Department of Veterans Affairs

VistA Clinical Application & Enterprise Core Services

System Design Document

for

Enterprise Health Management Platform (eHMP)

Contract Number: VA 118-1011-0013



September, 2014

Version 2.2

Revision History

Note: The revision history cycle begins once changes or enhancements are requested after the System Design Document has been baselined.

Date	Version	Description	Author
10/23/2014	2.2.3	Increment 1 SDD Updated	ASM Research
9/26/14	2.2.2	Final edits. Submitted as contract deliverable for CLINs: 0003AS, 1001AS, 1002AS, 1003AS, 2004AS	[redacted]
9/22/14	2.2.1	Draft version posted for Final Edit.	[redacted]
9/8/14	2.2	Review and updates for eHMP 1.0. Submitted as Draft	[redacted]
5/23/2014	2.1.1	Updated Figure 3.5	[redacted]
5/15/2014	2.1	Revisions requested from 2.0	[redacted] Waltman
5/5/2014	2.0	Updates for MS 1 Increment 1, eHMP Project	[redacted]
4/22/2014	1.6	Updated terminology/graphics, MS 1 Increment 2, UX Project	[redacted]
3/3//2014	1.4	Updates to incorporate D. Waltman comments	[redacted]
3/3/2014	1.5	Updated terminology regarding project name.	[redacted]
2/26/2014	1.3	Updates to enterprise terminology	[redacted]
2/12/2014	1.2	Update after review	[redacted]
1/14/2014	1.1	PSI 1 Draft Submission	[redacted]
1/10/2014	1.0.3	Editorial Review	[redacted]
12/30/2013	1.0.2	Peer Review/Quality Assurance (QA) Review	[redacted]
12/13/2013	1.0.1	Updates to draft	[redacted]
11/7/2013	1.0	Submission for PMAS Milestone (MS) 1, UX Project	[redacted]

SME Input and Review

Name	Title	Organization	Role	Document Version
[redacted]	VistA Exchange Product Manager	OI&T	VistA Exchange Product Manager	1.1
[redacted]	VistA Exchange Product Manager	OI&T	VistA Exchange Product Manager	1.2, 1.3, 1.4
[redacted]	VistA Exchange Product Manager	OI&T	VistA Exchange Product Manager	1.6
[redacted]	VistA Exchange Product Manager	OI&T	VistA Exchange Product Manager	2.0, 2.2

Artifact Rationale

The System Design Document (SDD) is a dual-use document that provides the conceptual design as well as the as-built design. This document will be updated as the product is built, to reflect the as-built product. Per the Project Management Accountability System (PMAS) Guide, the SDD with conceptual design is required prior to the Milestone 1 Review. The as-built for each delivery must be incorporated prior to the Milestone 2 Review.

Table of Contents

1. Introduction 1
   1.1. Purpose of this Document 1
   1.2. Identification 2
   1.3. Scope 2
   1.4. Relationship to Other Plans 3
   1.5. Methodology, Tools, and Techniques 4
   1.6. Policies, Directives and Procedures 4
   1.7. Constraints 5
   1.8. Design Trade-offs 5
   1.9. User Characteristics 5
   1.10. User Problem Statement 5
2. Background 6
   2.1. Overview of the System 6
   2.2. Overview of the Business Process 7
   2.3. Assumptions 8
   2.4. Legacy System Retirement 8
3. Conceptual Design 8
   3.1. Conceptual Application Design 8
      3.1.1. Application Context 8
      3.1.2. High-level Application Design 9
      3.1.3. Application Locations 14
      3.1.4. Application Users 14
   3.2. Conceptual Data Design 14
      3.2.1. Project Conceptual Data Model 14
      3.2.2. Database Information 14
      3.2.3. User Interface Data Mapping 15
   3.3. Conceptual Infrastructure Design 15
      3.3.1. System Criticality and High Availability 15
      3.3.2. Special Technology 15
      3.3.3. Technology Locations 15
      3.3.4. Conceptual Infrastructure Diagram 15
         3.3.4.1. Load Balancer 17
         3.3.4.2. Web Service 17
         3.3.4.3. Cache 18
         3.3.4.4. Synchronization 18
4. System Architecture 18
   4.1. Hardware Architecture 18

4.2. Software Architecture 18
4.3. Communications Architecture 18
4.4. 3rd Party Software Components 18
5. Data Design 19
5.1. Data Model 19
5.2. Database Management System Files 20
5.3. Non-Database Management System Files 21
6. Detailed Design 21
6.1. Hardware Detailed Design 21
6.2. Software Detailed Design 24
6.2.1. eHMP Software Development Kit (SDK) 24
6.2.2. Web Server 35
6.2.3. HTTP Proxy 35
6.2.4. Web Service API 36
6.2.4.1. Implementation of the Web Service 36
6.2.4.2. Authentication 36
6.2.4.3. Access Control 39
6.2.4.4. Audit 40
6.2.4.5. Patient Search and Selection 41
6.2.5. Patient Sync Subsystem 42
6.2.6. Patient Correlation Service 42
6.2.7. Cache 42
6.2.8. VistA Access Approach 42
6.2.9. Patient Generated Data 43
7. External Interface Design 43
8. Human-Machine Interface 43
8.1. Summary 43
8.2. JLV Functionality 44
8.3. Patient Search 44
8.4. Patient Information: 44
8.5. HMP Functionality 45
8.6. Header 45
8.7. Footer 45
9. System Integrity Controls 50
9.1. Transmission Encryption 51
9.2. System Administrator Access 51
9.3. Flaw Remediation 51
9.4. Configuration Management (CM) 52
10. Approval Signatures 53

Appendix A – Acronym List 54
Appendix B – FHIR Mappings 57

Table of Figures

Figure 1-1 Layered Architecture with Planned Components 2
Figure 2-1 Current Means for Accessing VA Patient Data 7
Figure 3-1 - eHMP Business Logical View for ONC Certification 9
Figure 3-2 VistA Exchange Infrastructure Phase 1 11
Figure 3-3 VistA Exchange Infrastructure Phase 2 12
Figure 3-4 eHMP Regional Architecture 16
Figure 3-5 AITC Topology 17
Figure 6-1 eHMP System Architecture 24
Figure 6-2 CPRS Tab Access 39
Figure 6-3 Patient Sensitivity RPC Response 40

Table of Tables

Table 3-1 Application Design Objects 13
Table 3-2 Internal Data Stores 14
Table 4-1 3rd Party Components 19
Table 5-1 FHIR Pros and Cons 20
Table 6-1– Production Detail VM requirements 22
Table 6-2 – Pre-Production Detail VM Requirements 22
Table 6-3 – Shared Servers Detail VM Requirements 23
Table 6-4 – Extra Servers Detail VM Requirements 23
Table 6-5 Installed Software 36

1. Introduction

This System Design Document (SDD) has been developed, and will be regularly updated as part of the Enterprise Health Management Platform (eHMP) project. The eHMP project within the Veterans Health Information Systems and Technology (VistA) Evolution program will introduce expanded capabilities and modernize existing features of the Department of Veterans Affairs (VA) VistA Electronic Health Record (EHR) system. The eHMP project is a multi-year effort to evolve a modern, service-oriented enterprise Health Application Platform. This platform includes VistA Exchange, the eHMP Clinical Practice Environment (CPE), and certain clinical knowledge enrichment and decision support services. VistA Exchange is the clinical data services engine, federating clinical data from a variety of VA and Department of Defense (DoD) sources into an Enterprise Virtual Patient Record (eVPR).The eHMP CPE framework will incorporate capabilities currently provided by the Joint Legacy Viewer (JLV) and the hi2 Health Management Platform. eHMP CPE will eventually replace the Computerized Patient Record System (CPRS) as VA's primary point of care application. eHMP Services components will span all application layers, including presentation, business/core services, and data access. The eHMP project will produce epics (capabilities) supportive of the VistA 4 Product [*see VistA 4 Product Roadmap*] that relate to the core clinical user experience. For an in-depth introduction to the project, refer to the Contractor Project Management Plan (CPMP).

1.1. Purpose of this Document

The SDD translates the requirement specifications for each increment of the eHMP project into a design from which the developers can create the actual system. It identifies the top-level system architecture, and identifies hardware, software, communication, and interface components. More specifically, this document explains how the eHMP project integrates architecture spanning all layers—presentation, service, and data—and supports the vision of a robust core clinical user experience application as laid out in the Vista 4 Product Roadmap. The key mandates supported by the eHMP project include: **Department of Defense (DoD) Interoperability**, Commercial third party interoperability, **Meaningful Use (MU) Certification**, and full **Generation 3 Electronic Health Record (EHR)** features. Figure 1-1 below provides a summary of these layers and their planned components.

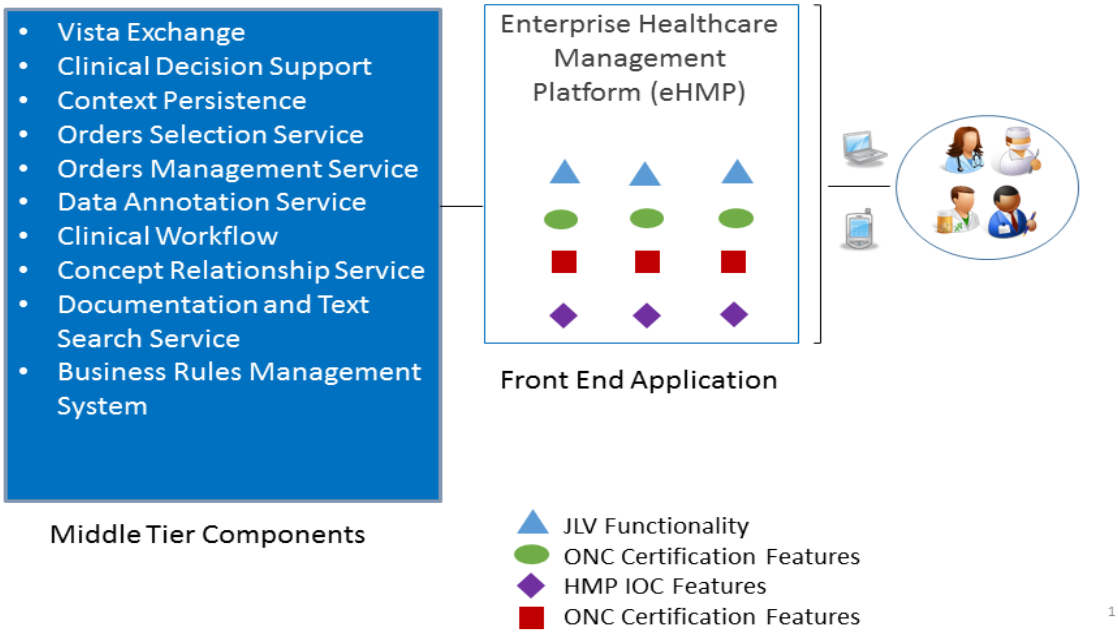


Figure 1-1 Layered Architecture with Planned Components

1.2. Identification

This version of the SDD describes ongoing evolution of the eHMP systems initiated under the VistA UX project, and continuing under the first increment of the eHMP project. Version numbers will be assigned to the resulting system once development of the increment has been completed and delivered to the VA. This document will be regularly updated to reflect the rapid, iterative evolution of the design, functional, and architectural capabilities defined for each increment of the eHMP project. This document is a living document, and will continually be updated as the system is developed.

