***By no means authoritative and it definitely needs some work, however this document might be a good start point for a developer that has nothing to start with. Intent is to inspire thought and consideration more than be a silver bullet.***

Date: ISO 8601 (YYYYMMDD)

# My Project (MyProj) Design Document: Application Development and Security STIG v4r2: APSC-DV-000960, APSC-DV-000970, APSC-DV-002950, APSC-DV-003220

**Some Contract**

**Task Order – DO 0001**

**Prepared by**

**Some Special Group**

**For**

**Customer “X”**

Version 1.0

Table of Contents

[My Project (MyProj) Design Document: Application Development and Security STIG v4r2: APSC-DV-000960, APSC-DV-000970, APSC-DV-002950, APSC-DV-003220 1](#_Toc119923424)

[Executive Summary 9](#_Toc119923425)

[Introduction 10](#_Toc119923426)

[1.1 Development Strategy 10](#_Toc119923427)

[1.2 Hardware Configuration Items 11](#_Toc119923428)

[1.3 Data Communications 12](#_Toc119923429)

[1.4 Software Configuration Items 13](#_Toc119923430)

[1.5 Data Design 14](#_Toc119923431)

[1.5.1 Data Objects and Resultant Data Structures 14](#_Toc119923432)

[1.5.2 File and Database Structures 14](#_Toc119923433)

[1.6 Data 14](#_Toc119923434)

[1.6.1 Data Flow Diagram 14](#_Toc119923435)

[1.6.2 Categories of Data Sensitivity 17](#_Toc119923436)

[2.0 Architecture 18](#_Toc119923437)

[2.1 Approved Architecture 18](#_Toc119923438)

[2.2 Communications 18](#_Toc119923439)

[2.3 Roles 19](#_Toc119923440)

[3.0 Requirements 20](#_Toc119923441)

[4.0 Software Specifications 21](#_Toc119923442)

[4.1 System Environment and System Components 21](#_Toc119923443)

[4.1.1 Managed Software Domain 21](#_Toc119923444)

[4.1.2 <explicit software package name, like Tomcat or JBoss> 21](#_Toc119923445)

[4.2 Development Technique 22](#_Toc119923446)

[4.2.1 SDLC 22](#_Toc119923447)

[4.2.2 Standards 22](#_Toc119923448)

[4.2.3 Process Asset Library (PAL) 22](#_Toc119923449)

[4.2.4 Software Configuration Management (APSC-DV-003010) 23](#_Toc119923450)

[4.4 Build Environment 24](#_Toc119923451)

[4.5 Introduction Coding Standards 24](#_Toc119923452)

[4.5.1 Definition 24](#_Toc119923453)

[4.5.2 Scope 25](#_Toc119923454)

[4.5.1 Why Coding Standards? 25](#_Toc119923455)

[4.5.2 Coding Standards Packages - Java 25](#_Toc119923456)

[4.6 Reusable Objects 33](#_Toc119923457)

[4.6.1 Modules 33](#_Toc119923458)

[4.6.2 Mobile Code 33](#_Toc119923459)

[4.6.3 References 34](#_Toc119923460)

[4.7 Transferring Code from Development to Deployment (APSC-DV-003010) 35](#_Toc119923461)

[4.7.1 Deployment 35](#_Toc119923462)

[5.0 Security 36](#_Toc119923463)

[5.1 Definition 36](#_Toc119923464)

[5.2 Scope 36](#_Toc119923465)

[5.3 Categories 37](#_Toc119923466)

[5.3.1 Security Technical Implementation Guide (STIG) 37](#_Toc119923467)

[5.3.2 Authentication 37](#_Toc119923468)

[5.3.3 Unique Security Requirements 37](#_Toc119923469)

[5.3.4 Open Web Application Security Project (OWASP) 38](#_Toc119923470)

[5.3.5 Common Weakness Enumeration (CWE) 38](#_Toc119923471)

[5.3.5 National Institute of Standards and Technology (NIST) 38](#_Toc119923472)

[5.4 References 39](#_Toc119923473)

[5.6 Threat Assessment (APSC-DV-003230) 40](#_Toc119923474)

[5.6.1 DREAD 41](#_Toc119923475)

[5.6.3 STRIDE 42](#_Toc119923476)

[5.7 Security Tools (APSC-DV-001460) 43](#_Toc119923477)

[5.8 Application Configuration Guide ~ ACG (APSC-DV-003285) 43](#_Toc119923478)

[5.9 References 43](#_Toc119923479)

[6.0 Web Site Design 45](#_Toc119923480)

[6.1 Web Design Standards and Guidelines 45](#_Toc119923481)

[6.2 Part I - Design Standards 45](#_Toc119923482)

[6.2.1 Standard HTML 5.0 45](#_Toc119923483)

[6.2.2 CSS version 2 46](#_Toc119923484)

[6.3 References 46](#_Toc119923485)

[7.0 Metrics 47](#_Toc119923486)

[7.1 Definition 47](#_Toc119923487)

[7.2 Scope 47](#_Toc119923488)

[7.3 Categories 47](#_Toc119923489)

[7.3.2 Issues (during Retrospective or Post-Mortem function) 47](#_Toc119923490)

[7.3.3 Lines of Code 47](#_Toc119923491)

[7.4 References 48](#_Toc119923492)

[8.0 Testing 49](#_Toc119923493)

[8.1 Definition 49](#_Toc119923494)

[8.2 Scope 49](#_Toc119923495)

[8.3 Categories 49](#_Toc119923496)

[8.3.1 Functional vs. Non-functional Testing 49](#_Toc119923497)

[8.3.2 Defects and Failures 50](#_Toc119923498)

[8.3.3 Compatibility 50](#_Toc119923499)

[8.3.4 Software Verification and Validation 50](#_Toc119923500)

[8.4 Testing Levels 51](#_Toc119923501)

[8.4.1 Unit Testing 51](#_Toc119923502)

[8.4.2 Integration Testing 51](#_Toc119923503)

[8.4.3 System Testing 51](#_Toc119923504)

[8.5 Test Plans 52](#_Toc119923505)

[8.5.1 Named Tester/Test Team (APSC-DV-003130) 53](#_Toc119923506)

[8.5.2 Test Procedures 54](#_Toc119923507)

[8.6 References 55](#_Toc119923508)

[9.0 Glossary of Terms 56](#_Toc119923509)

[10.0 References 57](#_Toc119923510)

[A.1 Classification Guide (APSC-DV-003290) 58](#_Toc119923511)

[A.2 Enclaves/Communications (APSC-DV-003285) 58](#_Toc119923512)

[A.2.1 Ports and Protocols 58](#_Toc119923513)

[A.2.2 PKI Certificate Configuration Settings 58](#_Toc119923514)

[A.3 Encryption Settings 58](#_Toc119923515)

[A.4 Password Settings 58](#_Toc119923516)

[A.4.1 Architecture Components Passwords (Service Accounts) 58](#_Toc119923517)

[A.4.2 Application Passwords 58](#_Toc119923518)

[A.4.3 Database Passwords 58](#_Toc119923519)

[A.5 Audit Settings 58](#_Toc119923520)

[A.5.1 Audit Standards 58](#_Toc119923521)

[A.5.2 Log Locations 58](#_Toc119923522)

[A.4 Standardized Environment (APSC-DV-003215) 58](#_Toc119923523)

[A.5 Best Practices (APSC-DV-003215) 58](#_Toc119923524)

[A.6 Deployment Plan (APSC-DV-003130, APSC-DV-003160) 59](#_Toc119923525)

[A.7 Deployment Settings (APSC-DV-003285) 59](#_Toc119923526)

[A.7.1 Known Dependencies 59](#_Toc119923527)

[A.8 Operational Backups 59](#_Toc119923528)

[A.8.1 Database Backups 59](#_Toc119923529)

[A.9 Security Assumptions/Threat Model 59](#_Toc119923530)

[B.1 Deployment Plan 60](#_Toc119923531)

[C.1 Dread Implementation 62](#_Toc119923532)

[Security Misconfiguration 62](#_Toc119923533)

[Insecure Cryptographic Storage 62](#_Toc119923534)

[Insufficient Transport Layer Protection 62](#_Toc119923535)

[C.2 Stride Implementation 63](#_Toc119923536)

[Spoofing Identity 63](#_Toc119923537)

[Tampering with Data 63](#_Toc119923538)

[Repudiation 63](#_Toc119923539)

[Information Disclosure 63](#_Toc119923540)

[Denial of Service 63](#_Toc119923541)

[Elevation of Privilege 63](#_Toc119923542)

[D.1 Definition 64](#_Toc119923543)

[D.2 Scope 64](#_Toc119923544)

[D.3 Database Cluster architecture 64](#_Toc119923545)

[D.4 Configuration Management 64](#_Toc119923546)

[D.5 Security 64](#_Toc119923547)

[D 5.1 Authentication 65](#_Toc119923548)

[D 5.2 Authorization & Authentication (Role Based Access Control [RBAC]) 65](#_Toc119923549)

[D.6 Transaction logs 65](#_Toc119923550)

[D.7 Database Backup 65](#_Toc119923551)

[System level backup 65](#_Toc119923552)

[Database level backup 65](#_Toc119923553)

[Application level backup 65](#_Toc119923554)

[D.8 Data Communications 65](#_Toc119923555)

[D.9 Data Objects and Resultant Data Structures 66](#_Toc119923556)

[D.10 Entity Relationship Diagram 67](#_Toc119923557)

[D.11 References 67](#_Toc119923558)

[E.1 Required Software for Development 68](#_Toc119923559)

[E.1.1 Integrated Development Environments (IDE) 68](#_Toc119923560)

[E1.2 Revision Control System Subversion 68](#_Toc119923561)

[E.1.3 Developer Tools 69](#_Toc119923562)

[E.1.4 References 69](#_Toc119923563)

[E.2 Required Libraries 70](#_Toc119923564)

Table of Figures

[Figure 1 - Communications 12](#_Toc119923565)

[Figure 2 - Data Flow Diagram 15](#_Toc119923566)

[Figure 3- Sequence Diagram 16](#_Toc119923567)

[Figure 4 - Communications 18](#_Toc119923568)

[Figure 5 - Deployment Path 35](#_Toc119923569)

[Figure 6- Entity Relationship Diagram (ERD) 67](https://usdagcc-my.sharepoint.com/personal/christopher_wood3_usda_gov/Documents/001_MANAGEMENT/TEMPLATES/Template_DesignDocument.docx#_Toc119923570)

Table of Tables

[Table 1 - Hardware Configuration Items (CI's) 11](#_Toc119923571)

[Table 2 - Software Domain 13](#_Toc119923572)

[Table 3 - Roles Definition 19](#_Toc119923573)

[Table 4 - Roles Definition 19](#_Toc119923574)

[Table 5 - SDLC Process Diagram 22](#_Toc119923575)

[Table 6 - Mobile Code 34](#_Toc119923576)

[Table 7- Designated Testers 53](#_Toc119923577)

[Table 8 - Library Dependencies 70](#_Toc119923578)

History of Work Performed

|  |  |  |
| --- | --- | --- |
| Version Number | Date | Description |
| 1.0 – alpha | 2014/06 | Initial Draft |
| 2.0 – beta | 2017/05 | Updated from culmination of multiple design document reviews from different sources and AppDev STIG v4r3 release. |

# Executive Summary

The purpose of this document is to provide a detailed description of the technical requirements, components, and development practices associated with software development projects for Fleet Synthetic Training (FST). This project's design will be laid out here in detail as a blueprint for project implementation. The following sections contain both detailed information about a software concept and the actions to be taken in order to create the system environment as well as details on all components used within the system.

# Introduction

<provide a description of your project (My Project), this description should likely come from a requirements document, a project plan, a Statement of Work (SOW), or historical material.>

Example: From a functional perspective, FST is focused on creating data/content sufficient to support warfare exercises. This design document is intended to provide an overall standard by which all software projects will be developed. Most development will be in support of Naval METOC Data Services Framework (PROGRAM 1) or discrete applications that perform support functions for FST staff. Where appropriate each section will reference a particular software package if the effort has to deviate from the standard.

