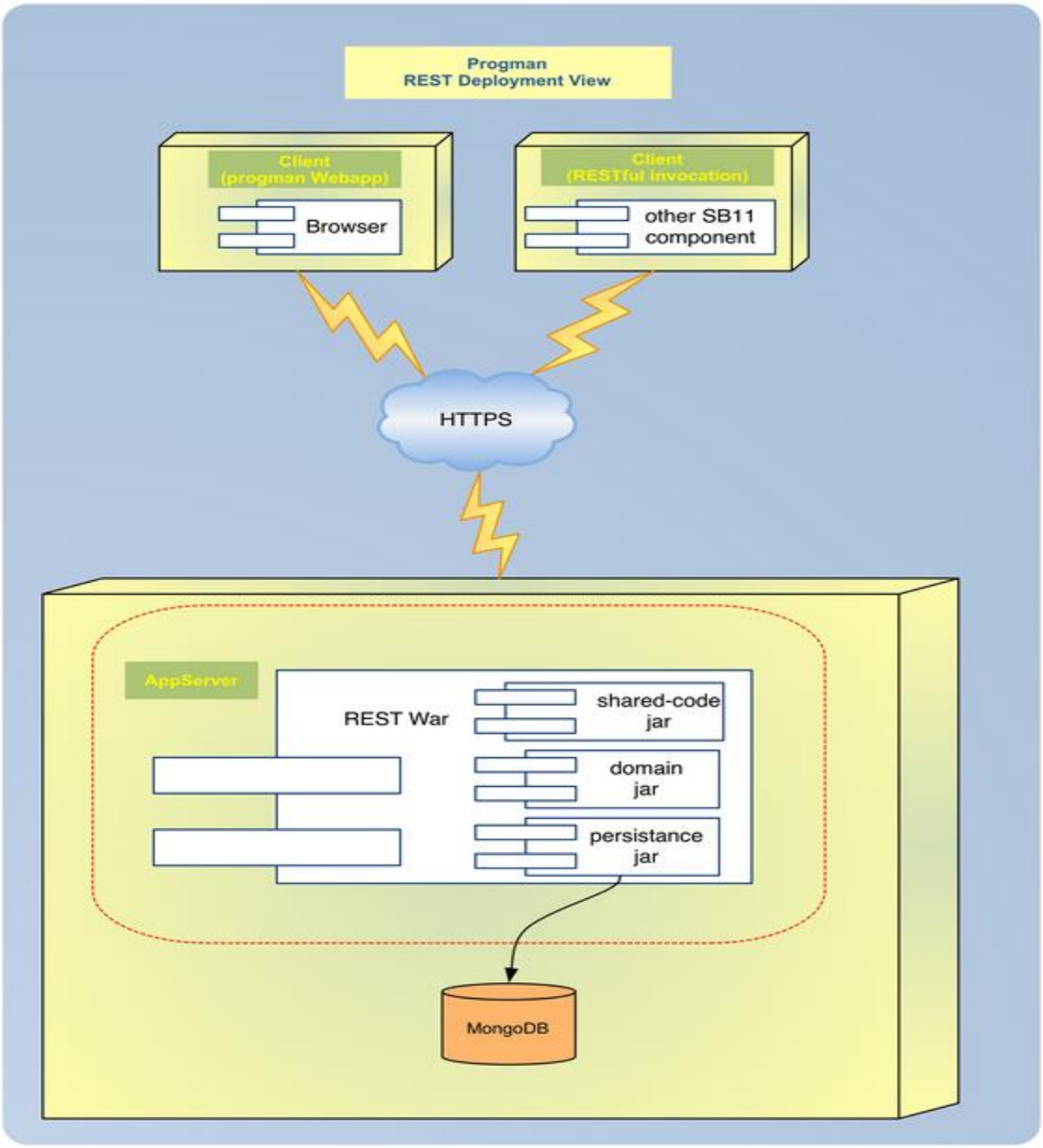
Test Dependencies

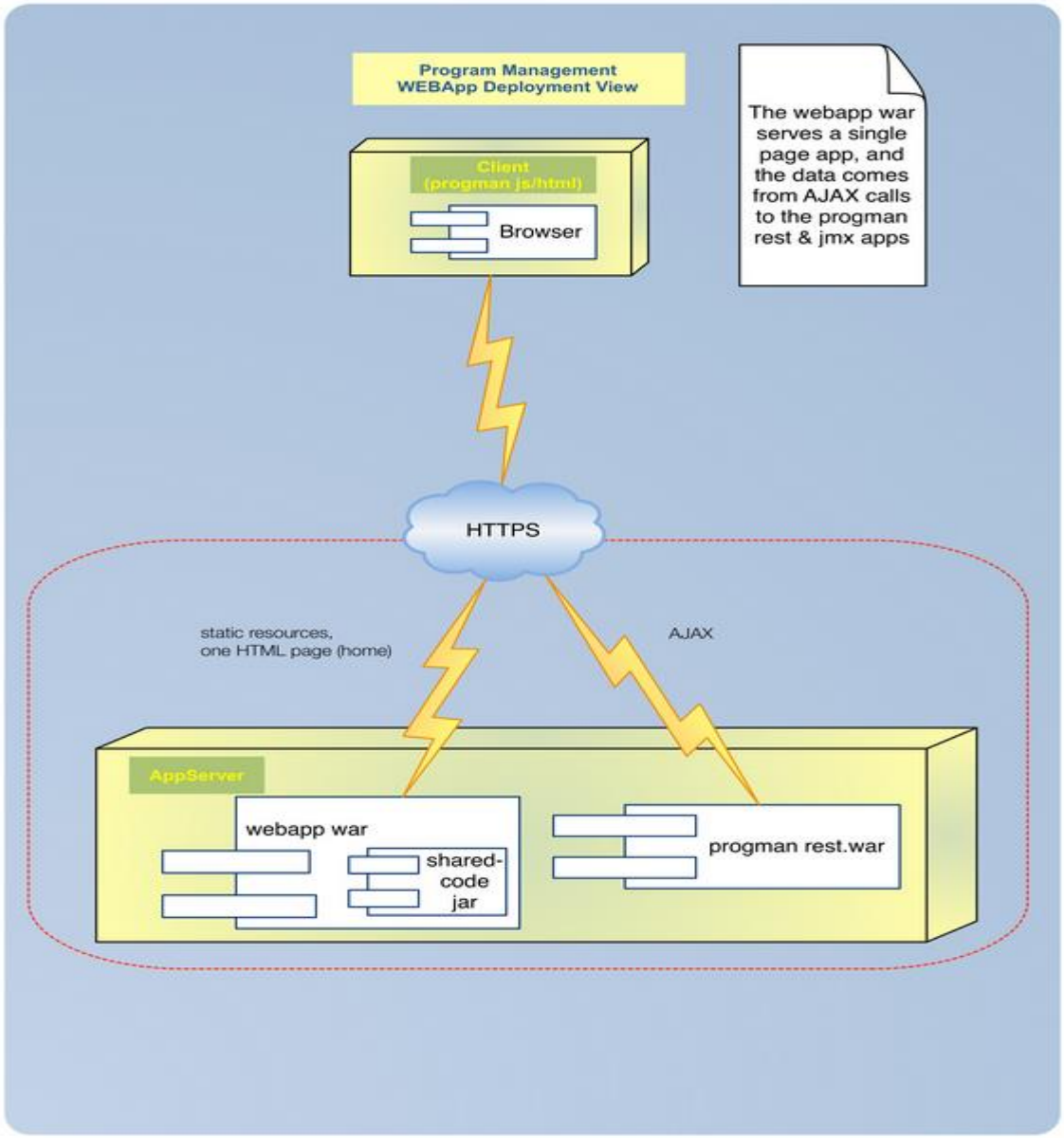
- Spring Test
- Hamcrest
- JUnit 4
- Flapdoodle
- Podam
- Log4J over SLF4J

Runtime Dependencies

- Servlet API
- Persistence API

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 Program Management 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 Program Management RESTful services allows horizontal scaling. Program Managements 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

Program Management 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 Program Management 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 Program Management component to provide failover.

12.3 **Availability**

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

13. API

Once the Program Management component is installed and running, the API can be viewed by entering one of the following links:

API Context Roots

- **/rest/api**
- **/rest/api/help**
- **/rest/api/tenants**
- **/rest/api/assetPool**
- **/rest/api/tenantchain**
- **/rest/api/propertyConfig**
- **/rest/api/component**
- **/rest/api/tenantTypes**
- **/rest/api/tenant**
- **/rest/api/assetGroup**
- **/rest/api/tenantsBySearchVal**
- **/rest/api/skinnableAssets**