1.3. Scope

Increment 1 Capability

The first Project Management Accountability System (PMAS) increment of the eHMP project will deliver the initial version of the eHMP CPE to access patient data cached in the eVPR. This UI will be connected to the VistA Exchange developed, tested and installed under the UX project. Increment 1 will continue use of the identity and access management service developed as part of the UX project. The eHMP CPE will include JLV functionality, allowing access to DoD data. This is in keeping with the goal of establishing one UI framework that will enable the eventual retirement of CPRS UIs and possible retirement of the existing JLV application. Other Increment 1 capabilities include the introduction of initial capabilities that will assist clinicians' use of the system, including enhanced patient search (when searching for a patient not registered in the user's local VistA site), free text search (finding entries in patient records based on user entered criteria), and context persistence. Although the VistA 4 Product Set

1 does not directly include eHMP Increment 1 capabilities, the completion of Increment 1 provides a foundational baseline for integration of all VistA 4 core clinical user experience features into a consolidated platform, based on VistA Exchange. This platform will form the foundation upon which additional capabilities will be built for VistA 4 Product Sets 2-4.

Future Incremental Capabilities
Based on priorities established by the eHMP Governance structure, future incremental capabilities are expected to include:

- **Enhanced DoD Interoperability**: Architectural improvements will expand performance and reduce maintenance for components supporting DoD interoperability.
- **Advanced indexing and clinical data enrichment**: VistA Exchange will leverage the eVPR cache to create a universal index for search, as well as allow the addition of external clinical enrichment features, e.g., natural language processing, tagging, and annotations.
- **Clinical Context Object Workgroup (CCOW)**: Synchronizing separate system components to a common "subject," e.g., patient or user.
- **Web-based Computerized Patient Record System (CPRS)**: One or more increments will be devoted to designing a web-based alternative to CPRS. This UI will be built leveraging designs, functionality and capabilities in the hi2 Health Management Platform (HMP) UI.
- **Open standards**: eHMP will enable VA to decouple ancillaries from core clinical functionality through open standards. One or more increments will be devoted to creating open standards-based services for Orders Management and Order Selection. This will allow orders to be managed as enterprise activities, and will decouple the generation of the clinical record from ancillary fulfillment, thus facilitating the selection and integration for Commercial-Off-the-Shelf (COTS) ancillary products, such as Laboratory and Pharmacy (the ancillary product acquisitions are outside the scope of eHMP project).
- **Other Evolutionary Generation 3 EHR Capabilities**: eHMP will enable other future evolutionary EHR capabilities, including services for Clinical Workflow and Activity Management, Decision Support, and Concept Relationship.

1.4. Relationship to Other Plans

This document discusses general system designs for the VistA Exchange and eHMP Web Application components within the eHMP project.

- **VistA 4 Product Roadmap:** The architecture and system components described in this document align with the VistA 4 Product Roadmap, Version 1, dated January 29, 2014. Particular areas in which this SDD and the VistA 4 Product Roadmap align include:
    - *Data Exchange Approach*: Fast Healthcare Interoperable Resource (FHIR), JavaScript Object Notation (JSON), and Representational State Transfer (REST), among other standards, will be used to support data exchanges with local VistA instances and outside data sources such as the DoD.
    - *User Experience and the UI:* Based on a web-services approach, the VistA Exchange architecture will support both web-browser hosted and locally installed, service-aware applications on various end user devices. The multi-layered approach (presentation, services, and data) described in this document is consistent with the qualities described for effective user experience.
    - *Use of Service-Oriented Architecture (SOA):* VistA Exchange will be built on web service architecture compliant with the principles detailed in the VistA 4 Product Roadmap.

- *VistA Infrastructure*: The VistA Exchange and eVPR services may be deployed within a virtualized VA cloud environment.
- *Data Model*: The eVPR cache will be implemented using a canonical model pattern, in which clinical data will be cached following a standard information model that allows for translation into various interoperability standard formats. The internal canonical models for the eVPR cache will be the VPR JSON canonical model, and FHIR-based canonical model.
- *Decision Support:* The eVPR will provide the aggregated patient records necessary to enable an efficient and effective clinical decision support system.
- *Meaningful Use:* This effort will enable VA to achieve the Meaningful Use goals outlined in the Office of Management and Budget (OMB) memorandum via EHR Meaningful Use.
- *Database Management System:* InterSystems Caché will be used to support the caching function of the eVPR service within VistA Exchange. VistA Exchange is not a new data source, but rather an aggregation/federation engine for data from existing Systems of Record.
- *Support for Future Ancillary Systems Integration:* Open standards will be used to facilitate the interfaces between the Orders Management service and future ancillary COTS.

- **Contractor Project Management Plan (CPMP)**: Summarizes the project and describes how the project will be managed.
- **Solution Architecture**: Brief summary of the architecture and design of the system.
- **Open Source Software Systems Engineering Plan**: This plan describes the engineering management approach and methodologies employed on this project.

1.5. Methodology, Tools, and Techniques

The SDD is a living document. This version reflects the ongoing system design as proposed and developed for increments 1 and 2 under the UX project and increment 1 under the eHMP project. Additional updates are planned as the design evolves within each increment, and for each subsequent increment, of the eHMP project. The Scaled Agile Methodology (SAM), summarized in the CPMP is a framework for a governance structure that aligns executive level priorities to an agile development methodology. Each increment of development will be managed through six two-week sprints grouped into Potentially Shippable Increment (PSI) cycles. Thus, the system design will be evolutionary with documented updates occurring with each PSI. In addition to PSI planning and development activities, other updates to the system design will be driven by VA Accreditation with Assessment and Authorization (A&A) Process and other certification events, e.g., user experience, 508.

1.6. Policies, Directives and Procedures

The system design will comply with VA policies, directives and procedures including:

- VA-One Technical Reference Manual (TRM)
- VA ProPath and PMAS standards
- VA A&A and related security policies
- VA Directive 6500 Information Security Handbook

If the planned system design is found to conflict with a VA policy, directive, or procedure, this conflict will be discussed with the VA Project Manager, who will help to determine whether a waiver or exception will be requested with the applicable VA governing board.

1.7. Constraints

For this current incremental release, the system is constrained so as not to be a system of record for any patient data. The VistA Exchange is an aggregation system, but any data aggregated will remain stored in source systems. The VistA Exchange's use of a data store is for caching/performance optimization only.

1.8. Design Trade-offs

The system responsibility is to provide a feature-rich and responsive UI. VistA Exchange provides the consolidated data services to drive the UI, including longitudinal patient data from various sources. It is expected that the number of users will be relatively low initially (during pilot stages), but will need to grow to the same number of users that utilize CPRS enterprise wide. Also, VistA Exchange could potentially support other consumers, such as DoD or veteran-centric mobile applications, staff-centric mobile applications, and external applications. Thus, VistA Exchange is architected to scale by ensuring that performance does not degrade as the number of users increases, with linear addition of hardware resources. This architectural focus can be demonstrated in the following design trade-offs:

- Favor caching data within VistA Exchange over retrieving all data from sources at every query. The cache allows for data to be aggregated from multiple sources, indexed based upon various usages, and to normalize terms against terminologies without having the user wait for these events to occur.
- Maximize the responsiveness of the UI by utilizing client side controllers, and only making network calls to retrieve data. This is achieved through the use of a Single Page Application.
- Favor an asynchronous API interface to maximize scalability over performance. When data is not already cached in VistA Exchange for a given patient, the API returns a deferred response using standard HTTP constructs, instructing the client to check again in a few seconds. During this time, the data is synchronized from its sources. This approach does minimally increase the time to get data in a situation where the data has not been cached prior to request, because the client will make a second request providing a delta between when the data is available in the cache to the time that the second request is made. However, this per transaction performance impact is expected to be overshadowed by the performance and scalability gains versus a synchronous interface. In the synchronous approach, HTTP connections would need to be maintained ("left open") for several seconds to minutes while data is retrieved from source systems, requiring more memory to the web API. This approach is further explained in section 6.2.3 (Web Service API).
- Favor the usage of microservices, rather than traditional web server clusters, to maximize discrete scalability options and to ensure that services remain loosely coupled.

1.9. User Characteristics

The first PMAS increment of this project will introduce the eHMP CPE for authorized users throughout a VA medical facility. Users will benefit from accessing a comprehensive patient record that includes documented care within the VA and DoD medical systems. Clinicians will use the system to help make health assessments, diagnosis and determine treatment based on a more complete longitudinal patient record. Clinicians will continue to use CPRS to document delivery of care, including order entry, until those CPRS capabilities are replaced by later increments of this system. It is also expected that in future increments of this system, select benefits claims examiners of the Veterans Benefits Administration (VBA), will use the system to help adjudicate benefits claims that require review of both DoD and VA medical records information. Initially, and through at least 2018, the VBA examiners will use the JLV functionality within eHMP.

1.10. User Problem Statement

The eHMP system represents the portion of the VistA 4 Product that implements end-user facing features—i.e. the User Experience/UX—for the clinical users (the team members that are responsible for direct patient care) with the VistA Exchange web services architecture. As such, it includes the UI as well as underlying supporting services and access to data. The eHMP system is an extension/evolution of VistA and does not represent a system wholly separate from VistA. Specifically, the eHMP system is not meant to produce a separate System of Records (SOR) apart from the existing VistA SOR.
The following excerpt from the Vista 4 Product Roadmap defines the user problem statement that the eHMP project addresses:

> "Today, VA providers depend on a point-of-care GUI called Computerized Patient Record System (CPRS) for accessing and recording most of the patient information needed for clinical documentation and decisions. CPRS provides very limited capability for accessing patient information that may have been recorded in many different locations both during and after a Veteran's military service. While CPRS has been an effective tool for over 17 years, its GUI is based on a paper-chart metaphor and segregates data that need to be considered together to improve understanding of the patient's state of health. For example, medications often need to be considered with laboratory tests to understand how medications might affect these results in negative or positive ways. In CPRS, users must frequently switch between windows to see both medication and other data.
>
> The VistA 4 GUI will facilitate display of interoperable data and help drive data standardization. The VistA 4 GUI, when integrated with a shared VA/DoD interoperability platform, will provide access to a Veteran's complete longitudinal health record, comprised of data collected from every military and VA treatment facility in which the Veteran has received care. For many Veterans, health data from other providers will be available through eHealth connections with participating institutions. The rich, multi-sourced data comprising a Veteran's longitudinal health record must be understood by all systems accessing and adding to it, including the VistA 4 GUI. The VistA 4 user experience will drive advances in the application of national standards for coding, sharing and interpreting health information both in VA and the nation. For example, in order to reliably graph vital signs or laboratory results, the GUI must be served homogeneous data such as a particular chemical measured by blood and not urine tests. The units of measurement must also be the same to allow meaningful comparison of all results."

The eHMP project will address these problems in the context of three major VA mandates:

- DoD Integration as defined by the 2014 National Defense Authorization Act (NDAA) legislation and 2014 Consolidated Appropriation acts
- Meaningful Use EHR certification using the 2014 ONC criteria (stage 1 and 2)
- Generation 3 EHR capabilities

2. Background

2.1. Overview of the System

The eHMP project is a multi-faceted development effort to modernize VistA. At its completion, this effort will produce:

- **Enterprise Virtual Patient Record (eVPR)**: A longitudinal patient record for a given patient that includes data from VA sources (such as VistA), as well as external sources such as the DoD and Nationwide Health Information Network (NwHIN)/eHealth Exchange.
- **VistA Exchange**: Middleware that provides the eVPR, allowing for delivery of a single, common health record to be utilized throughout the continuum of care. This implementation includes a web service API to access a given patient's data; retrieval, indexing, and aggregation of clinical data from current VistA instances (and eventually other data sources); caching of the patient record to optimize performance; document enrichment to add additional data to a given patient record event; and normalizes the data from various sources using standard terminology terms.

- **eHMP Web Application**: Includes the integration of selected HMP UI features, and selected JLV UI features. JLV functionality and the eHMP Web Application will support ONC 2014 EHR certification.
- Clinical Decision Support Service
- Clinical Workflow and Activity Management Service
- Orders Management and Orders Selection Service
- Concept Relationship Service
- Universal indexing and searching
- Data Annotation
- Persistence of context for a clinical user across devices
- Integration with VA enterprise Identity and Access Management

This document covers the portion of the eHMP project that is currently being developed, specifically the introduction of the eHMP CPE enabling viewing of aggregated VA and DoD patient data with initial supporting functions, e.g., patient search, free text search.