## 1.1 Development Strategy

<describe how you’re going to develop. If using DevOps talk about the various environments and the tools used to make DevOps happen. Explain in plain English how you’re going to develop this code base.>

* **Environment**
  + DEV – DEVELOPER lab personal computers and virtual machine servers
  + Staging – Not appropriate.
  + Operations – Typically NMCI SIPR desktops
* **Processes & Policies**
  + There are no discrete processes or policies required by the sponsor; as such development will utilize WI-301-BSL (software development process).
* **Communications**
  + Developers will contact the sponsor as needed for clarification on any issues encountered.
  + Bi-Weekly meetings with the sponsor will be used to communicate status or request clarification.
  + Weekly reports will provide status to the sponsor.

## 1.2 Hardware Configuration Items

Desktops will be the target architecture for the smaller applications; however Program1 has a specific set of requirements which are detailed in the software suite’s documentation maintained by Program 1’s Organization.

<provide details specific for the development environment and execution environment for hardware that is specific to this project. >

|  |  |  |  |
| --- | --- | --- | --- |
| Hardware Item | Configuration Identifier | Description | Comments |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table - Hardware Configuration Items (CI's)

## 1.3 Data Communications

Various data communications protocols will be utilized to support the overall effort however primary focus will be Port 443 (HTTPS) for any Navy METOC Data Services Framework (PROGRAM 1) or similar Machine to Machine (MTM) interaction efforts worked on. Communication to the web server will utilize SSL enforced by DoD PKI.

Smaller applications will not utilize any ports other than 443 (HTTPS).

< Provide a diagram of communications.>

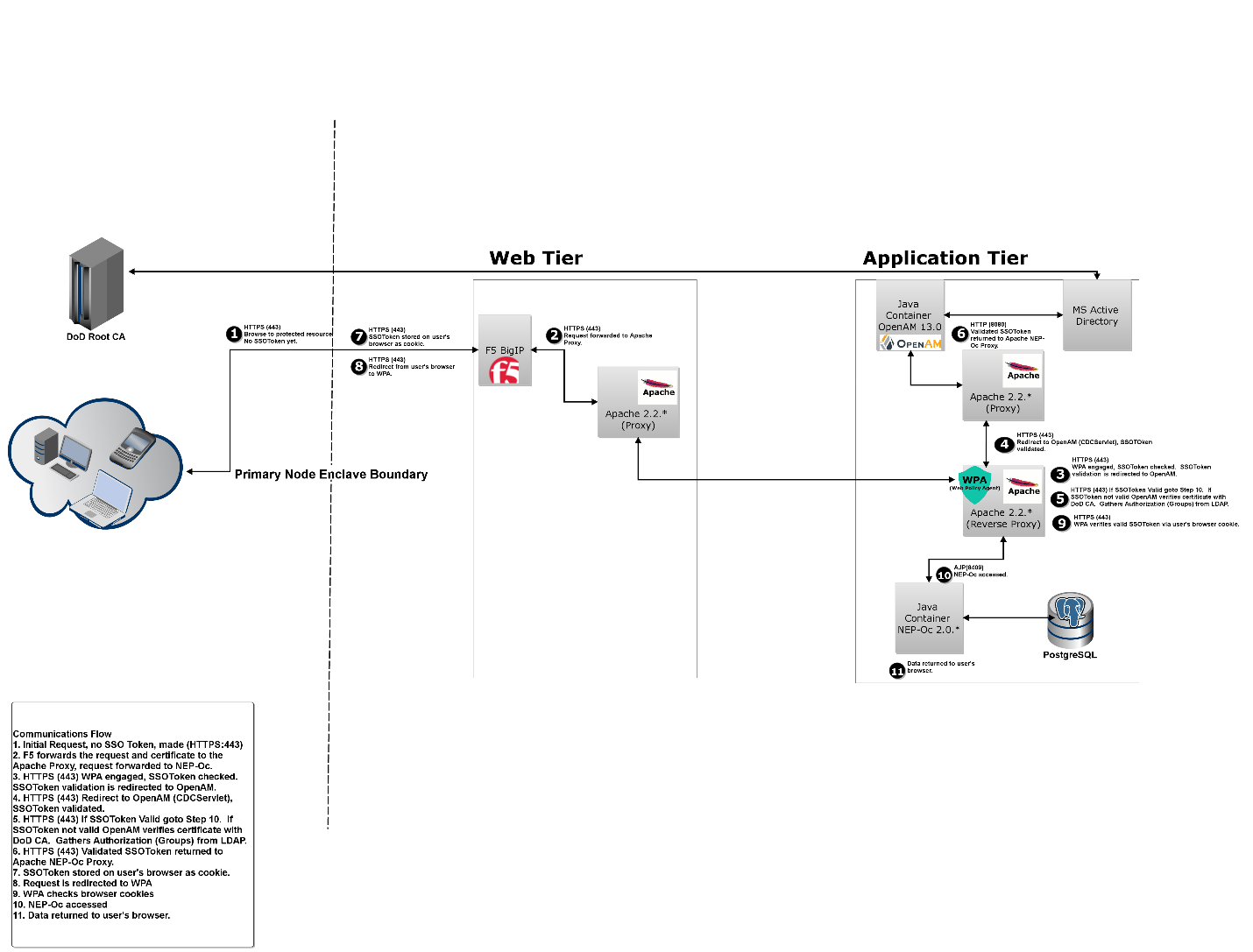


Figure - Communications

## 1.4 Software Configuration Items

Java will be used for all applications developed for Program 1 tasking. DADMS compliance will be adhered to which currently requires version 1.7.\*.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Server  Name | App  Name | Version | Function | FAM Approved | DADMS  Id | LDA | Deployed  (Yes | No) | DEV  ( Yes | No) |
| name | IIS | \* | Web/App Server | \* | \* | \* |  |  |
| name | MS SQL Server | \* | Database Server | \* | \* | \* |  |  |

Table - Software Domain

## 1.5 Data Design

### 1.5.1 Data Objects and Resultant Data Structures

Object Oriented Programming techniques will be utilized in all application development. Objects will represent more complex data structures starting with Plain Old Java Objects (POJO’s) to include the use of Interfaces.

Java Collections will be utilized whenever possible to help represent lists of data.

### 1.5.2 File and Database Structures

No database structure is anticipated for the smaller applications.

Program 1 has its own standards for files/database and its documentation will dictate utilization.

Files which contain delimited content will utilize “^” symbology to separate data

<Reference a database with Entity Relationship Diagram (ERD) as an Appendix. Include references to any documentation related to the database such as a Policy document or Configuration management document required by hosting agency. DO NOT embed the diagram in the middle of this document.>

## 1.6 Data

### 1.6.1 Data Flow Diagram

The following diagram illustrates how the data flows from client browser to persist in the database and subsequently return a response to the user. <provide any details you feel are appropriate to support the data flow diagram(s), sequence diagrams likely shouldn’t cover every possible service call but should show in a generic way how data is processed with emphasis on validation/security/etc. If you have many diagrams considering making an Appendix listing document filenames with URI and simply leave the image/graphic as a stand-alone artifact.>

<Explain what type of data is exchange. For a web application if HTTP Requests and Responses are exchanged between client and browser then say that. On the backend between application server and database or other application servers explain the exchange of data. For a Java Container exchanging data with a PostgreSQL database explain that port 5432 is exchanging data with PostgreSQL’s format to a Hibernate Object Relation Mapping (ORM) Object using one way encryption with the server’s PKI certificate.>

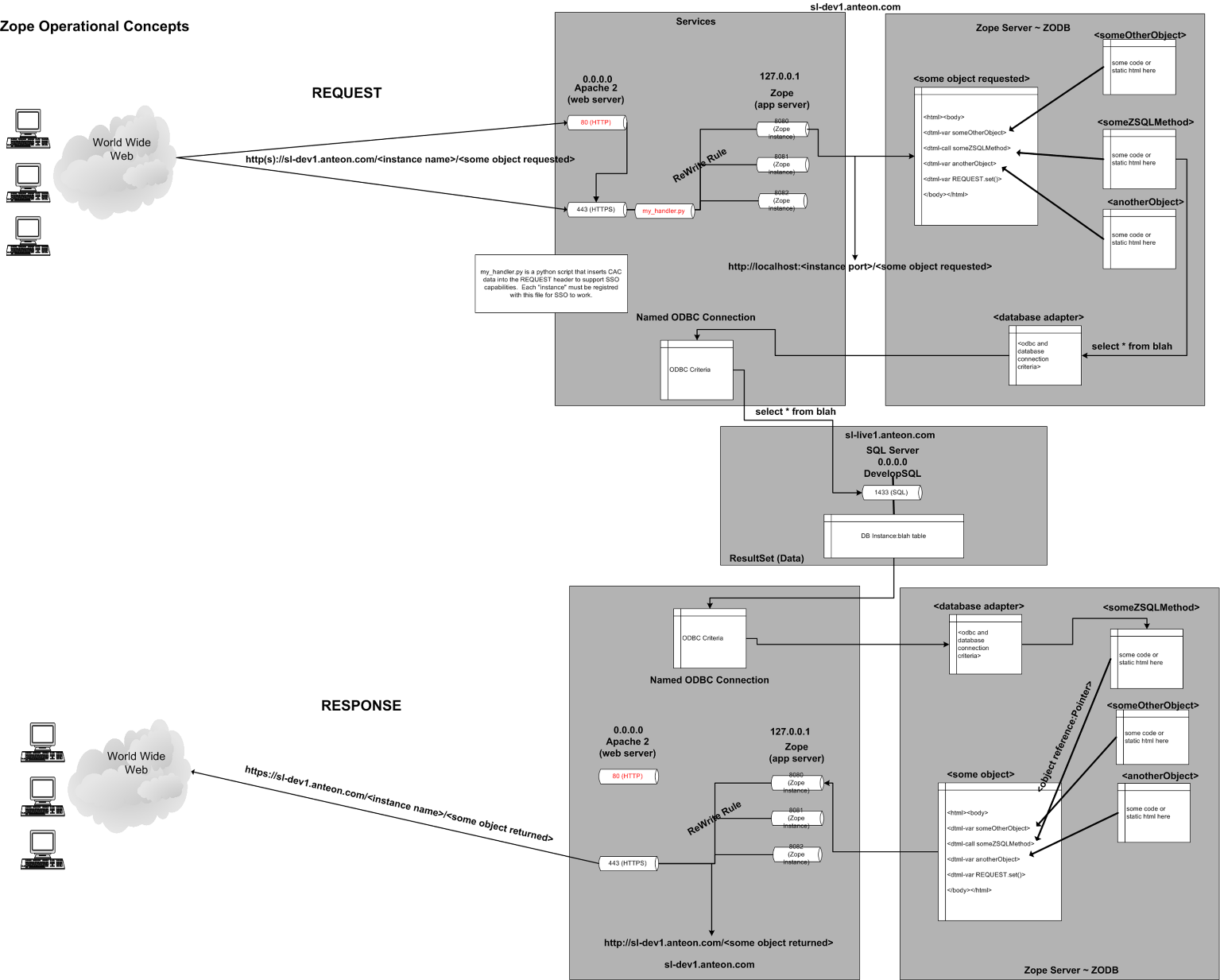
Figure - Data Flow Diagram



Figure - Sequence Diagram

### 1.6.2 Categories of Data Sensitivity

# 2.0 Architecture

## 2.1 Approved Architecture

PROGRAM 1 has a specific architecture that will be adhered to whereas smaller applications will be simple desktop applications.

Example: ODCPPM is represented by a classic three tiered configuration of presentation, logic, and storage encapsulated by an IIS web server, a Visual Basic processor within IIS, and a MS SQL Server relational database management system. All components are DoD (DADMS) and sponsor approved.

<provide input into the architecture this application will embrace/utilize>

## 2.2 Communications

Desktop applications will utilize HTTP with Secure Socket Layers (SSL) which leverages DoD PKI to perform certificate exchanges.