2.2. Overview of the Business Process

Presently, using CPRS, most VA clinicians have access only to patient data that resides at a single VistA location. Though standalone local demonstration projects exist through HMP and a single means to access enterprise-wide patient data only exist with the view-only JLV application, no enterprise wide application with both a data view and write back exists. Figure 2-1 Current Means for Accessing VA Patient Data identifies the current state of how patient record data is extracted from VistA. In addition, though CPRS has been an effective UI to local VistA data, it requires substantial efforts to update the thick client server configuration that is essentially bound to desktop computers.

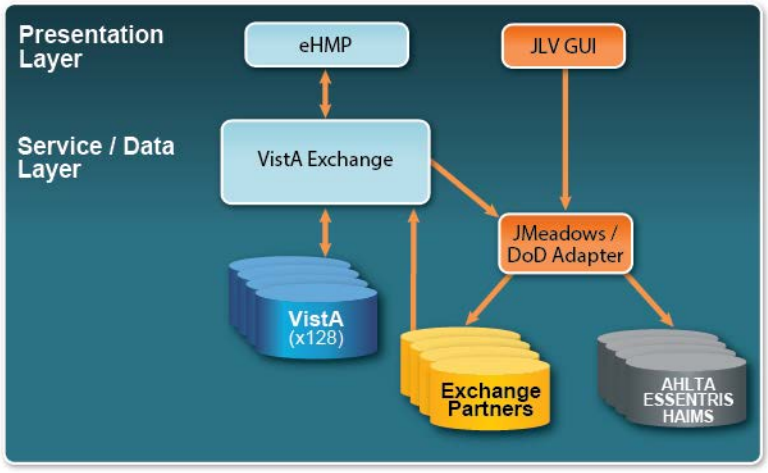


Figure 2-1 Current Means for Accessing VA Patient Data

Through the eHMP project, the presentation, service, and data layers will be modernized to enable the creation of a comprehensive patient record extracted from each VistA instance and outside sources including the DoD and outside partners. Figure 2-1 Common Platform for Accessing Patient Data depicts means for accessing patient data with eHMP. The resulting architecture will provide a single, adaptable, and flexible web-based UI allowing to the use of web-based services. To support the access and display of DoD and other externally sourced patient data, a JLV functionality (the same as DoD's JLV functionality)

will be available in the eHMP CPE. Beneficiaries of this single UI, with multiple views, will include clinicians and Benefits claims examiners who also require access to a comprehensive EHR.

2.3. Assumptions

The following assumptions were made in preparation for Increment 1 of this project:

- The proposed architecture and system design are accepted by VA Office of Information & Technology (OI&T) and Veterans Health Administration (VHA) decision makers.
- The VA Project Manager, under guidance from the Configuration Control Board (CCB), will determine when increments developed under this project will be deployed into a production environment; one or more increments could be bundled for deployment and submitted for PMAS Milestone 2 review.
- This SDD will be regularly updated and identified with version numbers to describe the expanded system design for each PSI and PMAS increment.
- Unless otherwise stated, software will be open source and compliant with VA's Technical Reference Manual (TRM).

2.4. Legacy System Retirement

The eHMP Web Application provides support for clinicians at point of care. This includes allowing a clinician to view the patient's data from VA, DOD, and other sources; update a patient record by updating data in VistA; perform a patient encounter; and place orders.

This functionality, at full deployment, potentially allows for deprecation of VistaWeb, BHIE Viewer, JLV and CPRS.

3. Conceptual Design

3.1. Conceptual Application Design

3.1.1. Application Context

Figure 3-1 provides context for eHMP project, in relation to other VA and DoD systems. It shows the context in which VistA Exchange and the eHMP CPE will operate towards meeting ONC 2014 Certification during the August 2015 period.

The system under development for "eHMP Increment 1" includes VistA Exchange and the eHMP CPE which will be introduced to the system. Further, VistA Exchange will be enhanced to support other eHMP services, offering patient data optimized for eHMP functionality and supplying operational data (such as lookup lists).

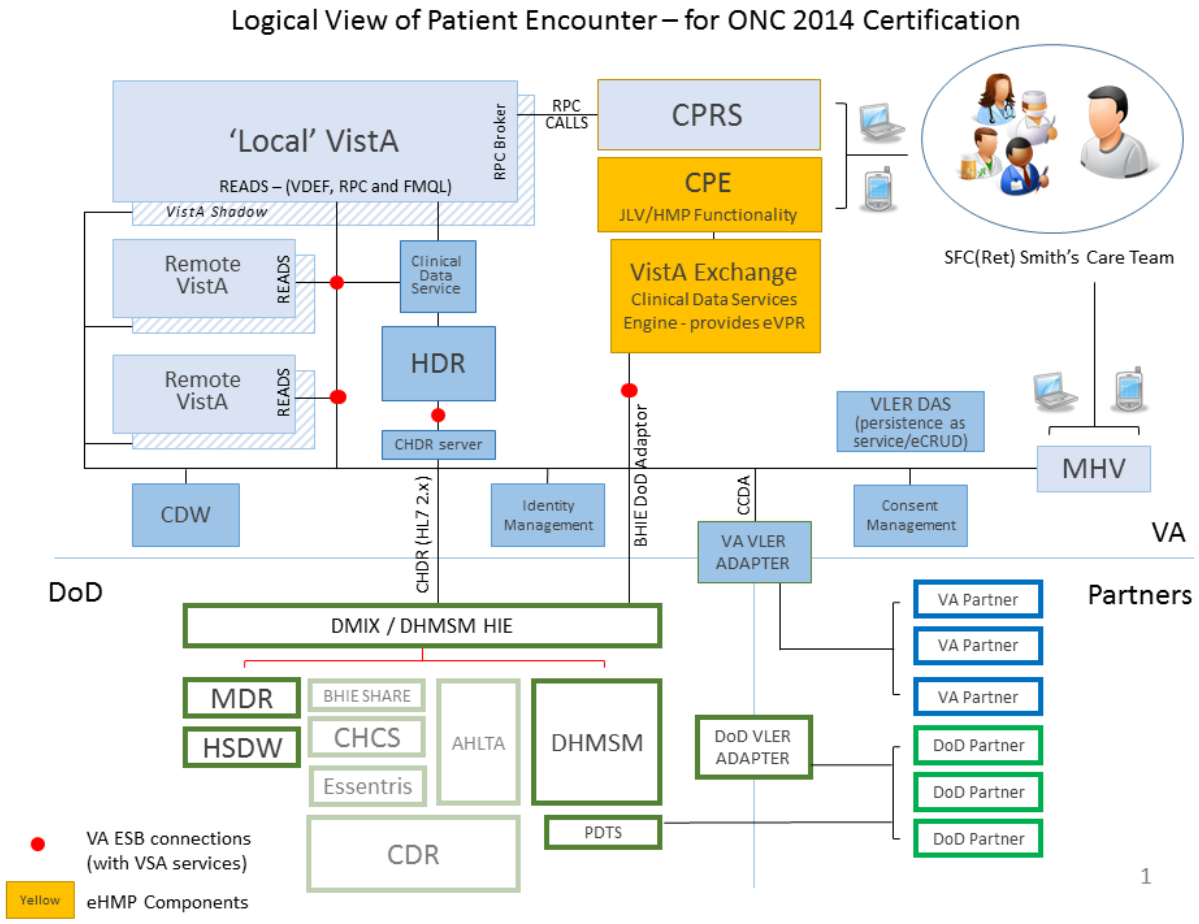


Figure 3-1 - eHMP Business Logical View for ONC Certification

Based upon current VistA Exchange requirements, the focus is on providing the patient record from VistA, with future requirements planned on access from other systems.

3.1.2. High-level Application Design

The system under development for eHMP Increment 1 includes VistA Exchange and the eHMP CPE. The high-level application design for each application will be described below.

VistA Exchange

VistA Exchange is responsible for providing an enterprise patient record, including data from the VA and DoD. Based upon business triggers (such as an appointment, admission, or patient search), VistA Exchange begins to aggregate data from source systems for a selected patient. The data is normalized using standard terminologies to include consistent terms; that data is stored in an indexed method to ensure rapid retrieval based upon the patient identifier or through free text search.

VistA Exchange will leverage the HMP Virtual Patient Record (VPR) logic. During early deployment of VistA Exchange, the eVPR will be used essentially as is. VistA Exchange will then be evolved to achieve scalability and performance goals as features and users are added. This evolution is depicted in the diagrams below where Figure 3-2 depicts Phase 1 and Figure 3-3 depicts Phase 2. Tables 3-1 and 3-2 describe each entity depicted in the Phase 1 and 2 figures. This evolution may iteratively happen during Increment 1, or in future increments. The primary influencer to drive these changes will be in the ability

to acquire direct connections to VistA and DoD Adapter (or the ability for the Enterprise Service Bus (ESB) to mediate these connections) and the ability to deploy the HMP VPR pub/sub remote procedure calls (RPCs) across the enterprise.

The VistA Exchange web API allows a user to retrieve a patient's record by ICN. If that patient has not been synchronized (does not currently have a valid record in the cache), a synchronization request is initiated. This establishes a subscription against a "local" site using a set of HMP VPR RPCs. Within VistA, the existing data for the patient is gathered and made available to the cache. All future changes to the patient's record will be made available to the cache. This process is explained in further detail in Section 6.2.

In addition to getting data from the local site, Vista Exchange retrieves data from remote VA and DoD sites via Clinical Data Service (pull) and the DoD Adaptor. Eventually, VistA Exchange data will be retrieved from VistA via pub/sub (push/pull).

The data is normalized using a set of terminology tables. This process ensures that the data has a standards-based term for certain concepts within the patient record, such as using an LOINC-based term for a lab test.

Data is cached in two ways. First, the data is stored within a custom document store (JDS), developed as part of HMP, and hosted by an InterSystems Caché database. This is used for primary retrieval of data by patient. Second, the data is stored within Searching On Lucene w/Replication (SOLR), optimized for free text searches across a patient's record.

eHMP Web Application

The eHMP web application provides functionality to support a clinician at the point of care with a patient. It is the application that will ultimately perform many of the functions of CPRS, and utilizes VistA Exchange as its source of patient data. The web application is developed as a Single Page Application, with client-side model view controller (MVC). This approach will limit the impact on the user experience associated with network latency as all behavior logic is contained within the client, not the server.

The application provides a development time Software Development Kit (SDK) and a runtime container in which small units of functionality can be developed and executed. The SDK is described in section 6.2.1.

Figures 3-2 and 3-3 describe the VistA Exchange Infrastructure in Phases 1 and 2 respectively.

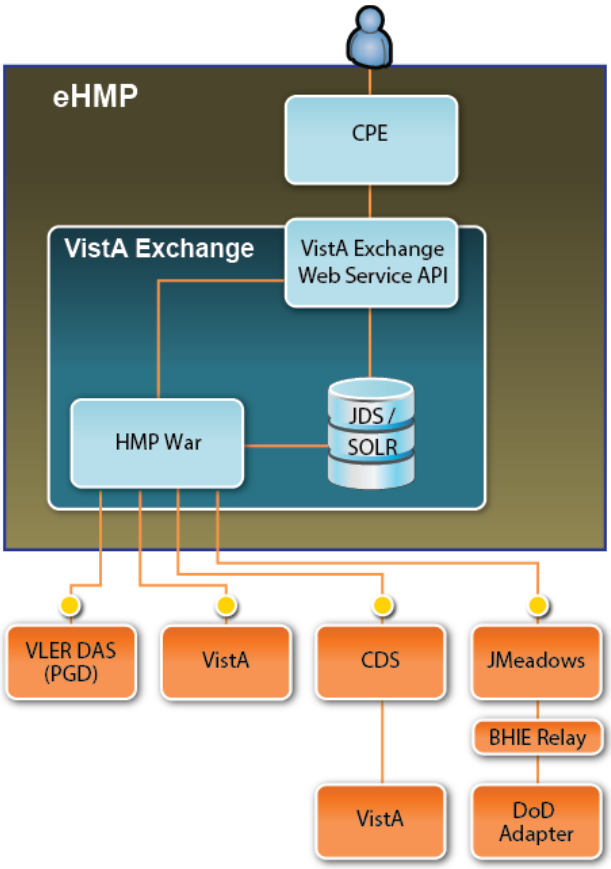


Figure 3-2 VistA Exchange Infrastructure Phase 1

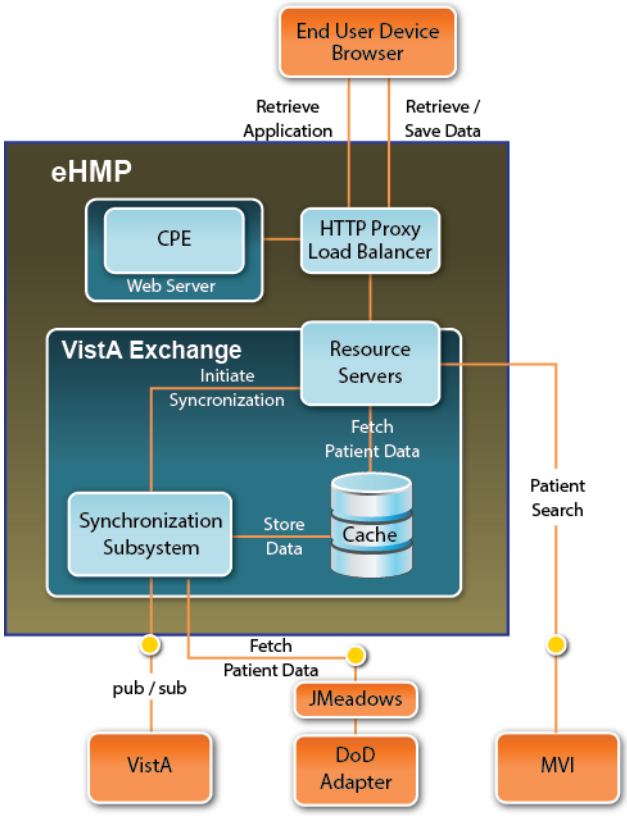


Figure 3-3 VistA Exchange Infrastructure Phase 2

Application Design Objects

Table 3-1 Application Design Objects describes the primary objects of the high-level application design for Phases 1 and 2.

ID	Name	Description	Service or Legacy Code
1	eHMP System	The system under construction, which includes eHMP CPE and eHMP Services (VistA Exchange and other clinical components)	
2	eHMP CPE	The web application to support provide point-of-care healthcare activities	
3	JLV functionality	Application components to viewing of DoD data.	
4	VistA Exchange	The middleware that aggregates, normalizes, and indexes the enterprise patient record	
5	VistA Exchange Web Service API	Exposes an API that allows for consumers (such as the eHMP CPE) to use to fetch a given's patient's record.	
6	JDS	Custom JSON document store, originally part of the HMP platform. This stores patient data in the form of JSON documents	
7	SOLR	SOLR indexes patient data, both structured and unstructured, to allow for full text searching	
8	HMP war	This web application archive (WAR) is the deployable artifact from HMP. For VistA Exchange, this is primarily used to initiate synchronization with VistA	
9	VistA	VistA is used as a source of clinical data for the patient	
10	JMeadows	JMeadows is a middleware component part of the JLV stack used from for retrieving data from the DoD and VA	
11	BHIE Relay	BHIE Relay acts as an adapter from the JMeadows SOAP service to the DoD Adapter REST interface	
12	DoD Adapter	DoD Adapter is a set of REST web services to retrieve data from various DoD systems	
13	MVI	Master Veteran Index provides the ability to search for patients and to translate veteran patient identifiers to other patient identifiers	
14	CDS	Clinical Data Services is used to retrieve data from remote VistA sources	
15	ESB	Enterprise Service Bus (denoted by yellow circles at connection points in the figures) – if available at the time of deployment, the ESB will mediate web service connections with the VA network.	

Table 3-1 Application Design Objects

Internal Data Stores
Tables 3-2 Internal Data Stores describes the internal data stores of the high-level application design for Phases 1 and 2.

ID	Name	Data Stored	Steward	Access
1	JDS	Cached instance of the patient health record data	VistA Exchange	This is read-only via the patient record web service API
2	SOLR	Cached instance of the patient health record data, optimized for full text search	VistA Exchange	This is read-only via the patient record web service API

Table 3-2 Internal Data Stores

3.1.3. Application Locations

The application integration environments will be located within the VA Enterprise Development Environment (EDE) cloud based out of the Austin Information Technology Center (AITC). The Pre-Production and Production environments will be in the VA Enterprise Operations (EO) cloud environment, also based out of AITC, and eventually to other regional data centers (as described in section 3.3.4). The EDE and EO backup cloud environments are located within the Philadelphia Information Technology Center (PITC).

3.1.4. Application Users

Application users of the eHMP system will be clinicians at VA medical facilities. Future users are expected to include VBA claims examiners, using the JLV application view, who require access to comprehensive VA and DoD patient records.

3.2. Conceptual Data Design

3.2.1. Project Conceptual Data Model

The canonical data model for eHMP is “VPR”. The VPR model is that which has been established by the Health Management Platform (HMP). This is the native format that is exposed by the HMP RPC’s for its request/response and pub/sub routines. This is a custom format.

In addition to the RPC’s, this format is realized in the JDS (the VistA Exchange cache), the VistA Exchange API (the set of web services exposing the clinical data), and the transport model for the web application.

In addition to the VPR web services, there is a set of FHIR based web services API’s for VistA Exchange. For these services, the data is transformed just in time from VPR to FHIR.

The choice to support these two models is based on the following assertions:

- the existing HMP RPC’s natively support VPR
- the web application needs a lightweight view model
- the VPR model is better optimized (lightweight) to be a view model than FHIR
- VistA Exchange needs a standards-based transport model to support interoperability
- FHIR is standards based

The data model is described further in section 5.1.

3.2.2. Database Information

The clinical data is cached in a subsystem called JDS, which is an M-based document data store developed originally as part of the HMP program. JDS provides an HTTP interface that allows access through a set of HTTP GET and HTTP PUT commands. JDS is hosted within InterSystems Caché.

3.2.3. User Interface Data Mapping

The user interface mapping to the data model will occur during the development phase of this increment.

3.3. Conceptual Infrastructure Design

The deployment of the eHMP system (herein, the "system") will be done using a multi-instance approach. Each instance is an autonomous deployment of the full stack of software, including the eHMP CPE web application and VistA Exchange. It is expected that at full deployment, there will be four instances, each to support 1/4 of the VistA sites, and a fifth instance to support external consumers. This configuration is referred to as the "4+1" model. Figure 3-5 diagrams the "4+1" model.

Each instance is autonomous from the other instances. An instance provides for a front-end proxy to direct HTTP traffic from the end user browsers or other consumers of the system; a web server to provide the static HTML/JavaScript/CSS of the web application; a cluster of middleware servers; a Caché instance to host the cached patient record; and synchronization processes to maintain data from VistA and retrieve data from other sources.

An instance has affinity towards a given set of VistA sites. That is, for a given site, when users access the eHMP CPE web application, they will be directed to a specific instance of the system. That instance, however, will have the ability to collect data from all VistA instances to ensure that the patient data is an enterprise perspective, including all VistA instances and external sources such as DoD. Although not required, it is expected to deploy each instance of the system to a different geographic data center. These can be organized to be optimized for the lowest network related latency for end users.

3.3.1. System Criticality and High Availability

The application integration environments will be located within the VA EDE cloud environment based out of AITC. The Pre-Production and Production environments will be in the VA EO cloud environment, also based out of AITC. The EO environment is a Federal Information Security Management Act (FISMA) high cloud environment and, as such, has disaster recovery and tape backup already in place. The EDE and EO cloud environments are also located within PITC.

3.3.2. Special Technology

No special technology is needed.

3.3.3. Technology Locations

The application integration environments will be located within the VA EDE cloud environment based out of AITC. This is the environment in which integration testing will be performed. The Pre-Production and Production environments will be in the VA EO cloud environment. The initial instance will be in AITC, rolling out to other data centers in future increments. Section 3.3.4 describes the future deployment model, which expands to regional data centers.

3.3.4. Conceptual Infrastructure Diagram

The full deployment of the system includes four autonomous instances, along with a fifth for supporting external consumers ("4+1" model). The VA has four regional Datacenters that will have Enterprise Operations (EO) cloud environments. The EO cloud environments will be deployed in each of the four

regional data centers where the VistA systems are also located. The +1 deployment can happen at any of the 4 regional datacenters and will be used for the external calls to systems like the DoD. Each regional datacenter will have the same deployment model/structure. Figure 3-4 diagrams the "4+1" model.

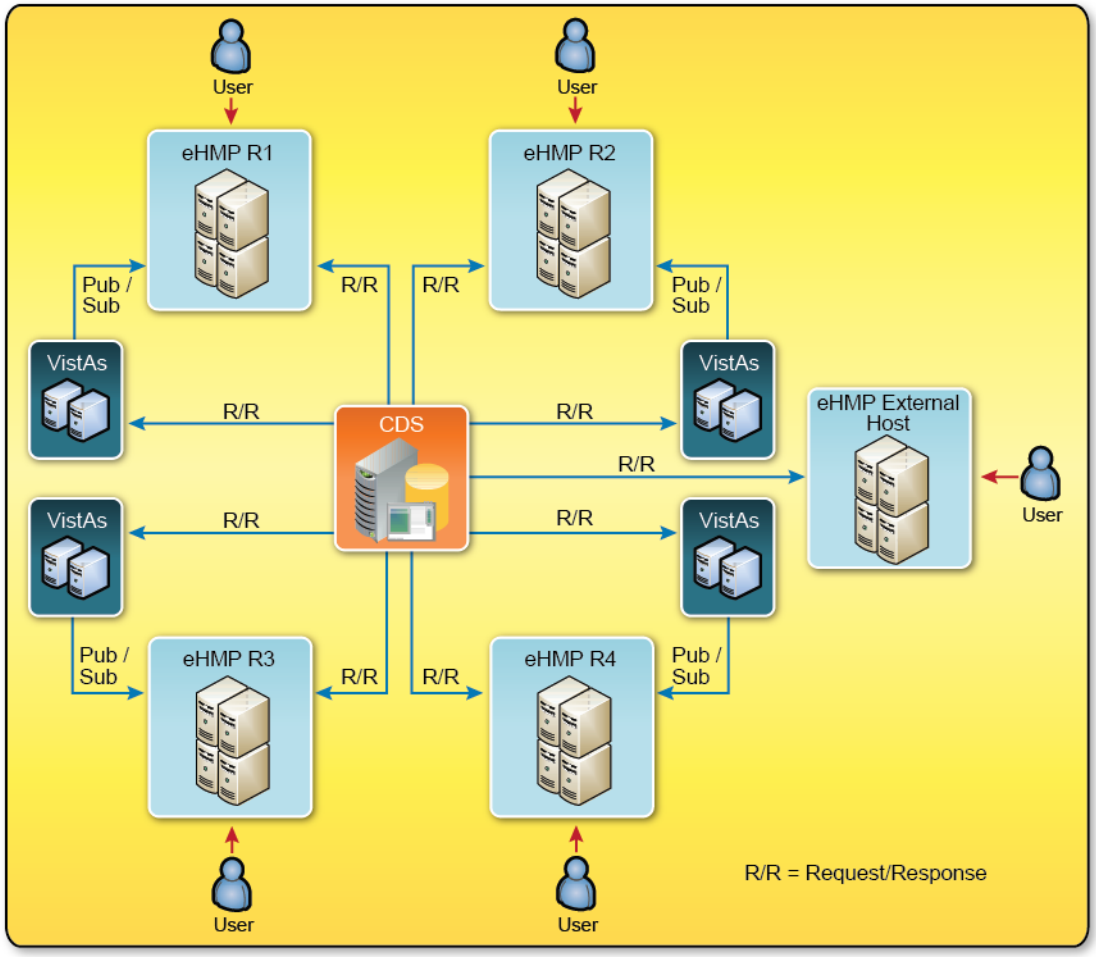


Figure 3-4 eHMP Regional Architecture

Each eHMP instance of the system in the Regional datacenters will be primarily tied to the VistA systems that are collocated at the datacenter.