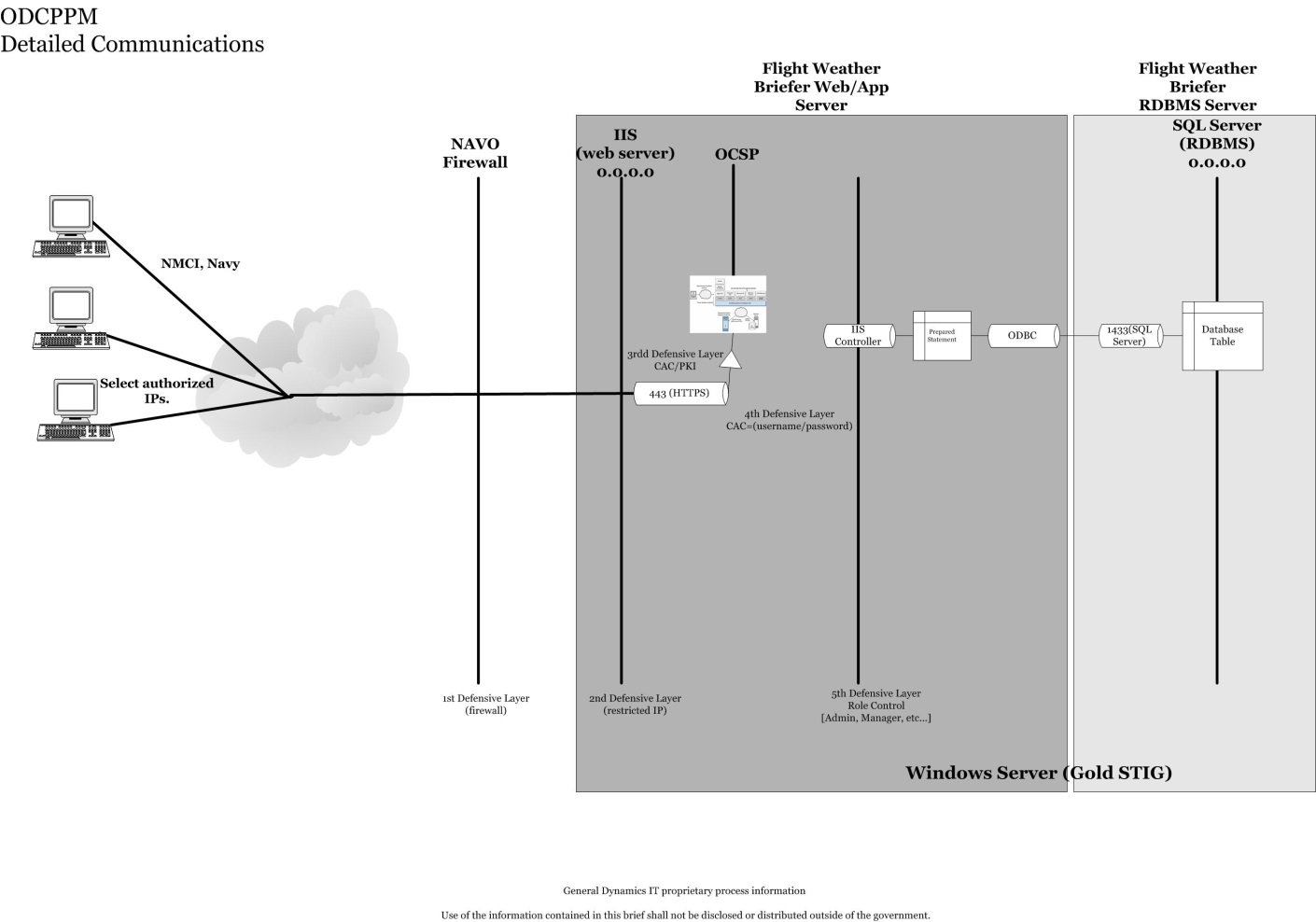


Figure - Communications

## 2.3 Roles

There are no roles defined for the smaller applications. PROGRAM 1 has specific roles which will be adhered to in accordance with PROGRAM 1 documentation.

|  |  |  |
| --- | --- | --- |
| Role | Function/Rights | Description |
| Administrator | admin | Effectively the super-user. |
| Maintenance | maint | Limited edit functionality, can view anything. |
| Browser | browser | Limited view. |

Table - Roles Definition

|  |  |  |  |
| --- | --- | --- | --- |
| Functionality | Administrator | Maintenance | Browser |
| Administration Menu Item | ● |  |  |
| Reports Menu Item | ● | ● |  |
| Create/Edit/Access/Approve Any Plan | ● | ● |  |

Table - Roles Definition

\*Only when content is owned or shared can a user in any role access subject content.

# 3.0 Requirements

Requirements are governed and managed by the Government. Typically an Enterprise Change Request (ECR) will be submitted requesting resources or authorization to execute against a requirement identified in the ECR. Most requirements executed by the DEVELOPER FST team will be based on an ECR and easily identified with an ECR number.

All requirements are tracked and managed by the government Point-of-Contact (POC) with a bi-weekly reporting tool. Work executed will reference the appropriate ECR when documenting any aspect of this project.

See the section entitled "Software Configuration Management (APSC-DV-003010)" for details regarding the proper way to start a change of this software application.

Requirements should be tracked with respect to change (of any type), status such as: new, tested, complete, and tracked by when the change was made and by whom. As previously mentioned the PMP will govern all requirements management.

# 4.0 Software Specifications

## 4.1 System Environment and System Components

The Java Virtual Machine (JVM) will be used to execute the applications developed for FST. Target environment are desktops which utilize the JVM. Unless noted in supplemental project documentation, software will execute on hardware compliant with IA-32 (32-bit) or x86-64 (64-bit) architecture most commonly provided by Intel (Intel 64), AMD (AMD64), and VIA (x64) chip manufacturers. Software will be targeted to x86-64 unless IA-32 support is specifically required.

Since Public Key Infrastructure (PKI) encryption is the primary mechanism for all web service interaction both a Common Access Card (CAC) and a CAC reader is required for both development and operational execution any software developed.

All aspects of Operating System security, maintenance, and installation are maintained by ??????; the Point-of-Contact is ???? at ????-????.

### 4.1.1 Managed Software Domain

See section, 1.4 Software, for details regarding the managed software domain associated with this IT solution.

### 4.1.2 <explicit software package name, like Tomcat or JBoss>

<Provide input into the function the software package provides in the execution environment, any specific rules or configuration required. Repeat as necessary.>

## 4.2 Development Technique

### 4.2.1 SDLC

The Software Development Lifecycle used for these applications will follow the <identify process used for software development> process which specifies how development will occur. This process adheres to the following lifecycle: Requirements, Design, Technical Design Review, Development, Test, Peer Review, and Deploy.

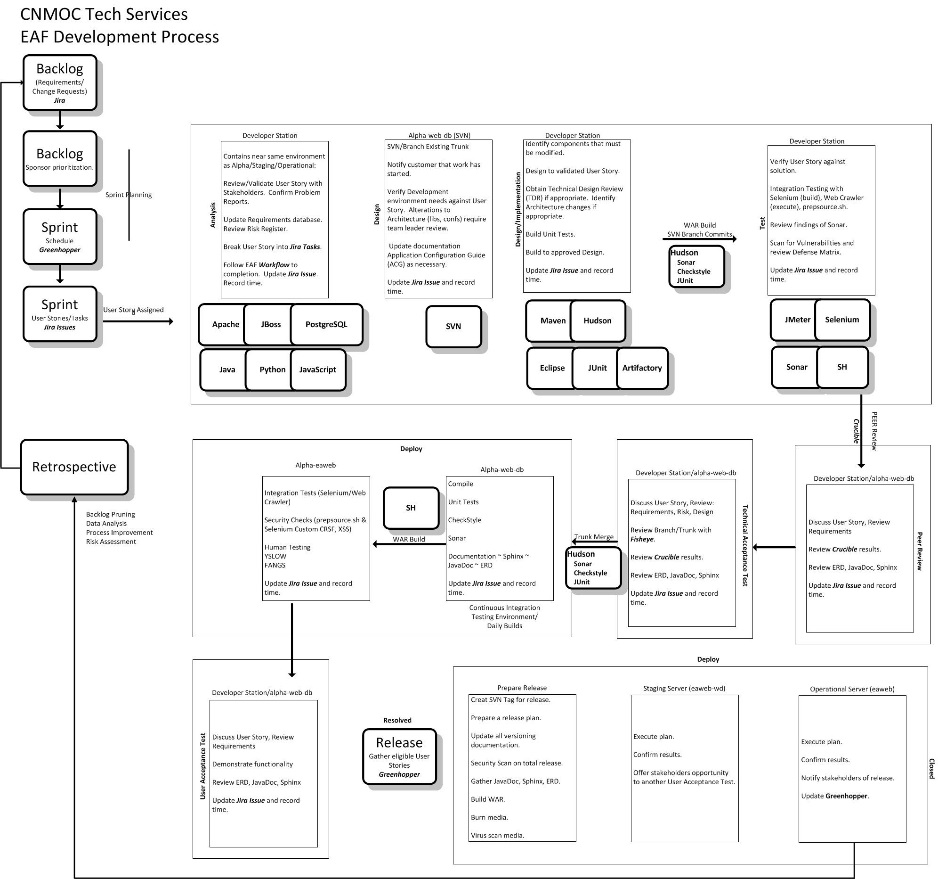


Table - SDLC Process Diagram

### 

### 4.2.2 Standards

Development efforts will follow the processes specified in the <PAL reference here> or other standard as specified in a kick-off meeting. <Enumerate the different processes and provide a hyperlink to their authoritative location>.

### 4.2.3 Process Asset Library (PAL)

The PAL contains all high level, corporate, desktop instructions, and localized versions of your processes.

Example: FST has a project folder which contains any specific desktop procedures or forms which will be utilized on the project. The Driftwood folder contains a Templates and Forms sub-folder which specifies how documents and deliveries will be structured. <Provide a reference/hyperlink to the PAL.>

### 4.2.4 Software Configuration Management (APSC-DV-003010)

Software Configuration Management (SCM) is the process of tracking code as it is developed, maintaining versions of code and their histories and ensuring that only one person at a time is making changes to a given section of code. The importance of SCM cannot be overstated, especially in a development environment with multiple programmers.

The applications will be modified as per the sponsor’s instruction. Each modification will be tracked as a task which in turn will be the basis for software modification, testing, and eventual feature release. All modifications will be correlated to the <host/parent organization tracking/requirements system> which authorizes the work.

Objects considered eligible for deployment:

* Source Code (Program files, HTML, JSP, etc…)
* Configuration/Settings files
* Libraries used for execution at build and run-time
* <add additional artifacts as required>

All source code developed is managed with a <explicit repository software name> repository that utilizes the <insert methodology used, example: classic Trunk|Branch|Tag> methodology. Updates and commentary applied to the repository will minimally reflect the tracking number and basic commentary regarding the change.

Additional information regarding operational deployment is located in the section entitled:

4.7 Transferring Code from Development to Deployment (APSC-DV-003010).

All modifications to any configurable part of the project will be managed and tracked by using the tracking system.

Each time this software is released a new version is indicated and a tag created, all change requests are recorded in the version history for that release. The team will have an environment that will represent the release of all software deployments. All software components, source code, libraries, and project documentation will be encapsulated within <repository name with URI> under specific structure requirements highlighted in the top level of the repository (APSC-DV-003010).

For a baseline of all components of the software package refer to the Deployment folder for the last deployment which should contain:

* Test report.
* Sonar Source report.
* Unit, Human Testing, Technical Acceptance Test | User Acceptance Test report.
* Security scan report (Sonar could duplicate this function).
* A copy of the actual software and any supplementary content to deploy as in the binary release.
* Reference to the <reference for repository> for subject release of the following components:
* Source Code.
* Documentation such as JavaDoc. As per the deployment guidance the following are a list of potential documents that must be created for deployments:
* Architecture diagram
* Design document
* User Release Notes
* Test Cases
* Application Configuration Guide
* Administrator’s Guide
* Deployment Guide
* <continue enumeration of deliverables>

For a delta of any two points in time code repositories can provide a detailed report. <name of code repository software> also provides a mechanism for controlled access to source and conflict resolution associated with multiple commits. A <name of repository software> Access Control List (ACL) is utilized to ensure only those team members that should have access can modify the repository.

This project has <enter classification of materials> components but is deployed to a classified network.

## 4.4 Build Environment

See Appendix E.1 Required Software for Development for an explanation of the tool used in the build environment.

## 4.5 Introduction Coding Standards

### 4.5.1 Definition

Common programming standards are adhered to for the following reasons:

* Programmers can go into any code and determine the current status which increases maintainability, readability, and reusability.
* New programmers can rapidly grasp the current status.
* Programmers that are unfamiliar with a programming language have a common standard which alleviates the need to develop a personal style.
* Standard programming language reduces errors and increases reliability.
* Individual styles are replaced and an emphasis is placed on business concerns - high productivity, maintainability, shared authorship, and other aspects that help the group achieve its goal.