Figure 3-6 AITC Topology diagrams the initial instance infrastructure. The initial instance will be deployed at the AITC EO cloud environment. The environment will sit behind the AITC firewall(s) and will be hosted on EO VMWare ESXi hosts. Each of the ESXi hosts will be connected to a Storage Area Network (SAN) that will hold all data and virtual images. EO will manage the SAN and ESXi servers and AITC will manage the firewall(s).

The virtual server will consist of a front end load balancer (unless an F5 or some other Load Balancing appliance can be provided) that will sit in front of the UI and API servers. The front end load balancer (hardware or virtual machine) will allow for easy expansion of the UI/API servers to meet capacity/load needs as they arise.

The Storage and Synchronization servers will be placed behind the UI/API servers. The storage server will be a single instance that will host JSON Data Store (JDS) and SOLR, with additional servers to be added as needed. The Synchronization server will be configured as a two-box cluster and expanded as needed.

Location within AITC will enable full connectivity to required remote systems, including VistA instances, MVI, Corporate Data Warehouse (CDW), Health Data Repository (HDR)/Clinical Data Service (CDS), and Veteran Lifetime Electronic Record (VLER) Data Access Service (DAS). Figure 3-5 AITC Topology further describes the above characteristics of the AITC environment.

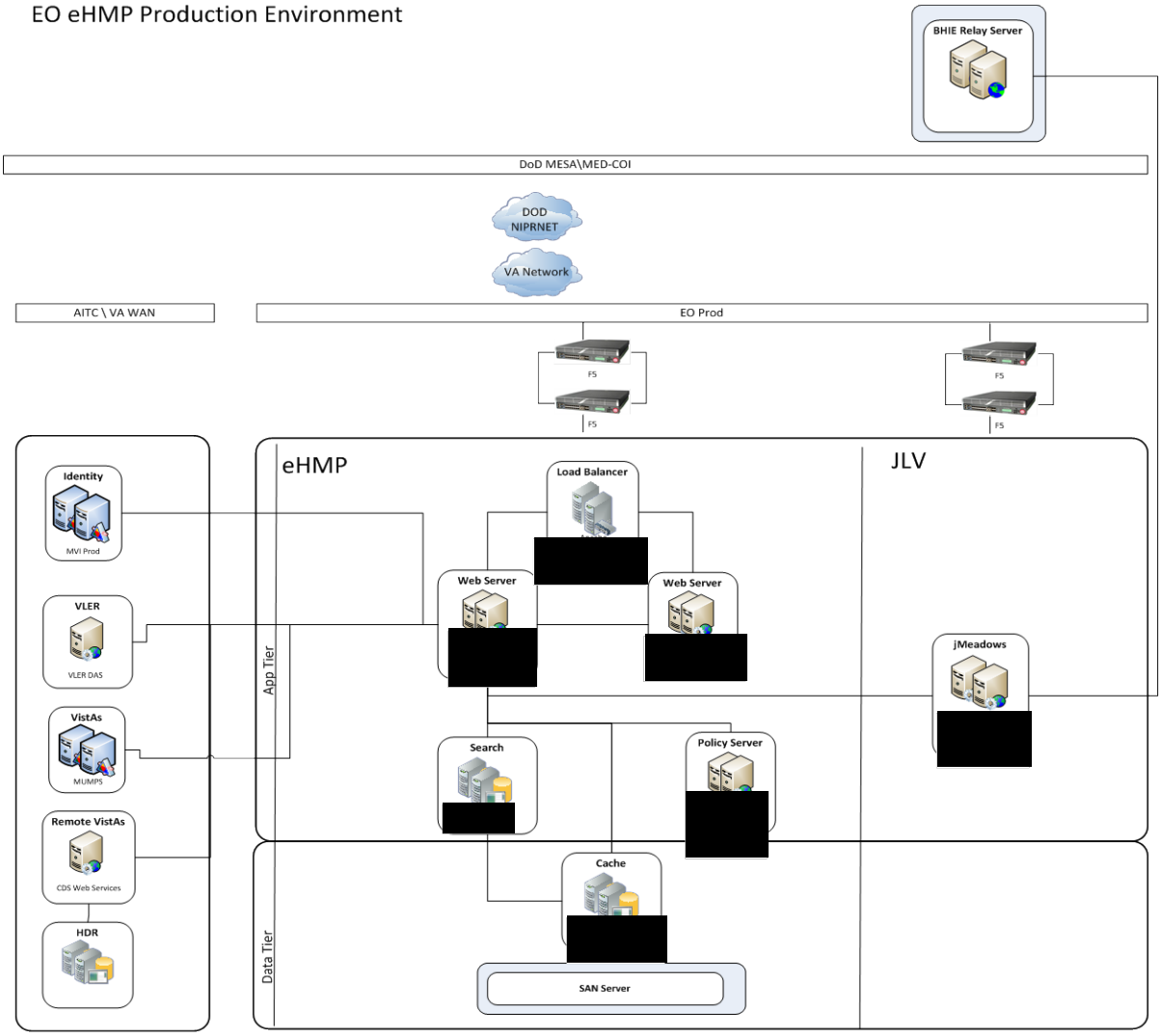


Figure 3-5 AITC Topology

3.3.4.1. Load Balancer

The load balancer will consist of an Apache server running Red Hat Enterprise Linux (RHEL). Additional details on the load balancer will be provided with updates to this document.

3.3.4.2. Web Service

The web server FHIR API cluster will be supported by Jetty running RHEL 6.5. Additional details on the web server will be provided with updates to this document.

3.3.4.3. Cache

The cache server will consist of SOLR and JDS running on RHEL 6.5. Additional details on the cache server will be provided with updates to this document.

3.3.4.4. Synchronization

The synchronization cluster will consist of Jetty and AMQ running on RHEL 6.5. Additional details on the synchronization virtual machine will be provided with updates to this document.

4. System Architecture

The development process and architecture are characteristic of the following key components:

- First, the infrastructure follows a principle of automation. This automation includes compile/build/packaging, infrastructure deployments onto local developer workstations, and infrastructure deployments in test and production.
- Second, this system architecture favors small, lightweight infrastructure (termed micro-services), rather than large, unmanageable infrastructure. This manifests itself in the use of small, autonomous, executable Java Archive (JAR) files used for web services and message handlers, rather than a traditional application server.

The system developed under the eHMP project will be designed to run in a cloud-based environment.

4.1. Hardware Architecture

The program will be setup within the VA's EO cloud environment. For details on the hardware architecture, please refer to section 6.1 Hardware Detailed Design.

4.2. Software Architecture

The software architecture is explained in section 6.2 Software Detailed Design.

4.3. Communications Architecture

All calls coming into the system will be over 443/HTTPS as either a web call from a browser to the UI server, or an REST API call to the API server. Communication from the UI/API servers will be over secure channel to either the synchronization server or to the JDS server. Communication to remote systems, VistA/CDS/DoD Adapter, will be over HTTPS on various ports, mostly 443.

4.4. Third Party Software Components

There are a number of third party software components planned for this increment. Table 4-1 Third Party Components describes each component, version, and role.

Component	Version	Role
Caché	2011	InterSystems Caché database to store the cached patient record.
Java Virtual Machine (JVM)	1.7.0_25	The JVM is used to host the VistA Exchange API
Jetty	TBD	Jetty is a servlet container used by the VistA Exchange API
Jersey	TBD	Jersey is the JAX-RS implementation used by the VistA Exchange API

Component	Version	Role
Red Hat Enterprise Linux (RHEL)	TBD	Primary hosting software.
SOLR	TBD	Creates full text indexes, allowing for text search across the entire patient record.

Table 4-1 Third Party Components

For release 1.1, this will expand to include:

Component	Version	Role
Caché	2011	InterSystems Caché database to store the cached patient record.
Java Virtual Machine (JVM)	1.7.0_25	The JVM is used to host the VistA Exchange API
Red Hat Enterprise Linux (RHEL)	TBD	Primary hosting software.
Node.js	TBD	Runtime engine for JavaScript
Express.js	TBD	Node.js module to handle routing of HTTP calls
SOLR	TBD	

5. Data Design

5.1. Data Model

Based on the project team's review of the given options, it was recommended that HL7 FHIR and VPR JSON be selected as the canonical model family to be used both for storage as well as the external service-facing interface. HL7 FHIR has support of HL7 and based on the analysis performed, will be able to support all of the required data types. It can also be extended to support any VA-specific data fields within the data types. In addition, it provides support for handling the various terminology code sets that will be used within the VA. The freely available tools will give a jump-start to the development of eVPR, and it easily supports the technologies that will be required. eVPR JSON is the native JSON document model used by HMP, and will provide a performance-tuned eHMP CPE model. Together these models will enable the eHMP system to meet both the interoperability and UI requirements currently defined. Table 5-1 FHIR Pros and Cons describes some of the tradeoffs in selecting FHIR.

FHIR Pros	FHIR Cons
• Published standard (by HL7) • Fast and easy to implement • Libraries freely available (reference implementation) • Standard is free for use • Support for needed technologies: JSON, XML, REST, etc. • Many resources already defined, including one for each required data type • Each resource can be easily extended with additional data fields • Extensions can be published to the broader community	• Currently the specification is in "Draft Status for Trial Use" • Full specification likely in 2015

Table 5-1 FHIR Pros and Cons

5.2. Database Management System Files

The VistA Exchange is designed to aggregate, cache, and process data from multiple VA, DoD and external partner sources for creation of an enterprise patient record. VistA Exchange will provide the patient record as an enterprise data service for transactional point of care applications. The eHMP system will include a transactional document database, search engine, and support for multiple data models and output formats. VistA Exchange 'processing' of cached data will include, but is not limited to, search indexing and knowledge enrichment, such as mapping to National Standard ontologies. VistA Exchange will aggregate all common healthcare-related document types and formats, including JSON, XML, delimited American Standard Code for Information Interchange (ASCII) (HL7 2.x messages), Resource Description Framework (RDF), Digital Imaging and Communications in Medicine (DICOM), Portable Document Formats (PDFs), still images, sound, video, and other multimedia files.

The VistA Exchange Core software solution must include an integrated document store database. Relational database software will not be considered. Table 5-2 Required Capabilities and Characteristics describes the additional requirements for the core software solution.

Transactional Data Layer Requirements	Schema agnostic ingest and storage of structured, unstructured and hybrid document types
	Ingested documents will be serialized to an XML format for processing in the VistA Exchange cache
	The document store will be programmatically addressable using World Wide Web Consortium (W3C) compliant processing and query languages for XML
	Support for rollback of multi-object database commits
	Document level concurrency control with deadlock resolution
	Both 'hard' and 'soft' journaling (hard journaling – transaction journal entry written to disk before commit, or soft journaling – transaction journal entry written in memory only before commit)
	Transactional data layer must be generally ACID (Atomicity, Consistency, Isolation, Durability) compliant, in addition to the above specific atomicity, concurrency and durability requirements

	Minimum 1PB (petabyte) horizontal scalability on commodity hardware
	Support for local disk failover, as well as both local and remote inter-cluster full database replication
	Role-based document level access control with support for both intersected and joined multi-role access resolutions
Search and Query Support Requirements	Non-schema dependent, structure-aware, full text search with word-level inverted indexing for all data in an eVPR
	Re-indexing will be non-transactional blocking during eVPR component refresh or new queries
	Multi-dimensional complex query support for human composed queries with simultaneous inclusion of word position, semantic association, range, time, provenance and hierarchical relationships
	Sub-second response time for complex queries across a single eVPR.
	Output formats must include XML, HTML and JSON

Table 5-2 Required Capabilities and Characteristics

5.3. Non-Database Management System Files

There are no non-database system files.

6. Detailed Design

6.1. Hardware Detailed Design

The current design consists of multiple physical servers running ESXi that will host all Virtual Machines (VMs) for the system. The Load Balancer, Web UI and API servers will have 2 VPUs, 4 GBs RAM and 40 GBs of hard drive space total. The JDS Server will have 4 VPUs, 16 GBs RAM and 200 GBs of hard drive space. The Synchronization Server will have 2 VPUs, 4 GBs RAM and 60 GBs of hard drive space. All of these servers will be virtual and their resource allocations could change based on what is found with continued load testing.

The following table describes the Detail VM Requirements in Production environment

Production									
Detail VM Requirements									
VM	RAM (GB)	C:\ for OS	Space* (GB)	CPUs	OS	VM Description/Use/DNS Required	Comments	Replicated Storage	
1	16	65	0	4	RHEL 6.5		Apache frontend load balancer for webapi servers cluster	32	* Per Mat Will, 65GB for OS / No additional partition
2	16	65	0	4	RHEL 6.5		Apache frontend load balancer for webapi servers cluster	32	
3	16	65	0	4	RHEL 6.5		Rest service cluster	32	
4	16	65	0	4	RHEL 6.5		Rest service cluster	32	
5	16	65	0	4	RHEL 6.5		UI Service Cluster	32	
6	16	65	0	4	RHEL 6.5		UI Service Cluster	32	
7	16	65	0	4	RHEL 6.5		Message handler service for vista 01	32	
8	16	65	0	4	RHEL 6.5		Message handler service vista 02	32	
9	16	65	0	4	RHEL 6.5		Queue service cluster	64	
10	16	65	0	4	RHEL 6.5		Queue service cluster	64	
11	32	65	5935	8	RHEL 6.5		Database server cluster	6000	* Per Mat Will, 65GB for OS / Balance as additional partition
12	32	65	5935	8	RHEL 6.5		Database server cluster	6000	
13	16	65	67	4	RHEL 6.5		Centralized logging	132	
14	16	65	67	4	RHEL 6.5		Monitoring, performance metrics, and status checks	132	
Total	256	910	12004	64				12648	
*Win VM: Minimum Storage of 60 GB; RHEL VM: Minimum Storage of 30 GB									

Table 6-1– Production Detail VM requirements

The following table describes the Detail VM Requirements in Pre-Production environment

Pre-Production							
Detail VM Requirements							
VM	RAM (GB)	C:\ for OS	Space* (GB)	CPUs	OS	VM Description/Use/DNS Required	Comments
1	16	65	0	4	RHEL 6.5		Apache frontend load balancer for webapi servers cluster
2	16	65	0	4	RHEL 6.5		Apache frontend load balancer for webapi servers cluster
3	16	65	0	4	RHEL 6.5		Rest service cluster
4	16	65	0	4	RHEL 6.5		Rest service cluster
5	16	65	0	4	RHEL 6.5		UI Service Cluster
6	16	65	0	4	RHEL 6.5		UI Service Cluster
7	16	65	0	4	RHEL 6.5		Message handler service for vista 01
8	16	65	0	4	RHEL 6.5		Message handler service vista 02
9	16	65	0	4	RHEL 6.5		Queue service cluster
10	16	65	0	4	RHEL 6.5		Queue service cluster
11	32	65	5935	8	RHEL 6.5		Database server cluster
12	32	65	5935	8	RHEL 6.5		Database server cluster
13	16	65	67	4	RHEL 6.5		Centralized logging
14	16	65	67	4	RHEL 6.5		Monitoring, performance metrics, and status checks
Total	256	910	12004	64			
*Win VM: Minimum Storage of 60 GB; RHEL VM: Minimum Storage of 30 GB							

Table 6-2 – Pre-Production Detail VM Requirements

The following table describes the Detail VM Requirements for Shared Servers

Shared Servers								
Detail VM Requirements								
VM	RAM (GB)	C:\ for OS	Space* (GB)	CPUs	OS	VM Description/Use/DNS Required	Comments	
1	8	65	67	4	RHEL 6.3		Encrypted CM repository for binaries and configuration files	*60 GB for OS / No Additional partition *Update RHEL version
2	8	60	0	4	RHEL 6.3		Server from which deployments are executed	
Total	16	125	67	8				
*Win VM: Minimum Storage of 160 GB; RHEL VM: Minimum Storage of 30 GB								

Table 6-3 – Shared Servers Detail VM Requirements

The following table describes the Detail VM Requirements for Extra Servers

Extra Servers							
Detail VM Requirements							
VM	RAM (GB)	C:\ for OS	Space* (GB)	CPUs	OS	VM Description/Use/DNS Required	Comments
1	8	65	67	2			
2	8	65	67	2			
Total	16	130	134	4			
*Win VM: Minimum Storage of 60 GB; RHEL VM: Minimum Storage of 30 GB							

VM	RAM (GB)	C:\ for OS	Space* (GB)	CPUs
32	544	2075	24209	140

Table 6-4 – Extra Servers Detail VM Requirements

6.2. Software Detailed Design

Figure 6-1 eHMP System Architecture depicts the various components of the eHMP system architecture.

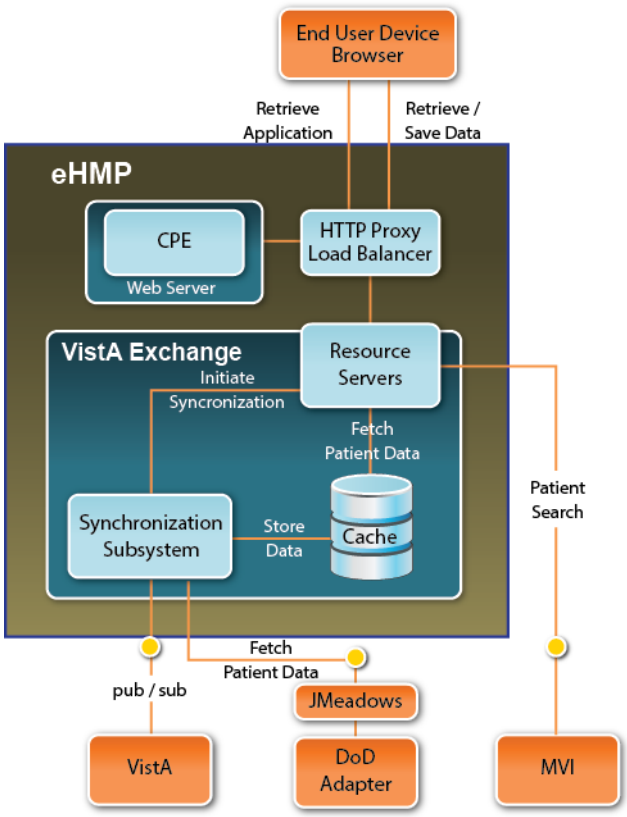


Figure 6-1 eHMP System Architecture

6.2.1. eHMP Software Development Kit (SDK)

With release 1.1, eHMP introduces a web application based upon a custom software development kit. The SDK is explained in this section.

Goal of the web application architecture

The VistA Core/eHMP program has a functional goal to create a web application to be used as point-of-care clinical application that contains, at a minimum, the following functionality:

- Meet criteria for meaningful use "ONC 2014 EHR certification" (PSI 5-PSI 8)
- JLV functionality of patient record, showing several domains at once (PSI 4)
- Advanced text search across patient record (PSI 4-5)
- Medication Review (PSI 4-5)
- Ability to document a clinical encounter (PSI 5-8)
- Ability to edit patient record (add allergies …) (PSI 5-PSI 8)
- Ability to place orders (PSI 5-PSI 8)
- Ability to recommend interventions based upon patient record derived from clinical practices (PSI 5-PSI 8)

In addition, there are several non-functional criteria driving the architecture and development approach:

- In order to achieve meaningful use criteria by September 2015, we will need to create functionality across several teams in parallel. The collaboration across these teams will vary from colocation to remote, but are part of the same Release Train [redacted] to teams with infrequent communication working from different projects/program/contracts. Therefore, there is a need for guidelines on how to add new functionality that is extremely decoupled from other functionality being added, but while staying consistent with functional and architectural approaches in order to build a single application rather than a "patchwork quilt" of an application. Further, the artifacts generated from different development teams must be safe from conflicting action which can be caused by global variables and global namespaces associated with immature JavaScript development. These artifacts are referred to as 'applets' and support the listed principles (must be able to develop applets in isolation without applet-to-applet dependencies or by modifying the underlying framework, but must provide guidelines on how to stay consistent).
- Incremental functionality must be testable in a manner to provide fast, consistent feedback to the development team. (Must be able to test applets in isolation. Testing must be automated.)
- The certification process for VA applications sets a high bar, and includes ensuring 508 compliance, standard look and feel toward usability, and technical standardization. To ensure high quality releases through the certification process, it will be important to set restrictions and guidelines toward adding additional functionality. (Common UI controls will be selected and will be used throughout development.)
- It is expected that a group of teams will develop a set of functionality and then move on to another area. Teams will need to be fluid in moving to different functional areas of the application. To ensure that teams can fluidly work in different areas of the application, it will be important to standardized the technology and development approach - it is not feasible for each team to make their own technical choices in how to add incremental functionality to the web application. (A set of technologies and patterns will be selected and identified as standard.)
- To add non-trivial incremental functionality, modifications will need to be made to the "front-end" (web application) as well as the "back-end" (resource server). Therefore, a team will need the skills to develop holistically, rather than being organized around specific skill sets. (The use of node.js will assure the use of the same technology on both the client and the server.)
- The application is expected to be a vital part of the VA's ecosystem for a long time. Therefore, it must provide a mechanism to add incremental functionality and have an easy deploy strategy. (The use of applets allows for teams to develop new functionality in isolation and have their own release cycle.)
- Although many teams will develop applications, as the program moves toward a sustainment model, maintenance will move to a much smaller team. To ensure that a small number of developers are able to maintain the application, it will be important to standardized the technology and development approach - it is not feasible for each team to make their own technical choices in how to add incremental functionality to the web application. (A set of technologies and patterns will be selected and identified as standard in order to reduce the number of technologies required to support the application.)
- The application will be used as the primary UI for many users, for many hours a day. A highly responsive (fast!) application is vital to garner user satisfaction. The architecture takes into account the minimization of perceived latency. (Putting the behavior logic in the client rather than the server will provide a more responsive (fast) UI.)
- The application has a complex relationship with other parts of the VA. It will be important to provide capability to fine-tune deployment for maximum scalability. (Favor applications/resources decoupled; favor micro services rather than single war file deployment.)