See: <http://www.cse.buffalo.edu/~rapaport/code.documentation.Excerpts.pdf> for details and a more robust explanation as to why a coding standard is necessary.

### 4.5.2 Scope

The coding standards outlined in this document are applicable to all materials produced by DEVELOPER for the My Project (MyProj) 1.0 task order.

### 4.5.1 Why Coding Standards?

Coding standards will be utilized for the My Project (MyProj) 1.0 development effort to help ensure the most efficient means of introducing new developers can be achieved. By having a consistent mechanism for the presentation of solution logic we ensure a common ground by which any developer can work from. Establishing and adhering to a coding standard will help reduce the introduction of bugs at any insertion point of the software’s lifecycle.

### 4.5.2 Coding Standards Packages - Java

See the Sun Java Coding standard as the basis of the coding standard for this project.

<http://www.oracle.com/technetwork/java/codeconventions-150003.pdf> - it should be noted that this is a dated document and adherence will be maintained through IDE configuration. A script will be utilized to iterate through all source code files and apply the format template established by the team.

Use the Google Java Style guide and the following solution for formatting your source: <https://github.com/google/google-java-format>

#### 4.5.2.1 Coding Standards - Dates

ISO 8601 will be utilized for all date display, storage and interaction: <http://en.wikipedia.org/wiki/ISO_8601>

ISO 8601 is an international standard covering the exchange of date and time-related data. The standard is titled "Data elements and interchange formats ~ Information interchange ~ Representation of dates and times" and was issued by the International Organization for Standardization (ISO). The purpose of this International Standard is to eliminate the risk of misinterpretation where numeric representation of dates and times are interchanged across national boundaries, and to avoid the confusion and other consequential errors or losses. The standard organizes the data so the largest temporal term (the year) appears first in the data string and progresses to the smallest term (the second). It also provides for a standardized method of communicating time-based information across time zones by attaching an offset to Coordinated Universal Time (UTC).

Date and time expressed according to ISO 8601:

Date: 2010~02~19

Separate date and time in UTC: 2010~02~19 15:00Z

Combined date and time in UTC: 2010~02~19T15:00Z

Date with week number: 2010-W07-5

Ordinal date: 2010~050

#### 4.5.2.2 Coding Standards - NESI 3.1

* Make components independently deployable.
* Use a set of services to expose Component functionality.
* Access databases through open standard interfaces.
* Validate all input fields.
* Follow W3C standards for code which will generate a Web page display.
* Separate formatting from data through the use of style sheets instead of hard coded HTML attributes.
* Comply with Federal accessibility standards contained in Section 508 of the Rehabilitation Act of 1973 (as amended) when developing software user interfaces.
* Encapsulate Java code that is used in JSP(s) in tag libraries.
* Separate application, presentation, and data tiers.
* Use a build tool.
* Use a build tool that supports operation in an automated mode.
* Use a build tool that checks out files from configuration control.
* Use a build tool that compiles source code and dependencies that have been modified.
* Use a build tool that creates libraries or archives after all required compilations are completed.
* Use a build tool that creates executables.
* Use a build tool that is capable of running unit tests.
* Use a build tool that cleans out intermediate files that can be regenerated.
* Use a build tool that is independent of the Integrated Development Environment.
* Do not hard-code the configuration data of a Web service vendor.

#### 4.5.2.3 Coding Standards – YSLOW

* Minimize HTTP Requests ~ combine files
* Use a Content Delivery Network ~ distributed CPU geographically
* Add an Expires or a Cache-Control Header ~ have client cache reused (img, jscript) content
* Gzip Components ~ compress content (mod\_deflate)
* Stylesheets at the Top
* Scripts at the bottom ~ blocks parallel downloads, either push to bottom or call DEFER attribute
* Avoid CSS Expressions ~ calculated values on the fly
* Make JavaScript and CSS External
* Reduce DNS Lookups ~ reduce unique hostnames referenced
* Minify CSS and JavaScript
* Avoid Redirects ~ specifically for initial page starts
* Remove duplicate scripts ~ repetitive JS
* Configure Etags ~ entity tags, more guaranteed mechanism for confirmation
* Make Ajax Cacheable
* Flush the buffer early
* Use GET for Ajax Requests
* Post-load Components
* Pre-load Components
* Reduce the Number of DOM Elements
* Split Components Across Domains
* Minimize the Number of iframes
* No 404s
* Reduce Cookie Size
* Use Cookie-free Domains for Components
* Minimize DOM Access
* Develop Smart Event Handlers
* Choose <link> over @import
* Avoid Filters ~ example: png fix for IE
* Optimize Images
* Optimize CSS Sprites
* Don't Scale Images in HTML
* Make favicon.ico Small and Cacheable
* Keep Components under 25K
* Pack Components into a Multipart Document
* Avoid Empty Image Src

#### 4.5.2.4 General Conventions - Java

Unsafe Methods (APSC-DV-003210)

Unsafe methods are those that have side-effects that are not understood properly or require elevated security privileges in order to perform their function.

##### System Executables

Using command line exe's are considered inherently unsafe, as such a SHA256 hash of the executable will be gathered, stored in a configuration file and referenced prior to any call/execution of it. Algorithm should be:

* Perform SHA256 comparison of executable about to be called.
* All input gathered from user, in other words, is not statically defined must be scrubbed.
* If executable is SHA256 tracked and verified then the System.exec can be called.

##### No URL's in Compiled Code

URL's on pages, specifically URL’s which are presented on page, will not be statically defined in page thus requiring recompilation. Exceptions to this rule are if the language is scripted and does not require recompile (JSP, ASP, HTML, etc...). The following algorithm will be used:

All URLs will be registered with a standard XML file.

* Each segment of the site (think different package schema) will be separated within the XML file.
* A SHA256 of the XML configuration file will be stored in a configuration file.
* Upload load of the site read in a singleton after confirming the SHA256 signature.
* Use a structure to persistent package/url information and call as necessary.
* Failure of any type will result in no URL being displayed on page.

##### Exceptions/Error Messages

Error messages (Exceptions) will be invoked with a custom class (API infrastructure) specifically designed to support the Application Development STIG to ensure no information expressed during error (stack traces) are displayed to the end-user thus revealing information that could be deemed valuable to an attacker. Output of error should go to the log, based on log levels, and an obfuscated/reduced output should be displayed to the end user.

The software specific error/exception class will extend from the base functionality of the existing language error/exception class/object.

A domain of standard responses, which are application specific, will be matured over time based on encounters of the running package.

##### Best Practices

###### Local Variables

* Do not name local variables with the same name as an instance (global scope) variable.
* One per line.
* Use the local variable for only one thing.

###### Methods

* Separate with blank lines.
* Refactor where appropriate. If you have a method that takes more than 60 seconds to understand, break it into helper methods.
* Reuse. If you have a routine that is called by someone/something else consider incorporation into a larger schema for access such as our APIs.

###### Imports/Library references

* Use separate lines.
* Preference is to group as follows: 1) standard library 2) related third party 3) local project
* Bulk import for #1, #2, and #3 bring in specific objects needed.
* Regardless, be consistent.
* Remove unused import statements; leverage the IDE to find these if possible.

###### Error Handling

* All code will be written to degrade gracefully...if possible and where appropriate.
* All routines that execute a portion of the web page must be wrapped in a try/catch/finally.
* All routines that involve IO or external program calls must be wrapped in a try/catch/finally.
* All routines that the developer considers "risky" meaning has potential for failure or the developer recognized potential flaw in algorithm and is unsure of a solution will wrap the code in a try/catch/finally.
* All results of failure will gracefully degrade and report the error along with meaningful commentary.
* An entire page should not fail if a single component has a problem.

###### Abstraction

Always abstract one level between API's and invocation. If there's a class that provides Servlet functionality then the code developed will be abstracted at one level to inherit from Servlet and all use of Servlet would be that extended class. This concept applies to any classes that will be used repeatedly or involve global level utilization.

###### Source Formatting

All source, for the entire project, will be formatted prior to a tag being created using, preferred, an external formatting tool thus ensuring no dependency on a specific IDE.

###### Static Code Analysis (SonarQube)

Source code will be scanned for classic errors using a standardized tool (could be considered standard if a custom written solution is the only way to verify error for a language).

* No issues flagged as Critical will be permitted to go forward into a release unless agreed to by team.
* Update tool configuration file so error doesn't propagate.
* CM configuration files for tool.
* Add to comments/readme in deployment document and design document (coding standards)

##### Top 10 Secure Coding Practices (Software Engineering Institute)

1. Validate input Validate input from all untrusted data sources. Proper input validation can eliminate the vast majority of software vulnerabilities. Be suspicious of most external data sources, including command line arguments, network interfaces, environmental variables, and user controlled.

2. Heed compiler warnings. Compile code using the highest warning level available for your compiler and eliminate warnings by modifying the code. Use static and dynamic analysis tools to detect and eliminate additional security flaws.

3. Architect and design for security policies. Create software architecture and design your software to implement and enforce security policies. For example, if your system requires different privileges at different times, consider dividing the system into distinct intercommunicating subsystems, each with an appropriate privilege set.

4. Keep it simple. Keep the design as simple and small as possible. Complex designs increase the likelihood that errors will be made in their implementation, configuration, and use. Additionally, the effort required to achieve an appropriate level of assurance increases dramatically as security mechanisms become more complex.

5. Default deny. Base access decisions on permission rather than exclusion. This means that, by default, access is denied and the protection scheme identifies conditions under which access is permitted.

6. Adhere to the principle of least privilege. Every process should execute with the least set of privileges necessary to complete the job. Any elevated permission should be held for a minimum time. This approach reduces the opportunities an attacker has to execute arbitrary code with elevated privileges.

7. Sanitize data sent to other systems. Sanitize all data passed to complex subsystems such as command shells, relational databases, and commercial off-the-shelf components. Attackers may be able to invoke unused functionality in these components through the use of SQL, command, or other injection attacks. This is not necessarily an input validation problem because the complex subsystem being invoked does not understand the context in which the call is made. Because the calling process understands the context, it is responsible for sanitizing the data before invoking the subsystem.

8. Practice defense in depth. Manage risk with multiple defensive strategies, so that if one layer of defense turns out to be inadequate, another layer of defense can prevent a security flaw from becoming an exploitable vulnerability and/or limit the consequences of a successful exploit. For example, combining secure programming techniques with secure runtime environments should reduce the likelihood that vulnerabilities remaining in the code at deployment time can be exploited in the operational environment.

9. Use effective quality assurance techniques. Good quality assurance techniques can be effective in identifying and eliminating vulnerabilities. Fuzz testing, penetration testing, and source code audits should all be incorporated as part of an effective quality assurance program. Independent security reviews can lead to more secure systems. External reviewers bring an independent perspective; for example, in identifying and correcting invalid assumptions.

10. Adopt a secure coding standard. Develop and/or apply a secure coding standard for your target development language and platform.

#### 4.5.2.4 Documentation Conventions – Java Source Code

##### General Rules

* Comments are necessary. However not all comments are necessary.
* Doc comments should not be all inclusive. A summary of the purpose of the declaration is a must.
* Multiple paragraphs explaining every step of the implementation of the method is NOT required, or even wanted!
* Don't put implementation details in the comments. Leave implementation in the code which should read like a story.
* Never assume someone else will know what you are thinking. Don't be familiar.
* Document arguments in a complete manner. If unclear use an example.

##### Single Line Comments

* Are used within methods to document business logic, code sections, or declarations of temporary variables.
* Start with "'".

##### Comment Header Block

Each major routine should have a header that identifies:

* who wrote it
* what it is supposed to do
* what the parameters mean (both input and output)
* what it returns (if it's a function)
* what assumptions it makes about the state of the program or environment
* any known limitations
* an amendment history

#### 4.5.2.5 References

|  |  |
| --- | --- |
| Subject | URL |
| Java Coding Standards (September 12, 1997) | <http://www.oracle.com/technetwork/java/codeconvtoc-136057.html> |

## 4.6 Reusable Objects

All member variables of a class that are no longer required after object utilization should be set to null in the finally portion of the object as appropriate.

No information, including encrypted representations of information, produced by prior actions is available to any subsequent use of the object. There should be no residual data from the former object.

Of concern would be any database representations, as objects, that contain residual data after load, therefore clearing/nullifying database objects is considered essential.

### 4.6.1 Modules

Where appropriate Java package structure will be created (modules) to support specific functionality, Object Oriented Programming principles will be followed whereby objects communicate with one another vice straight procedure programming techniques.

### 4.6.2 Mobile Code

As per Assistant Security of Defense, March 14, 2011 Policy Guidance for use of Mobile Code Technologies in Department of Defense (DoD) Information Systems. By definition, mobile code is software obtained from remote systems outside the enclave boundary, transferred across a network, and then downloaded and executed on a local system without explicit installation or execution by the recipient.

#### 4.6.2.1 Categories

Category 1 - Great thread, broad functionality with unmediated access to DoD IT systems. Can pose a significant threat to DoD information systems.

* + 1A - Can be signed, is allowed, used only when the mobile code is signed with a DoD-approved PKI code signing certificate and the mobile code is obtained from a trusted source. Examples include: ActiveX controls, mobile code scripts that execute in Windows Scripting Host (WSH).
  + 1X - Cannot be signed, is **not allowed**. Examples include MS-DOS batch scripts, Unix shell scripts, Shockwave movies that execute in the Shockwave for Director plugin.

Category 2 – Moderate thread, limited capability that can be controlled.

* + May be used in DoD information systems if the mobile code is obtained from a trusted source over a assured channel.  In addition, unsigned Category 2 mobile code, whether or not obtained from a trusted source or an assured channel, may be used if it executes in a constrained environment without access to the local system and network resources (other than its originating host).
  + Mention of a mobile code risk strategy as part of the risk management plan should be included.
  + The Application Security and Development STIG V3R10 highly leans towards signing Category 2 mobile code.

Category 3 – Limited functionality with no capability for unmediated access to DoD IT systems.

* + May be used in DoD information systems.  No additional requirements.

Technology cited as being outside the scope of the DoD Mobile Code Policy DoDI8552.01 are scripts and applets that execute in the context of the web server. Examples of technologies in this application area are Java servlets, Java Server Pages, Common Gateway Interface (CGI) applications, Active Server Pages (ASP), etc. Distributed object-oriented programming systems that do not convey executable objects such as: SOAP, CORBA, and DCOM.

#### 4.6.2.1 My Project Mobile Code Categorization

|  |  |  |
| --- | --- | --- |
| Code | Category | Description |
| JavaScript | 3 | JavaScript used in client browser to perform in browser functions related to the application. |

Table - Mobile Code

No additional requirements for the code in this application are required with respect to Mobile Code concerns.

### 4.6.3 References

|  |  |
| --- | --- |
| Subject | Reference |
| Data Validation | <http://en.wikipedia.org/wiki/Data_validation> |
| DEVELOPER BSL Mobile Code Documentation | [http://confluence.developer.bsl.lab/pages/viewpage.action?pageId=11600094](http://confluence.gdit.bsl.lab/pages/viewpage.action?pageId=11600094) |
| Department of Defense Instruction 8550.01 | <https://powhatan.iiie.disa.mil/mcp/mobile-code-memo-2011Mar14.pdf> |

## 4.7 Transferring Code from Development to Deployment (APSC-DV-003010)

As referenced in section 4.4 Build Environment, multiple environments are used to develop this application. Build and Test (BAT) is the actual developer's workstation. Code is written after the trunk is branched and the appropriate change request is updated. The developer implements the algorithm/modifications to their branch on their workstation and conducts tests prior to reintegration of the branch into the trunk.

The Operational environment is <name of environment> and is maintained by <maintainer’s name> representatives.

### 4.7.1 Deployment

The final steps in any deployment are to specifically test both general functionality as well as specific functionality of what was released with the specific stakeholder for each change request.

<provide details regarding deployment requirements, references, and techniques>  
 The diagram below is a workflow that specifies the <name of organization> deployment path:

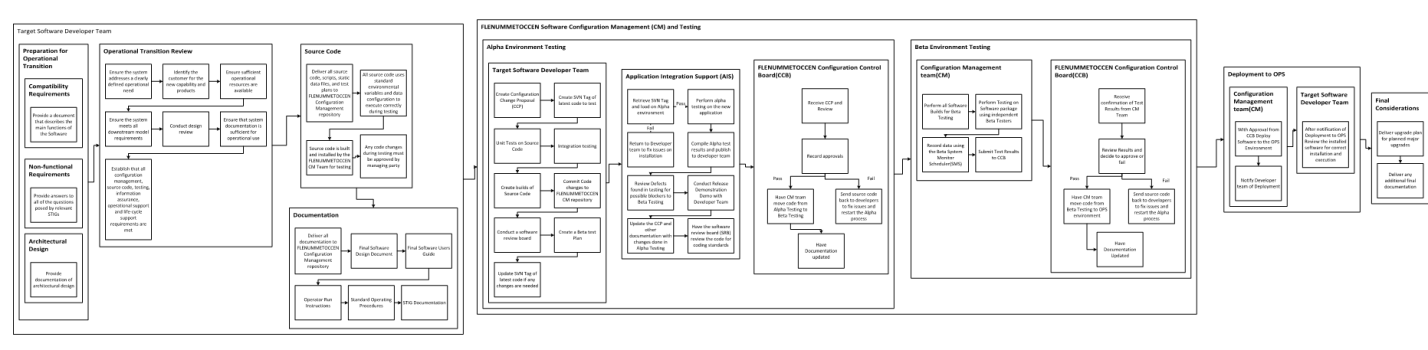


Figure - Deployment Path

<Reference Organization specific deployment requirements here. >

#### 4.7.1.1 Creation of Deployment Materials

<indicate where deployment materials are kept as part of a deployment, if using a code repository be sure to include the URI.>

# 5.0 Security

## 5.1 Definition

The focus of this section is to discuss security within a DoD environment for web application as well as Desktop application development. Security in this context are those measures taken to protect an application from aberrant/malicious actions by any user accessing the application. Security is the use of various mechanisms, with defense in depth taken into consideration, intended to thwart and/or record the aforementioned aberrant/malicious user behavior.

## 5.2 Scope

This document represents core knowledge of the project in question regarding application security which will serve as the authoritative medium for all decisions. Standards, rules, and guidance on action will come from the material contained in this document.

The goal of application security is to prevent unauthorized access to the application and its respective data. It should be noted that there is no completely secure application; however attack vectors can be secured such that the efforts to overcome security implementations far outweigh any gains that could be had.

The SANS Institute has an excellent and sufficiently generic checklist regarding application security. This checklist notes important points to consider during the design process: risk assessment, authentication, and authorization/access control. Security issues within the actual codebase fall outside the scope of this document.

The Department of Defense (DoD) is sufficiently concerned about this topic to dedicate a complete Security Technical Implementation Guide (STIG) to the subject. Information Assurance Officers (IAO) and even end users should also be concerned with web application security because of its direct effect on data integrity.

Major categories within application security discussed in this text are:

* Security Technical Implementation Guide (STIG)
* Application Security and Development (AppDev), Version 4, Release 3 (27 January 2017)
* Authentication
* Desktop applications will not utilize authentication and will rely solely on Windows credentials/rights to execute.
* Common Access Card (CAC) where necessary.
* Open Web Application Security Project (OWASP) https://www.owasp.org/index.php/Main\_Page
* National Institute of Standards and Technology (NIST) - http://www.nist.gov/

## 5.3 Categories

### 5.3.1 Security Technical Implementation Guide (STIG)

From the STIG:

DoD Directive 8500.01E requires that "all IA and IA-enabled IT products incorporated into DoD information systems shall be configured in accordance with DoD-approved security configuration guidelines" and tasks DISA to "develop and provide security configuration guidance for IA and IA-enabled IT products in coordination with Director, NSA".

By using the STIG, developers achieve a twofold accomplishment: compliance with DoD mandates, and usage of fully vetted best practices. The STIG addresses three levels (or categories) of vulnerabilities -- CAT I (highest, can lead to immediate unauthorized access), CAT II (high potential of granting unauthorized access), and CAT III (can indirectly cause unauthorized access).

A vulnerabilities document related to each application will be created, for example the FST-BTPortlet has a STIG review document that will be maintained with the code base.

### 5.3.2 Authentication

DoD PKI credential authentication has been mandated for all web applications residing on DoD Information Systems (IS) where user authentication is necessary. The DoD CAC card uses DoD PKI credentials and should be the primary form of authentication for any web applications developed for this effort.

If a desktop application connects to a DoD web service the regular rules associated with that server will apply. Desktop applications will not utilize an authentication schema outside of the host operating system unless otherwise noted in this section.

Any Java web services or Rich Internet Applications (RIA’s) running on NMCI will have to be signed by a Certificate with in the Certificate Authority (CA) as the JRE will be locked down and not allow execution of subject Java based technology.

Java Archive Resources (JAR) executions will not be affected by Java signing.

### 5.3.3 Unique Security Requirements

<specify any unique security requirement associated with this application. For example, if a web application explain that two-way SSL using TLS > 1.0 is in use and that the encryption certificates are part of the Department of Defense (DoD) Public Key Infrastructure (PKI).>

### 5.3.4 Open Web Application Security Project (OWASP)

OWASP advocates the following security principles:

* Minimize Attack Surface Area
* Secure Defaults
* Principle of Least Privilege
* Principle of Defense in Depth
* Fail Securely
* External Systems are Insecure
* Separation of Duties
* Do not trust Security through Obscurity
* Simplicity
* Fix Security Issues Correctly

### 5.3.5 Common Weakness Enumeration (CWE)

From the website:

"International in scope and free for public use, CWE(TM) provides an unified, measurable set of software weaknesses that is enabling more effective discussion, description, selection, and use of software security tools and services that can find these weaknesses in source code and operational systems as well as better understanding and management of software weaknesses related to architecture and design." The Common Weakness Enumeration website can certainly be consulted periodically concerning vulnerabilities and weaknesses or as issues arise, otherwise this website should not be considered a must-visit by developers.

It should be noted that valuable information related to application security can be obtained from this site. While not mandatory, as DoD STIG requirements represent the highest standard, this site is worth periodic review for potential vulnerabilities when executing a threat assessment.

### 5.3.5 National Institute of Standards and Technology (NIST)

From the website:

"Founded in 1901, NIST is a non-regulatory federal agency within the U.S. Department of Commerce. NIST's mission is to promote U.S. innovation and industrial competitiveness by advancing measurement science, standards, and technology in ways that enhance economic security and improve our quality of life."

It should be noted that valuable information related to application security can be obtained from this site. While not mandatory, as DoD STIG requirements represent the highest standard, this site is worth periodic review for potential vulnerabilities when executing a threat assessment.

## 5.4 References

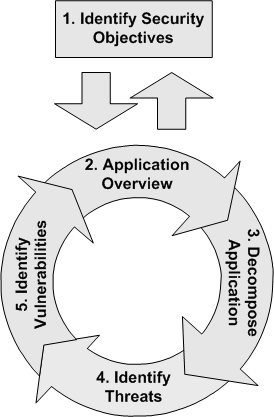
|  |  |
| --- | --- |
| Subject | Reference/Document |
| Security Technical Implementation Guide and Application Security and Development (STIG) | <http://iase.disa.mil/stigs/Pages/index.aspx> |
| SANS Security Checklist | <http://www.sans.org/reading_room/whitepapers/securecode/a_secuity_checklist_for_web_application_design_1389/show=1389.php&cat=securecode> |
| Open Web Application Security Project | <http://www.owasp.org/index.php/Main_Page> |
| A Guide to Building Secure Web Applications and Web Services 2.0 Black Hat Edition | <http://iweb.dl.sourceforge.net/project/owasp/Guide/2.0.1/OWASPGuide2.0.1.pdf> |
| Web Services Security Standard v1.1 | <http://www.oasis-open.org/specs/index.php#wssv1.1> |
| Common Weakness Enumeration | [http://cwe.mitre.org](http://cwe.mitre.org/) |
| National Institute of Standards and Technology | <http://ts.nist.gov/> |

## 5.6 Threat Assessment (APSC-DV-003230)

The STIG details Threat Modeling:

"Threat Modeling is the process of identifying potential threats to the application, risk ranking these threats, and selecting appropriate countermeasures or mitigations for the threats. Threat modeling is a critical step in securing an application from attack."