SDK to help to achieve goals

The eHMP team will utilize a SDK to make development more efficient.
The SDK is comprised of:

- Application Development Kit (ADK) to drive development of the client-side web application
- Resource Development Kit (RDK) to drive the development of service-side resources (web services) to support the web application

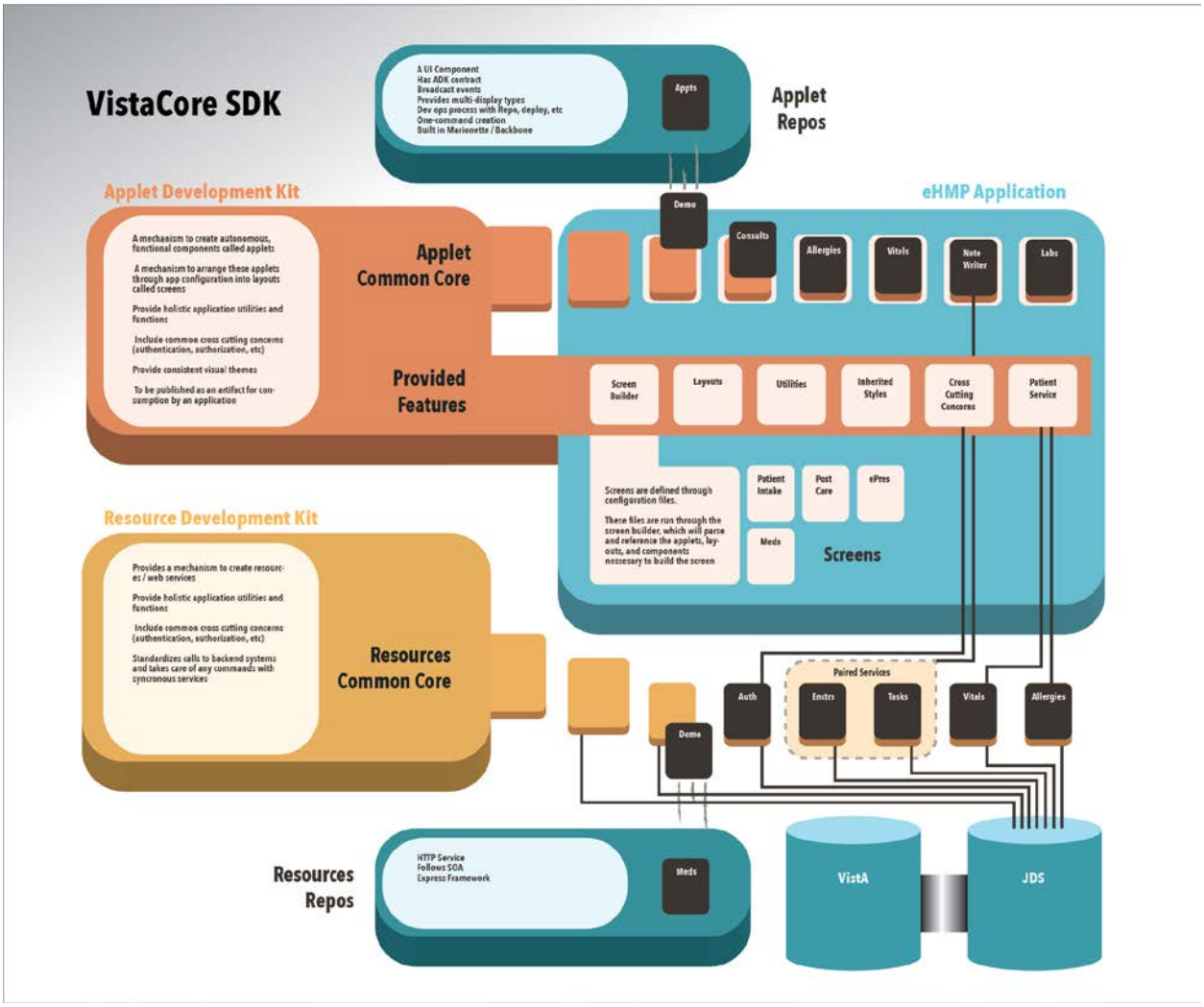


Figure 6-2 – VistACore SDK

ADK Features

Anatomy

The ADK breakdown:
An **applet** is a single unit of visual functionality. Examples of applets include a list of allergies, an ability to enter a new allergy, ability to place a new order.

A **screen** is a defined collection of applets and UI components comprising the application chrome.

A **layout** is an abstract arrangement of applets. A screen typically chooses from a set of ADK pre-defined layouts.

An **application** is a collection of screens.
Example of a screen utilizing a 3x3 layout to display 9 applets:

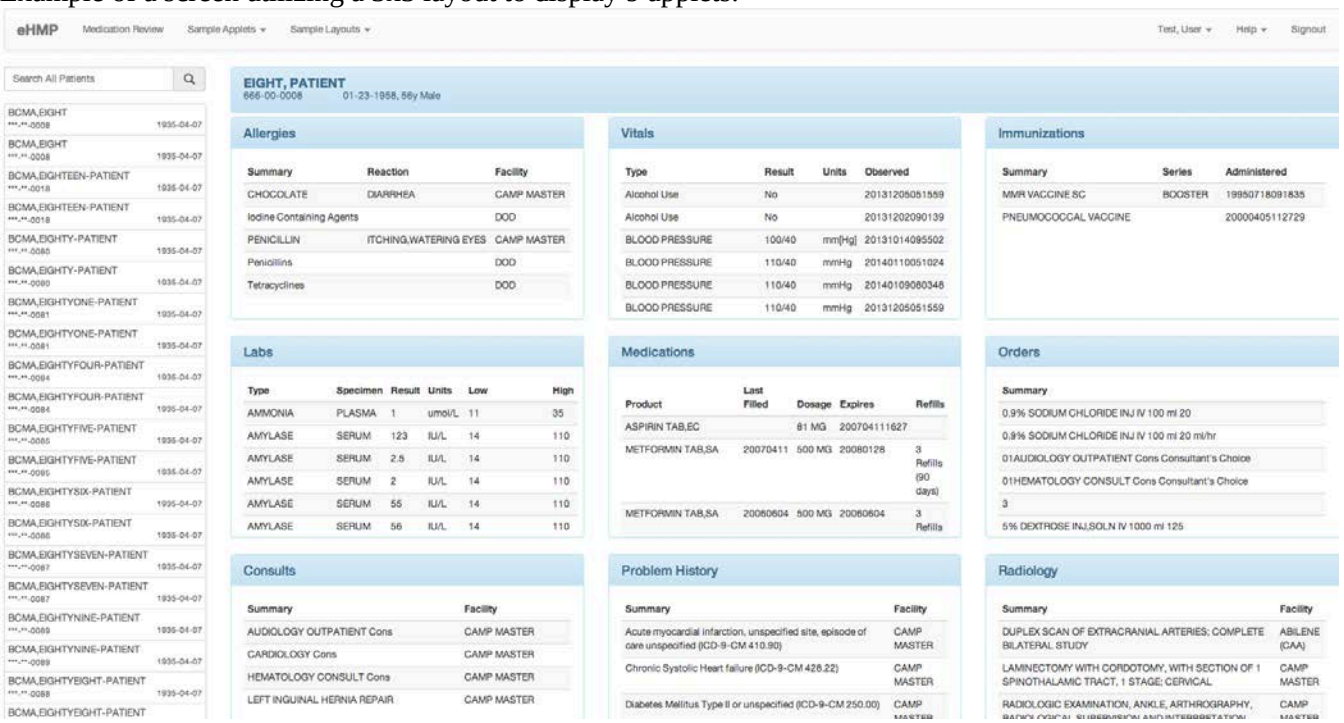


Figure 6-3 – Example of a 3x3 screen layout

Roles

Applets are created by applet developers. Applet developers can often develop applets without consideration to the screens in which the applet will run. Applet developers utilize JavaScript and Marionette to develop applets.
Screen designers choose how to arrange a set of applets. Screen designers use configuration files.

Technical Information

SDK built using JavaScript, backbone.js, marionette, bootstrap
Resulting application is a SPA, static HTML served from web server (not dynamic web pages such as JSP, PHP, etc. No J2EE web server)

ADK responsibilities for applet developers

- Provide mechanism for applet developers to discover the current patient
- Provide mechanism for applet developers to fetch patient data from VistA Exchange, can use canonical model (based on VPR) or provide custom "view" model
- Provide mechanism for applet developers to bind data to backbone views, provide templating
- Provide mechanism for applet developers to choose preselected display paradigms (tableview, etc.) and UI controls. UI style set by application
- Applet developers create three different "size" views to support responsive/adaptive design
- CI pipeline/DevOps for "compiling" applet, packing, running tests, publishing to artifact repository. This produces a CM'ed version of the applet
- The applet developer is responsible for producing a marionette "view." This often will fall into patterns as below:
- HTML template for a row using handlebars

- Creation of backbone view-model
- Registering these with ADK
- Show example for allergy template, both short and complex version

ADK responsibilities for screen designers

- Provide mechanism for screen designers to create a screen, choose from predefined layouts, and assign applets to regions

ADK responsibilities for application designer

- SDK provides the runtime UI shell for the application. Displays current patient, current user, provides navigation
- Provide mechanism for the application designer to choose the screens that are part of the application
- CI pipeline/DevOps to pull applets/screen configurations from artifact repository, compiling, packing, testing, and publishing to artifact repository. This produces a CM'ed version of the web application (which includes specific version of applets).

RDK Features

Anatomy

A resource is implemented as a single web service (allergies, or "save allergies"). (In JAX-RS, a single method that gets the @GET annotation)
A resource family is a collection of resources that are generally deployed together. (In JAX-RS, a class)
A resource server is a deployable unit, including a set of resource families and specific configuration. (In JAX-RS, a war file).

Roles

A resource developer develops the resources and resource families in JavaScript.
A deployment engineer chooses how to deploy a set of resource families as a resource server, and develops in JavaScript.

Technical Information

RDK developed using JavaScript, express.js

Deployed to node.js; relies on npm for package management.

RDK Responsibilities for resource developer

- Provide configuration of the resource family
- For each resource, specify relative mount path and method using standard express signature function(request, response, next)
- Specify resource characteristics (sensitive, audit information)
- Request provides resource information about the request: URL, path parameters, query parameters, header, current user, access to RDK utility methods, access to current resource server, configuration information, logger
- Response provides resource ability to send response: status code, json or text body, media information
- RDK provides resources with handles to common external systems, including JDS, VistA(s), SOLR
- CI pipeline/DevOps of resource including compiling, packing, testing, and publishing to artifact repository. This produces a CM'ed version of the resource

RDK responsibilities for deployment engineer when creating a resource server:

- Group resources into a logical deployment unit called a resource server.
- Specify how many processes use cluster/fork
- Deployment engineer chooses which resource family should be registered using: (show code)
- Resource Server deployed behind a reverse proxy (Apache httpd) for load balancing
- Ability to enable authentication, authorization/PEP, audit
- Specify logging rules
- Specify configuration
- Specify resource caching rules
- RDK routes calls to resources based upon URL and media type (content-negotiation)
- Provides health check (both binary and discrete information) based upon each resource registered. Deployment engineer can specify additional health check rules.
- CI pipeline/DevOps to pulling resources from artifact repository, compiling, packing, testing, and publishing to artifact repository. This produces a CM'ed version of the resource server (which includes specific version of resources)

Other responsibilities

RDK also provides:

- Centralized dependency management
- Automatic creation of resource directory

ADK Details

SPA

The implementation of the web application created through the SDK is SPA ("single-page application"), utilizing a set of server-side resources (RESTful web services).
The HTML/JavaScript/CSS for the application is returned as "static HTML" from a web server, specifically Apache httpd. The client-side application utilizes the resource server (web services) to execute various data-driven activities, such as reading the patient record, updating the patient record, documenting a patient encounter, ordering medications, etc.
Note that this pattern is similar to a traditional client-server or native mobile application: the user downloads the application on first use, the application itself contains behavior but not the data, the application reaches back out to a server for read/write activities.
The term "single-page application" can be misleading in that it may suggest that there is a single "view" for the entire application. The term "page" refers to the handling of the lifecycle of the page within the browser. Rather than recycling the entire page, elements on the DOM are modified as needed.
Although there are differing approaches for implementation of a SPA, ours will be to have the static assets (HTML/JavaScript/CSS) decoupled from the data-centric resource services on which the rely. The static assets will be available from a web server (like Apache) independent from the resource server (running within JVM and/or node.js).