* Define Common Usage
* Identify External Dependencies
* Enumerate Security Assumptions
* Identify Objects and Interactions
* Identify Entry Points
* Determine Threat Risk
* Identify Potential Mitigations

[](http://www.google.com/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&docid=OAkNr2qVzOqS2M&tbnid=UwqiZikMjndXFM:&ved=0CAUQjRw&url=http://msdn.microsoft.com/en-us/library/ff648006.aspx&ei=cULJU8byGen48QGR7oDgDA&bvm=bv.71198958,d.aWw&psig=AFQjCNH_ACjLYpn2v0RG_mecyk__-ixOsg&ust=1405785067407519)The threat risk modeling process has five steps, enumerated below and shown graphically below. They are:

* Identify Security Objectives.
* Survey the Application
* Decompose it
* Identify Threats
* Identify Vulnerabilities

The remainder of this section identifies different threat assessment models and how they might be used to assist in an assessment. Refer to [Appendix C: THREAT ASSESSMENT](#_C.1_Dread_Implementation) for the actual assessment for My Project (MyProj) v1.0.

### 5.6.1 DREAD

DREAD is a classification scheme for quantifying, comparing and prioritizing the amount of risk presented by each evaluated threat. The DREAD acronym is formed from the first letter of each category below. DREAD modeling influences the thinking behind setting the risk rating, and is also used directly to sort the risks. The

DREAD algorithm, shown below, is used to compute a risk value, which is an average of all five categories.

Risk\_DREAD = (DAMAGE + REPRODUCIBILITY + EXPLOITABILITY + AFFECTED USERS + DISCOVERABILITY) / 5

The calculation always produces a number between 0 and 10; the higher the number, the more serious the risk. Here are some examples of how to quantify the DREAD categories:

#### Damage Potential

If a threat exploit occurs, how much damage will be caused?

* 0 = Nothing
* 5 = Individual user data is compromised or affected.
* 10 = Complete system or data destruction

#### Reproducibility

How easy is it to reproduce the threat exploit?

* 0 = Very hard or impossible, even for administrators of the application.
* 5 = One or two steps required, may need to be an authorized user.
* 10 = Just a web browser and the address bar is sufficient, without authentication.

#### Exploitability

What is needed to exploit this threat?

* 0 = Advanced programming and networking knowledge, with custom or advanced attack tools.
* 5 = Malware exists on the Internet, or an exploit is easily performed, using available attack tools.
* 10 = Just a web browser

#### Affected Users

How many users will be affected?

* 0 = None
* 5 = Some users, but not all
* 10 = All users

#### Discoverability

How easy is it to discover this threat?

* 0 = Very hard to impossible; requires source code or administrative access.
* 5 = Can figure it out by guessing or by monitoring network traces.
* 9 = Details of faults like this are already in the public domain and can be easily discovered using a search engine.
* 10 = The information is visible in the web browser address bar or in a form.

\*\*Note:\*\* When performing a security review of an existing application, "Discoverability" will often be set to 10 by convention, as it is assumed the threat issues will be discovered.

### 5.6.3 STRIDE

STRIDE is a classification scheme for characterizing known threats according to the kinds of exploit that are used (or motivation of the attacker). The STRIDE acronym is formed from the first letter of each of the following categories.

#### Spoofing Identity

“Identity spoofing” is a key risk for applications that have many users but provide a single execution context at the application and database level. In particular, users should not be able to become any other user or assume the attributes of another user.

#### Tampering with Data

Users can potentially change data delivered to them, return it, and thereby potentially manipulate client-side validation, GET and POST results, cookies, HTTP headers, and so forth. The application should not send data to the user, such as interest rates or periods, which are obtainable only from within the application itself. The application should also carefully check data received from the user and validate that it is sane and applicable before storing or using it.

#### Repudiation

Users may dispute transactions if there is insufficient auditing or recordkeeping of their activity. For example, if a user says, “But I didn’t transfer any money to this external account!”, and you cannot track his/her activities through the application, then it is extremely likely that the transaction will have to be written off as a loss. Therefore, consider if the application requires non-repudiation controls, such as web access logs, audit trails at each tier, or the same user context from top to bottom. Preferably, the application should run with the user’s privileges, not more, but this may not be possible with many off-the-shelf application frameworks.

#### Information Disclosure

Users are rightfully wary of submitting private details to a system. If it is possible for an attacker to publicly reveal user data at large, whether anonymously or as an authorized user, there will be an immediate loss of confidence and a substantial period of reputation loss. Therefore, applications must include strong controls to prevent user ID tampering and abuse, particularly if they use a single context to run the entire application. Also, consider if the user’s web browser may leak information. Some web browsers may ignore the no caching directives in HTTP headers or handle them incorrectly. In a corresponding fashion, every secure application has a responsibility to minimize the amount of information stored by the web browser, just in case it leaks or leaves information behind, which can be used by an attacker to learn details about the application, the user, or to potentially become that user. Finally, in implementing persistent values, keep in mind that the use of hidden fields is insecure by nature. Such storage should not be relied on to secure sensitive information or to provide adequate personal privacy safeguards.

#### Denial of Service

Application designers should be aware that their applications may be subject to a denial of service attack. Therefore, the use of expensive resources such as large files, complex calculations, heavy-duty searches, or long queries should be reserved for authenticated and authorized users, and not available to anonymous users.

For applications that do not have this luxury, every facet of the application should be engineered to perform as little work as possible, to use fast and few database queries, to avoid exposing large files or unique links per user, in order to prevent simple denial of service attacks.

#### Elevation of Privilege

If an application provides distinct user and administrative roles, then it is vital to ensure that the user cannot elevate his/her role to a higher privilege one. In particular, simply not displaying privileged role links is insufficient. Instead, all actions should be gated through an authorization matrix, to ensure that only the permitted roles can access privileged functionality.

## 5.7 Security Tools (APSC-DV-001460)

SonarQube will be used as a static code analyzer; no automated test tools have been identified save direct user manipulation for the desktop application(s). If web development occurs additional tools will be selected.

Fuzz Scan will be performed with (MyProj) v on the execution environment.

## 5.8 Application Configuration Guide ~ ACG (APSC-DV-003285)

The (MyProj) v. will be designated as MAC III with For Official Use Only (FOUO) data. An application configuration guide defines the components, configuration of the components, and how those components work together. See Appendix A: Application Configuration Guide (ACG) ~ APSC-DV-003285. Installation guides are located in a separate text.

## 5.9 References

|  |  |
| --- | --- |
| Name | Reference |
| OWASP Application Threat Modeling | <https://www.owasp.org/index.php/Application_Threat_Modeling> |
| Microsoft Web Application Security Framework (Cheat Sheet) | <https://msdn.microsoft.com/en-US/library/ms978518.aspx> |
| Microsoft Template Sample for Web Application Threat Model | <https://msdn.microsoft.com/en-US/library/ms978534.aspx> |
| Microsoft Walkthrough: Creating a Threat Model for a Web Application | <https://msdn.microsoft.com/en-US/library/ms978538.aspx> |
| Microsoft How To: Create a Threat Model for a Web Application at Design Time | <https://msdn.microsoft.com/en-US/library/ms978527.aspx> |
| Microsoft Threat Modeling Web Applications | <https://msdn.microsoft.com/library/ms978516.aspx> |
|  |  |

# 6.0 Web Site Design

The following sections layout the methods for future FST web application development. This section is placed to support web application, if developed, in the future. Understanding the web components and proper layout are essential towards making a user-friendly, engaging site. It should be noted that these guidelines represent a potential standard to follow but are not representative of the current implementation which utilizes Microsoft default web page layout.

## 6.1 Web Design Standards and Guidelines

To design a user friendly web-application it necessary to lay down guidelines that insure both an intuitive interface and a cohesive presentation throughout the application. This is accomplished by defining guidelines that apply to structural elements within the site and applying those guidelines throughout the site. The purpose of this paper will be to define those guidelines and aid developers in implementing them throughout the application.

## 6.2 Part I - Design Standards

The web-application interface will be presented graphically by a web-browser's interpretation of a combination of the following elements:

* Standard HTML 5.0
* CSS version 2 , preference is to use BootStrap
* Images conforming to the JPEG standard.
* PNG- Portable Network Graphics
* Images conforming to the GIF standard. Graphic Interchange Format

### 6.2.1 Standard HTML 5.0

HTML5 is a markup language used for structuring and presenting content on the World Wide Web. It is the fifth and current version of the HTML standard. HTML5 includes detailed processing models to encourage more interoperable implementations; it extends, improves and rationalizes the markup available for documents, and introduces markup and application programming interfaces (APIs) for complex web applications.[6] For the same reasons, HTML5 is also a candidate for cross-platform mobile applications, because it includes features designed with low-powered devices in mind.

HTML 5 has ties to semantic elements, specifically constructed form elements, and graphic/multi-media elements. Typical concerns over browser compliance apply if considering use of HTML 5 custom elements.

Many new syntactic features are included. To natively include and handle multimedia and graphical content, the new <video>, <audio> and <canvas> elements were added, and support for scalable vector graphics (SVG) content and MathML for mathematical formulas. To enrich the semantic content of documents, new page structure elements such as <main>, <section>, <article>, <header>, <footer>, <aside>, <nav> and <figure>, are added. New attributes are introduced, some elements and attributes have been removed, and others such as <a>, <cite> and <menu> have been changed, redefined or standardized. The APIs and Document Object Model (DOM) are now fundamental parts of the HTML5 specification and HTML5 also better defines the processing for any invalid documents.

### 6.2.2 CSS version 2

CSS version 2 refers to the second version of the cascading style sheet standard approved by the W3C (World Wide Web Consortium). CSS provides a way to segregate repetitive display information from the data that it is being applied to. CSS will be used in FST to provide the web browser with information about where specific data will be displayed on a page and how it will look. The CSS used in the development of FST web applications will conform to the CSS version 2 standards and will be validated thru W3C Validation Service found at the following web address: <http://jigsaw.w3.org/css-validator/>.

## 6.3 References

|  |  |
| --- | --- |
| Subject | References |
| Webmaster | <http://en.wikipedia.org/wiki/Webmaster> |
| Digital Images | <http://www.wfu.edu/~matthews/misc/graphics/formats/formats.html> |
| Cascading Style Sheet | <http://en.wikipedia.org/wiki/.css> |
| Base 64 Encoding | <http://email.about.com/cs/standards/a/base64_encoding.htm> |
| Section 508 | <http://www.justice.gov/crt/508/508law.php> |
| More Section 508 | [http://www.section508.gov](http://www.section508.gov/) |
| CSS Software | <http://code.google.com/p/minify/wiki/UserGuide> |
| CSS | <http://friendlybit.com/css/how-to-structure-large-css-files> |
| HTML 5 | <https://en.wikipedia.org/wiki/HTML5>  <http://www.w3.org/TR/html5/> |

# 7.0 Metrics

## 7.1 Definition

Software metric is a measure of some property of a piece of software or its specifications. Since quantitative measurements are essential in all sciences, there is a continuous effort by computer science practitioners and theoreticians to bring similar approaches to software development. The goal is obtaining objective, reproducible and quantifiable measurements, which may have numerous valuable applications in schedule and budget planning, cost estimation, quality assurance testing, software debugging, software performance optimization, and optimal personnel task assignments.

## 7.2 Scope

The scope of the Metrics Topic will cover the future development effort such as lines of code and tracking elements (time). Metrics, for both process and software, tell us to what extent a desired characteristic is present in our processes or our software systems. Maintainability is a desired characteristic of a software component and is referenced in all the main software quality models (including the ISO 9126). Measures will be taken at each software release after the initial deployment.

## 7.3 Categories

* Issues
* Priority overall/per Cycle
* Level of effort overall per/Cycle
* Open/Closed overall/per Cycle/per release
* Lines of Code

Obtained through Sonar or related metrics code tool.

### 7.3.2 Issues (during Retrospective or Post-Mortem function)

* Priority overall/per Cycle
* All issues will be queried per Cycle
* Level of effort overall per/month
* All issues will be selected by level of effort overall and per Cycle
* Open/Closed overall/per Cycle/per release
* All issues Open & Closed by level per Cycle and overall will be maintained.

### 7.3.3 Lines of Code

* Length of source code (measured by lines of code).

Lines of code will be counted with a standard metrics tool like SonarQube or other DADMS compliant tool.

## 7.4 References

|  |  |
| --- | --- |
| Subject | Reference/Document |
| 10th International Software Metrics Symposium (Metrics 2004) | <http://www.kaner.com/pdfs/metrics2004.pdf> |
| Software Metrics | <http://www.sqa.net/softwarequalitymetrics.html> |

# 8.0 Testing

## 8.1 Definition

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing also provides an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include, but are not limited to, the process of executing a program or application with the intent of finding software bugs (errors).

Software testing can also be stated as the process of validating and verifying that a software program/application/product:

* Meets the business and technical requirements that guided its design and development;
* Works as expected; and
* Can be implemented with the same characteristics.

Software testing, depending on the testing method employed, can be implemented at any time in the development process. However, most of the test effort occurs after the requirements have been defined and the coding process has been completed. As such, the methodology of the test is governed by the software development methodology adopted.

Different software development models will focus the test effort at different points in the development process. Newer development models, such as Agile, often employ test driven development and place an increased portion of the testing in the hands of the developer, before it reaches a formal team of testers. In a more traditional model, most of the test execution occurs after the requirements have been defined and the coding process has been completed.

## 8.2 Scope

A primary purpose of testing is to detect software failures so that defects may be discovered and corrected. FST testing will be conducted for smaller application such as: FSTBulkUploader, DateChanger. All testing for release will have a Test Plan prior to release to the customer. User Acceptance Test (UAT) will be conducted prior to operational release of the software.

## 8.3 Categories

### 8.3.1 Functional vs. Non-functional Testing

Functional testing refers to activities that verify a specific action or function of the code. These are usually found in the code requirements documentation, although some development methodologies work from use cases or user stories. Functional tests tend to answer the question of "can the user do this" or "does this particular feature work".

Non-functional testing refers to aspects of the software that may not be related to a specific function or user action, such as scalability or security. Non-functional testing tends to answer such questions as "how many people can log in at once".

Most applications developed will be desktop applications and many traditional non-functional tests such as mutli-user access, band-width utilization will not be applicable. Specific tests will be documented in Use Test Cases. Only those tests the customer requests of a non-functional nature will actually have registered tests in Use Test Cases.

### 8.3.2 Defects and Failures

All defects and failures encountered will be registered with the sponsor and designated as a “bug”. All defects will be addressed and prioritized based on communication with the sponsor. Typically software changes will be exercised first followed by defects during a development cycle unless the defect is born from a newly created software change. <describe the process and how you manage defects.>

### 8.3.3 Compatibility

All software developed with target the Java Virtual Machine (JVM) and DEVELOPER will make a final compilations and tests against the sponsor’s target infrastructure thus ensuring full compatibility with final environment.

### 8.3.4 Software Verification and Validation

Software testing is used in association with verification and validation:

*\* Verification: Have we built the software right? (i.e., does it match the specification)?*

*\* Validation: Have we built the right software? (i.e., is this what the customer wants)?*

The terms verification and validation are commonly used interchangeably in the industry; it is also common to see these two terms incorrectly defined.

According to the IEEE Standard Glossary of Software Engineering Terminology:

**Verification** is the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.

**Validation** is the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements.

All tests conducted will have a Verification and Validation component, meaning does the function perform what the requirement specifies and is the function performing properly. Verification will occur by confirming the requirement against the actual implementation. Validation will occur by actually executing the function to ensure the desired outcome is performed.

## 8.4 Testing Levels

Tests are frequently grouped by where they are added in the software development process, or by the level of specificity of the test. The main levels during the development process as defined by the SWEBOK guide are unit-, integration-, and system testing that are distinguished by the test target without implying a specific process model. Other test levels are classified by the testing objective.

### 8.4.1 Unit Testing

Unit testing refers to tests that verify the functionality of a specific section of code, usually at the function level. In an object-oriented environment, this is usually at the class level, and the minimal unit tests include the constructors and destructors.

It will not be necessary to unit test every function in the application, primary focus will be object (which contain data or perform operations on data) method unit tests. 100% code coverage is not a reality in most applications thus focus on unit testing will be at a discrete versus a raw code coverage value as performed by SonarQube.

Unit Testing – Does the project actively advocate and create Unit Tests for lower level function execution?

Click here to enter text. Coverage – Provide input on the percent (%) coverage your project has. Tools like Cobertura (Mavel plug-in), SonarQube, or similar tools can provide insight.

### 8.4.2 Integration Testing

Integration testing is any type of software testing that seeks to verify the interfaces between components against a software design. Software components may be integrated in an iterative way or all together ("big bang"). Normally the former is considered a better practice since it allows interface issues to be localized more quickly and fixed.

Integration testing works to expose defects in the interfaces and interaction between integrated components (modules). Progressively larger groups of tested software components corresponding to elements of the architectural design are integrated and tested until the software works as a system.

The [JUnit](http://junit.org/junit4/) framework can create integration test between objects.

Integration Testing – Does the project actively advocate and create Integration Tests for interaction of functions, objects, and procedures?

### 8.4.3 System Testing

System testing tests a completely integrated system to verify that it meets its requirements.

The completed compiled application will be tested in the DEVELOPER development environment and ultimately tested on DOD and NMCI workstations. <describe how system testing is performed and indicate by reference where standards for testing exist.>

## 8.5 Test Plans

The application testing process is vital in identifying security flaws before the application is released. In addition to the standard functional testing performed, the development team will have at least one tester specifically testing the application for security flaws. Standard security flaws will be evaluated on the development environment, review of source code for security flaws will be reviewed as well.

Test plans specify how you will test the application. Test plans say how you will test. Use Cases represent a function that the application is performing and is the basis for requirements/code creation. Test Cases are the embodiment of the Use Case and provide details descriptions of how the test of the Use Case will be conducted. Test Cases persist, have a specific date/time of execution, and individual performing the test. Integration tests could include any number and types of tests, they represent a body of tests that confirm past testing, which was successful, continues to be successful due to any change to the application or its components. An integration test passing represents acknowledgement of a known past functional state.

Automated testing is nothing more than tests, at any level of granularity, which can be invoked, typically triggered by an event like a build, and run by themselves with no human interaction. The resultant of an automated test should be persisted.

The following material in Section 8.5.\* is a Test Plan.

### 8.5.1 Named Tester/Test Team (APSC-DV-003130)

For each deployment a single developer will be designated as the Test Manager and responsible for the review of all tests performed and will consult with the Team Leader regarding the results of the tests performed.

The team will ensure tests procedures exist and at least annually executed to ensure system initialization, shutdown, and aborts are configured to ensure the system remains in a secure state (APSC-DV-003160).

|  |  |  |  |
| --- | --- | --- | --- |
| Name of Tester | Phone Number | Email Address | Last Test related training |
| DEVELOPER Developer | 555-5555 | [Developer.devleoper@developer.com](mailto:Gdit.devleoper@gdit.com) | 2017/01/01 – Software Testing via MyLMS. |

Table - Designated Testers

### 8.5.2 Test Procedures

The testing procedure will function as follow:

#### Unit Testing

Unit test will be created for the appropriate functions as prescribe previously in the code.

Unit test will be conducted before each commit to the repository.

#### Integration Testing

Integration testing will be written when two objects are communicating.

Integration test will be run before source is committed to the repository.

Selenium test will be conducted on web applications.

#### System Testing

System testing will occur by the developer prior to commit.

Updates to the test plan will occur after a successful system test with screen shot where appropriate.

Web application will have the Selenium test run by the developer.

#### Technical Acceptance Test

Will be conducted prior to task close out using these test plans.

All result of tests will be documented.

The full Selenium test suite will be executed and documented.

#### User Acceptance Test

Will occur minimally at customer’s resource, however, the sponsor will be consulted if DEVELOPER task required.

All result of tests will be documented.

Users will minimally execute the same functions tested with Selenium.

See the Deployment build process in the Application Configuration Guide for details on Security Scanners which will utilize fuzz testing (APP5100). Fuzz testing will be conducted primarily on web applications, desktop applications will be tested against standard set of attacks.

## 8.6 References

|  |  |
| --- | --- |
| Subject | Reference/Document |
| Junit | <http://junit.org> |
| Selenium | <http://www.seleniumhq.org/> |
| JMeter | <https://jmeter.apache.org/> |

# 9.0 Glossary of Terms

**Access Control** - based on your system classification, you will be given access to particular package functions specific to your job.

**Attribute** - information about a process or task.

**Checklist** - a list of items, each with a check box, displayed in the task window.

**Class** - a category, to which documents and process definitions are assigned, used to organize documents and process definitions.

**Graphical User Interface (GUI)** - visually oriented interface to talk with a program.

**Host** - generally a program that stores data for a client and provides that data to the client as requested.

**Primary Key** - a value that uniquely identifies each row in a database table. See Relational Database management on the Internet.

**Privilege** - an access control mechanism for processes.

**Python** - a script oriented language with functionality provided for both procedural and object oriented programming. Python is an open source language with strong community support.

**RDBMS** - Relational Database Management System

**RDBMS ER Diagram** - (Relational Database Management System Entity Relationship Diagram).

**Security Technical Information Guide (STIG)** - The documentation that specifies and explains the procedures for performing the SRR.

**SCM** - Software Configuration Management.

# 10.0 References

Application Development and Security (AppDev) Security Technical Implementation Guide (STIG) v4r3 : <http://iase.disa.mil/stigs/app-security/app-security/Pages/index.aspx>

Appendix A: Application Configuration Guide (ACG) ~ APSC-DV-003285

Review the Application Configuration Guide and determine if development systems are documented. If no development is being performed where the application is hosted this part of the requirement is NA.

Development systems, build systems, and test systems must operate in a standardized environment. See [4.4 Build Environment](#_4.4_Build_Environment) & [E.1 Required Software for Development](#_E.1_Required_Software) for details regarding the construction of the development and build environment.

## A.1 Classification Guide (APSC-DV-003290)

The application will run on a SECRET network (SIPR) and access a SECRET web server. The software associated with this design document is not inherently classified.

## A.2 Enclaves/Communications (APSC-DV-003285)

The application resides on server SECRET NMCI desktop and communicates with the DoD PKI enabled web servers on port 443 using the DoD PKI infrastructure.

### A.2.1 Ports and Protocols

### A.2.2 PKI Certificate Configuration Settings

#### A.2.2.1 Server Certificate Settings

#### A.2.2.2 Client Certificate Settings

## A.3 Encryption Settings

## A.4 Password Settings

### A.4.1 Architecture Components Passwords (Service Accounts)

### A.4.2 Application Passwords

### A.4.3 Database Passwords

## A.5 Audit Settings

### A.5.1 Audit Standards

### A.5.2 Log Locations

## A.4 Standardized Environment (APSC-DV-003215)

Refer to the [Build Environment](#_E.1_Required_Software) section for details.

## A.5 Best Practices (APSC-DV-003215)

Refer to the [Standards](#_4.5_Introduction_Coding) section of the document.

## A.6 Deployment Plan (APSC-DV-003130, APSC-DV-003160)

Refer to [Appendix B, Deployment Path & Plan](#_B.1_Deployment_Plan).

## A.7 Deployment Settings (APSC-DV-003285)

### A.7.1 Known Dependencies

The following are Dependencies that must be confirmed with each release:

1. To be elaborated on if new dependencies are added.

## A.8 Operational Backups

All code will reside in the DEVELOPER SVN repository which has nightly backups performed and a two week rotation with off-site encrypted backup.

### A.8.1 Database Backups

See[D.7 Database Backup](#_D.7_Database_Backup).

## A.9 Security Assumptions/Threat Model

Appendix B: Deployment Path & PLAN

# B.1 Deployment Plan

<Provide a reference, body of text, illustration, or any combination therein to help describe how the deployment of this software package will be conducted.>

FNMOC based deployments are scripted IAW 5234.2G and referenced in NAVOCEANO DoDIN Node SOA Policies Version 1.0, Section 4.3.2 Deliveries.

Prior to packaging a deployment the following steps will be executed:

1. Custom scripts are executed to perform source code cleanup and formatting.

1.1 Checkout Trunk.

1.2 Perform mass format of all source code.

2. Custom scripts are executed to test for specific conditions such as:

2.1 Scan for URL prefixes and suffixes.

2.2 Search System.print\*() method calls.

2.3 Check in Trunk.

2.4 Create tag for the release.

2.5 Output a report indicating status of aforementioned checks.

3. Jenkins-CI (continuous integration server) builds the application from the trunk of Subversion (in progress).

3.1 Jenkins-CI output is checked to ensure a proper build.

4. Security scans are performed:

4.1 Based on NAVOCEANO policy.

4.2 SonarQube (ASD v3 ~ APP3050) - an open source code analyzer reviews the final product as part of our deployment process. If unused libraries or code are detected they will be removed by the Release Manager and Step 1 will restart. Metrics such as SLOC, Code Coverage, and complexity are all recorded and will be stored in a persistent mechanism for future analysis.

4.3 A review of the scans is performed by a member of the team and report issued.\*

5. Tests are run against the application built for the proposed deployment. Use Case Tests will be applied with each release.

6. After a successful build and review of the previous Deployment Report artifacts by the team lead if approved to continue the software is ready for the next stage.

7. JavaDoc is run.

8. All media are packaged to CD-ROM with proper label, virus scanned, and delivered to the customer. A tag of the deployment materials is created against the trunk in the Deliveries repository.

Appendix C: Threat Assessment (APSC-DV-003230)

# C.1 Dread Implementation

### Security Misconfiguration

Attacker accesses default accounts, unused pages, unpatched flaws, unprotected files and directories, etc. to gain unauthorized access to or knowledge of the system.

***Mitigation***: Review best practices, turn off any functionality not required, and CM all settings/config files. FWB resources follow the STIG at OS, database, web server, and application server level which helps mitigate configuration errors save human fault. Primary concern would be protection of the Java security certificate stores.

8.8 = (DAMAGE (10) + REPRODUCIBILITY (5) + EXPLOITABILITY (10) + AFFECTED USERS (10) + DISCOVERABILITY (9)) / 5

### Insecure Cryptographic Storage

Attackers typically don't break the crypto. They break something else, such as find keys, get clear text copies of data, or access data via channels that automatically decrypt.

***Mitigation:*** N/A – DoD PKI infrastructure is assumed to be solvent solution and is FIP-140-2 compliant. These controls are Enterprise Class solutions and out of DEVELOPER control.

N/A = (DAMAGE (0) + REPRODUCIBILITY (0) + EXPLOITABILITY (0) + AFFECTED USERS (0) + DISCOVERABILITY (0)) / 5

### Insufficient Transport Layer Protection

Monitoring users' network traffic can be difficult, but is sometimes easy. The primary difficulty lies in monitoring the proper network's traffic while users are accessing the vulnerable site.

***Mitigation:*** SSL is used with a PKI certificate for ALL communications. Cookies are not used. Encryption at all network levels Outside Enclave and within is applied.

6 = (DAMAGE (5) + REPRODUCIBILITY (5) + EXPLOITABILITY (10) + AFFECTED USERS (5) + DISCOVERABILITY (5)) / 5

# C.2 Stride Implementation

### Spoofing Identity

The EAF uses PKI authentication to prevent the possibility of identity spoofing. In addition, new user account requests are vetted by a human agent before being granted access.

### Tampering with Data

The EAF uses both client-side and server-side validation to prevent both accidental as well as malicious data manipulation. The EAF edit pages use obfuscated input field names that do not match the rendered field names of a view page.

### Repudiation

The EAF uses both user roles as well as organizational filters to compartmentalize the portions of the site that a user is allowed to utilize. The EAF also incorporates a transaction log which saves relevant details of each transaction (minus views) per user. The EAF server and access logs are audited at regular intervals.

### Information Disclosure

The EAF does not store any information on the client-side. All data is saved and stored on the server-side.

### Denial of Service

The EAF does not support any functionality for anonymous users; all functions require authentication and authorization.

### Elevation of Privilege

The EAF enforces a strict authorization matrix for each page based on both roles and organizational filters. Privileged links are inaccessible to unprivileged users both by not being shown as well as prevention of use by the Controller (PageController performs role checks and org checks and enforces the annotated privileges on each page). Non-administrative users cannot modify their own access level.

Appendix D: Database Design

If you do not have a separate document, which should be referenced here, then use the sections below to help guide you towards some semblance of a database document.

# D.1 Definition

A database is an integrated collection of logically-related records or files consolidated into a common pool that provides data for one or more multiple uses.

http://en.wikipedia.org/wiki/Database

# D.2 Scope

A database management system (DBMS) consists of software that organizes the storage of data. A relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model.

The goal of this topic is to address the database management system which will be used for the ODCPPM application. Microsoft SQL Server is the relational model database server used in the multi-tiered architecture for the ODCPPM web application.

# D.3 Database Cluster architecture

Named database with schema (no cross-database referencing):

* ODCPPM (database)
* ODCPPM\_Data (schema contains plan data and report specification)
* ODCPPM\_Accounts (contains user, roles, and membership data)

# D.4 Configuration Management

Deployment scripts where appropriate

Maintain build scripts

Backups performed at Flight Weather Briefer rotation schedule

# D.5 Security

STIG - Database Security Technical Implementation Guide (STIG)

Department of Defense (DOD) Directive 8500.1 and DOD Instruction 8500.2 and the Information Assurance (IA) Controls.

Generic Database Security Checklist Version 8, Release 1.3 - U\_SRRChklst\_Generic\_V8R1-3.pdf

Generic Database Installation - U\_INS\_SRRChklst\_Generic\_V8R1-3.pdf

Generic Database - U\_DB\_SRRChklst\_Generic\_V8R1-3.pdf

## D 5.1 Authentication

Centralized

Establish Application-specific users - Separate database users for each web application

Establish database-owner different from default superuser

## D 5.2 Authorization & Authentication (Role Based Access Control [RBAC])

Authorization - authentication

role-based privileges to access and manipulate the data stored within the database objects

privileges to administer the database configuration and operation limited to superuser with server access

# D.6 Transaction logs

Database logging will be utilized for information regarding modifications by date/time, user of the application.

# D.7 Database Backup

## System level backup

<Provide input regarding how the RDBMS system as a whole is backed up.>

## Database level backup

<Provide input regarding how the database itself is backed up.>

## Application level backup

<Are any application level concerns appropriate?>

# D.8 Data Communications

Visual Basic communications with SQL Server are best described in this location:

http://technet.microsoft.com/en-us/library/aa198030%28v=sql.80%29.aspx .

Visual Basic communications with SQL Server is accomplished by way of ODBC. ODBC is a standard definition of an application programming interface (API) used to access data in relational or indexed sequential access method (ISAM) databases. Microsoft® SQL Server™ supports ODBC as one of the native APIs for writing C, C++, and Microsoft Visual Basic® applications that communicate with SQL Server. SQL Server Setup installs an ODBC driver for use with SQL Server when it installs the SQL Server client utilities.

ODBC defines a call-level interface, or CLI. A CLI is defined as a set of function calls and their associated parameters. A CLI definition uses a native programming language to call functions; therefore a CLI requires no extensions to the underlying programming language. This contrasts with an embedded API, such as Embedded SQL, where the API is defined as an extension of the source code for a programming language, and applications using the API must be precompiled in a separate step.

# D.9 Data Objects and Resultant Data Structures

Standard database tables are used; no stored procedures or esoteric objects are in use. Constraints are used to ensure data integrity and consistency.

# D.10 Entity Relationship Diagram

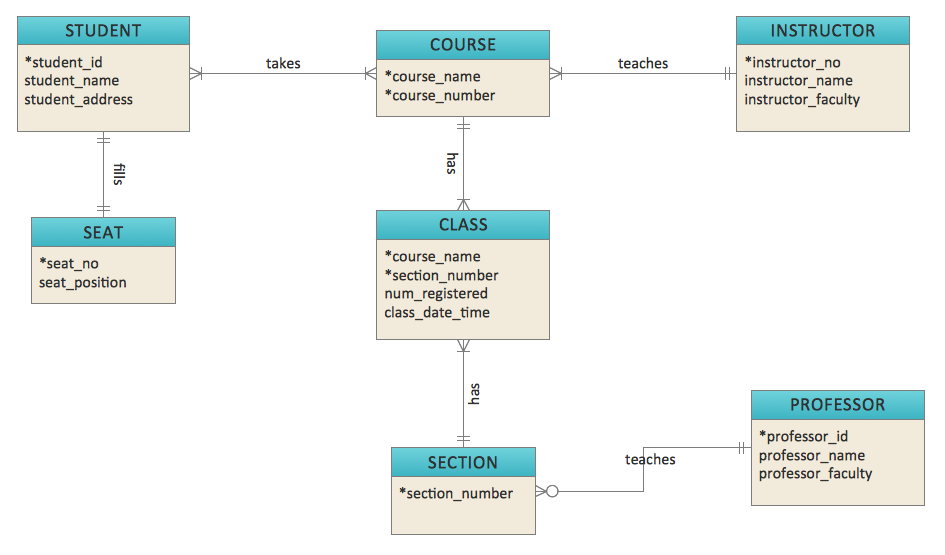


Figure - Entity Relationship Diagram (ERD)

# D.11 References

|  |  |
| --- | --- |
| Subject | Reference/Document |
| Secure Database Design Concepts | Translucent Databases, Peter Wayner, 2002, Flyzone Press |
| Database Concepts | <http://en.wikipedia.org/wiki/Database> |
| MS SQL Server Product Page | <http://www.microsoft.com/en-us/server-cloud/products/sql-server> |
| MS SQL Server Documentation | http://msdn.microsoft.com/en-us/library/bb54450.aspx |

Appendix E: Build Environment

The BAT (Build and Test) environment is located at the DEVELOPER facility. A 64 bit Windows Machine is used for testing the actual application. All CM utilizes the Subversion (SVN) tool to a centralized repository. As developers work on their code a branch is formed from the classic concept of a trunk. When complete and ready to "merge" the branch is merged with the trunk, rechecked out and compiled/tested to ensure the trunk remains pure.

# E.1 Required Software for Development

See 1.4 Software. Each solution is enumerated below to provide a detailed description

## E.1.1 Integrated Development Environments (IDE)

<if a specific IDE is required it should be documented here>

Both NetBeans and Eclipse are open source software packages, they are registered with DADMS. Integrated development environment (IDE) also known as integrated design environment or integrated debugging environment is a software application that provides comprehensive facilities to computer programmers for software development.

An IDE normally consists of:

* source code editor
* compiler and/or an interpreter
* build automation tools
* debugger

Sometimes a version control system and various tools are integrated to simplify the construction of a GUI. Many modern IDEs also have a class browser, an object inspector, and a class hierarchy diagram, for use with object-oriented software development.

## E1.2 Revision Control System Subversion

<name of repository solution> is a type of version-control system. Version control systems are the management of changes to documents, programs, and other information stored as computer files. It is most commonly used in software development, where a team of people may change the same files.

Application development will utilize the <name of code repository> methodology when it comes to source code version control. Below outlines the community accepted way to think about Subversion's structure.

**Trunk:** Main branch. This is where your next major release of the code lives, and generally has all the newest features.

**Branches:** Represent development efforts. Every time you release a major version, it gets a branch created. This allows you to do bug fixes and make a new release without having to release the newest, possibly unfinished or untested features.

**Tags:** Every time you release a version you make a tag for it. This gives you a point-in-time copy of the code as it was at that state, allowing you to go back and reproduce any bugs if necessary in a past version, or re-release a past version exactly as it was.

<name of code repository> layout will follow the standard trunk/branch/tag methodology.

## E.1.3 Developer Tools

|  |  |
| --- | --- |
| Required Functionality | Tools/Capability |
| SVN Repository | Maintained at DEVELOPER facility. |
| SVN at Developer Station | Tortoise SVN Shell Extension |
| Continuous Integration | Jenkins |
| Compilation/Dependency Resolution | Ant/Maven |
| Check style/Metrics | SonarQube run from DEVELOPER Facility |
| Issue Tracking/Team Integration | ECR |
| Wiki/Knowledge Management | Confluence |
| Version Control Indexing/Peer Review | Manual |
| Testing (Human) | DEVELOPER personnel and Customer, MS-Excel spreadsheets to capture test and results |
| Testing (Automated) | None |
| IDE (syntax highlight, folding, debugging) | NetBeans or Eclipse (build process will be agnostic to IDE) |

## E.1.4 References

|  |  |
| --- | --- |
| Subject | Reference/Document |
| SVN Red-bean book | <http://svnbook.red-bean.com/en/1.5/svn-book.html> |
| Subversion | [http:// https://subversion.apache.org/](http://en.wikipedia.org/wiki/Subversion_(software)) |

# E.2 Required Libraries

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | Version | DADMS Id | LDA | Deployed to Execution Environment? | Description |
| Example Name | 1.0 | 802119 | 5/20/2019 | No | Provides support for Unit Testing. |
| Example Name 2 | 1.9.1 | N/A | N/A | Yes | Provides API for creating PDF’s. |
|  |  |  |  |  |  |

Table - Library Dependencies