Development Lifecycle

(applet developer) -> (source code repo) -> (build process) -> artifact repository
(screen designer) -> (application source code repo) -> (build process) (pulls from applet artifact repository) -> artifact repository (application designer) ->
deployment process deploys local source code (applets, screens, application) to private cloud (local VM's)
deploys official CM'ed application from artifact repository to test/integration environment
deploys specified official CM'ed application to preproduction / production environment

Applet Development

The majority of web developers working within the creation of the eHMP UI will spend their time developing applets. The applet is the incremental user functionality. An applet is a set of HTML/CSS/JavaScript and runs within the ADK.

Applet development begins with using a DevOps service/offering to initialize the applet (currently all applet development will be done in a single eHMP-UI App repo). The result of this service includes: a new Git repository for the applet, seeded based upon an applet template; CI (Jenkins) jobs responsible for reacting to check-in, compiling the HTML/CSS/JavaScript, running unit and acceptance tests, and publishing the resulting artifact to an artifact repository.

The resulting artifact of an applet will likely include two versions: a development version optimized for debugging (not minified), and a version optimized for production (minified).

The goal of the ADK is to provide the guidelines, constraints, and technology to ensure that teams can efficiently create incremental functionality *without* requiring unsalable close collaboration between development teams. Yes, collaboration is a good thing. However, many teams modifying the same code or having to coordinate routine changes is not scalable to an ecosystem of many parallel functional development areas.

The ADK includes:

- Provide mechanism for applet developers to discover the current patient
- Provide mechanism for applet developers to fetch patient data from VistA exchange, can use canonical model (based on VPR) or provide custom "view" model
- Provide mechanism for applet developers to bind data to backbone views, provide templating
- Provide mechanism for applet developers to choose from preselected display paradigms (tableview, etc.) and UI controls. UI style set by application
- Applet developers create three different "size" views to support responsive/adaptive design
- CI pipeline/DevOps for "compiling" applet, packing, running tests, publishing to artifact repository. This produces a CM'ed version of the applet
- Applet Developer responsible for producing an marionette "view." The view returned by the applet can be any type of marionette view, CollectionView, CompositeView, ItemView, or LayoutView. A CompositeView and LayoutView can include additional regions that applet developers have control of to load other sub views. This often will fall into patterns as below:
- HTML template for a row using handlebars
- Creation of backbone view-model
- Registering these with ADK
- Example code to create "allergy applet":

applet

```
var dependencies = [
    "backbone",
    "marionette",
    "underscore",
    "ADKApp",
    "hbs!main/applets/allergies/table",
    "hbs!main/applets/allergies/row",
    "api/AllergyFactory"
];
define(dependencies, exportAMD);
function exportAMD(Backbone, Marionette, _, ADKApp,
AllergyViewTemplate, AllergyItemTemplate, AllergyFactory) {
    var AllergyItemView =
Backbone.Marionette.ItemView.extend({
```

```
15          tagName: "tr",
16          template: AllergyItemTemplate
17      });
18      var AllergyCompositeView =
19  Backbone.Marionette.CompositeView.extend({
20          childView: AllergyItemView,
21          childViewContainer: "tbody",
22          template: AllergyViewTemplate,
23          className: "panel panel-info",
24          collectionEvent: {
25              "sync": "render"
26          }
27      });
28      var AppletLayoutView =
29  Backbone.Marionette.LayoutView.extend({
30          template: _.template('<div id="applet-main"></div>'),
31          regions: {
32              appletMain: "#applet-main"
33          }
34      });
35      var rootView;
36
37      var applet = {
38          id: "allergies",
39          getRootView: function() {
40              rootView = new AppletLayoutView();
41              return rootView;
42          },
43          onDisplay: function() {
44              var viewModel = {
45                  parse: function(response) {
46                      if (response.reactions) {
47                          response.reaction =
48  response.reactions[0].name;
49                      }
50
51                      return response;
52                  }
53              };
54              var allergies =
55  AllergyFactory.initCollection(patient, viewModel);
56              var allergyCompositeView = new
```

```
57  AllergyCompositeView({
                    collection: allergies
                });
                rootView.appletMain.show(allergyCompositeView);
            }
        };
        return applet;
    }
```

applet template

```
 1   <div class="panel-heading">
 2      <h3 class="panel-title">Allergies</h3>
 3  </div>
 4  <div class="panel-body" id="appletContent">
 5      <table class="table table-condensed table-striped table-
 6  hover">
 7          <thead>
 8              <tr>
 9                  <th>Summary</th>
10                  <th>Reaction</th>
11                  <th>Facility</th>
12              </tr>
13          </thead>
14          <tbody>
15          </tbody>
16      </table>
    </div>
```

row template

```
1  <td>{{summary}}</td><td>{{reaction}}</td><td>{{facilityName}}</
   td>
```

Much of the boilerplate of the applet has been reduced utilizing a ADK convenience method:

simplified applet

```
 1  var dependencies = [
 2      "main/ADK",
 3      "hbs!main/applets/allergies/table",
 4      "hbs!main/applets/allergies/row"
 5  ];
 6  define(dependencies, exportAMD);
 7  function exportAMD(ADK, allergyTableTemplate,
 8  allergyRowTemplate) {
 9      var viewModel = {
10          parse: function(response) {
11              // this is handling of canonical model to view
```

```
12  model
13              if (response.reactions) {
14                  response.reaction =
15  response.reactions[0].name;
16              }
17              return response;
18          }
19      };
20      var appletDefinition = {
21          appletId: "allergies",
22          resource: "allergies",
23          viewModel: viewModel,
24          tableTemplate: allergyTableTemplate,
25          rowTemplate: allergyRowTemplate
        };
        return ADK.createSimpleApplet(appletDefinition);
    }
```

Accessing Patient Record

Applet developers can access the patient record using the ADK.PatientRecordService. The fetchCollection method returns a Backbone Collection object.

```
var fetchOptions = {};
fetchOptions.patient = adkApp.session.patient;  //optional,
defaults to current patient context
fetchOptions.resourceTitle = 'patient-record-med';  //resource
title from resource directory
fetchOptions.viewModel = viewModel;  //optional override of
default viewModel
fetchOptions.criteria = criteria;  //optional criteria object
gets converted to query string parameters on resource call
medicationCollection =
ADK.PatientRecordService.fetchCollection(fetchOptions);
```

RDK Details

Resource Server

A resource server represents the information and processing that occurs in the server tier of the web application. The "resources" include: the patients clinical record, an order, a user, the current patient, and a list of allergens known to the VA. The resource server includes web services that allow for the retrieval of a patient's clinical record, the edit of a patient's record, and the ability to place an order, etc.
The existing patient record resource server is written as a "microservice". The implementation is in Java, using a light weight embedded HTTP listener. The approach of the "microservice" allows for discrete instances to be independently deployed, optimized, and load balanced. This is contrary to an approach of

deploying several logical "services" as a single WAR file and deployed to a clustered app server. With the "single WAR file deployed to cluster" individual optimization becomes more challenging.
Additional resource web services will be written in JavaScript using node.js. The concept of a microservice will be used. Each additional web service is to be written as an NPM (node package manager) module. The modules (implementation of the web service) are then packaged in a logically decomposed microservice, resulting in a deployable, runnable artifact.
As a team developing an applet recognizes that there is a need for an additional resource/web service (suppose there is need to discoverer the list of clinics within a VA facility), the team then creates an additional resource module.
Like the applet, the resource module development begins with the usage of the DevOps service/offering to initialize the module. The result of this service includes: a new Git repository for the module, seeded and based upon a module template; CI (Jenkins) jobs responsible for reacting to check-in, running unit and acceptance tests, and publishing the resulting artifact to an artifact repository.
All new resources services are to be written in node.js. Expect to migrate existing patient record web service to node.js.

RDK

The cross-cutting features provided by the resource development kit (RDK) include:

- authentication
- authorization
- web service / resource server decomposition, registration
- logging
- audit
- health check
- cluster/fork
- caching
- content negotiation
- data formatters
- handle to JDS
- handle to vista RPC's

Resource Developer Responsibility

1) Create a method to represent a single endpoint. This method supports the express format of function(request, response, next).
Common items to access on request include:

- param: includes any path parameters, query parameters, http header parameters. See param documentation.
- app.config: application configuration
- logger: access to RDK logger
- audit: if this resource needs to provide any custom audit information, resource developer can specify by adding to req.audit

The resource must send a response using response.send, typically response.send(myObject). See response documentation.
2) Create a resource configuration. This is invoked by the resource server during startup. This is done by specifying an exported method of getResourceConfig. The resource config is an array of objects, with each object containing the following:

```
exports.getResourceConfig = function() {
    return [{
        // optional: name of resource
        name: '',
```

```
        // optional: relative path for this resource
        path: '',
        // specify http action (get, put, post, delete, head),
followed by resource method with pattern function(req, res, next)
        get: performPatientSearch,
        // optional: provide method for health check, which
returns true if healthy or false if not healthy
        healthcheck: function() {
            return true;
        },
        // provide a list of common middleware/interceptors
        // note that method for specifying these likely to change
in next internal release
        interceptors: ['audit', 'authentication', 'pep',
'metrics', 'operationalDataCheck']
    }];
};
```

6.2.2. Web Server

The role of the web server is to provide the static files that represent the web application and present the application to the user. These static files contain the HTML, JavaScript, and CSS to drive the behavior of the application. No data is provided from this web server.

The web server is implemented with Apache Web Server [1].

6.2.3. HTTP Proxy

The role of the HTTP Proxy is the following:

- act as a network buffer between the public client and the internal web service API instances
- perform Uniform Resource Locator (URL) routing to web service API instances
- load balance calls to appropriate web service API instances

The HTTP proxy is implemented with Apache Web Server.

When an HTTP request is performed, the HTTP Proxy interrogates the request, and based upon the URL, routes it to the appropriate server. These requests include:

- calls from a browser to access the web application static assets
- calls from browser-based web application to access data resources
- calls from any other client to access data resources

[1] Apache Web Server: http://httpd.apache.org

Under Release 1.1, the proxy server has the additional responsibility of validating the client certificate presented from the client application. This will be used to ensure that the user has a PIV card.

6.2.4. Web Service API

The VistA Exchange Web Service API allows for a consumer to retrieve a patient record for a given patient. The web service API provides a VPR and FHIR web service that allows for the retrieval of patient records using the patient's ICN. Upon request, a request is made to all VistA instances to initiate a patient subscription, if the patient data has not previously been synchronized. This subscription is initiated by invoking the HMP VPR pub/sub RPCs. Within a few seconds, the data is available within the JDS cache. Once it is available, the web service API transforms the data into the FHIR, and returns it to the consumer. Table 6.1 Installed Software depicts the technology stack for the Web Service API.

Technology	Version	Role
Java Runtime Environment	JRE 1.7.0_25	JVM
Jersey	1.17.1	JAX-RS implementation
Jetty	9.0.5	HTTP Web Listener

Table 6-5 Installed Software

6.2.4.1. Implementation of the Web Service

Under release 1.0,the web services are developed using a set of Java classes using Jersey (JAX-RS). These web services are compiled into an executable JAR. This executable JAR file contains an embedded web server (Jetty). This allows for the JAR to be executed rather than deployed to a traditional app server. To scale the web service, an additional microprocess is executed. The microprocesses are controlled via a process watcher to ensure that they are started, and restarted should they shutdown.

Under release 1.1, the Java web services will be deprecated in favor of the SDK implementation of a resource servers. These SDK-based resource servers expose the data to be used by the web application, ranging from patient data, lookup lists of allergens, and interventions recommended by a clinical decision support system.

If the scaling characteristics between the Patient Resource (which has a patient correlation/MVI dependency) and the Patient Record Resource (which has a cache and Patient Sync Subsystem dependency) require different scaling concerns, these can be deployed to separate virtual machines. This will ensure that one resource does not negatively affect the other from a performance and scalability perspective.

The SDK resources utilize node.js and are written in JavaScript.

Technology	Version	Role
Node.js	TBD	JavaScript runtime engine
Express.js	TBD	Node.js module used to handle routing of HTTP calls

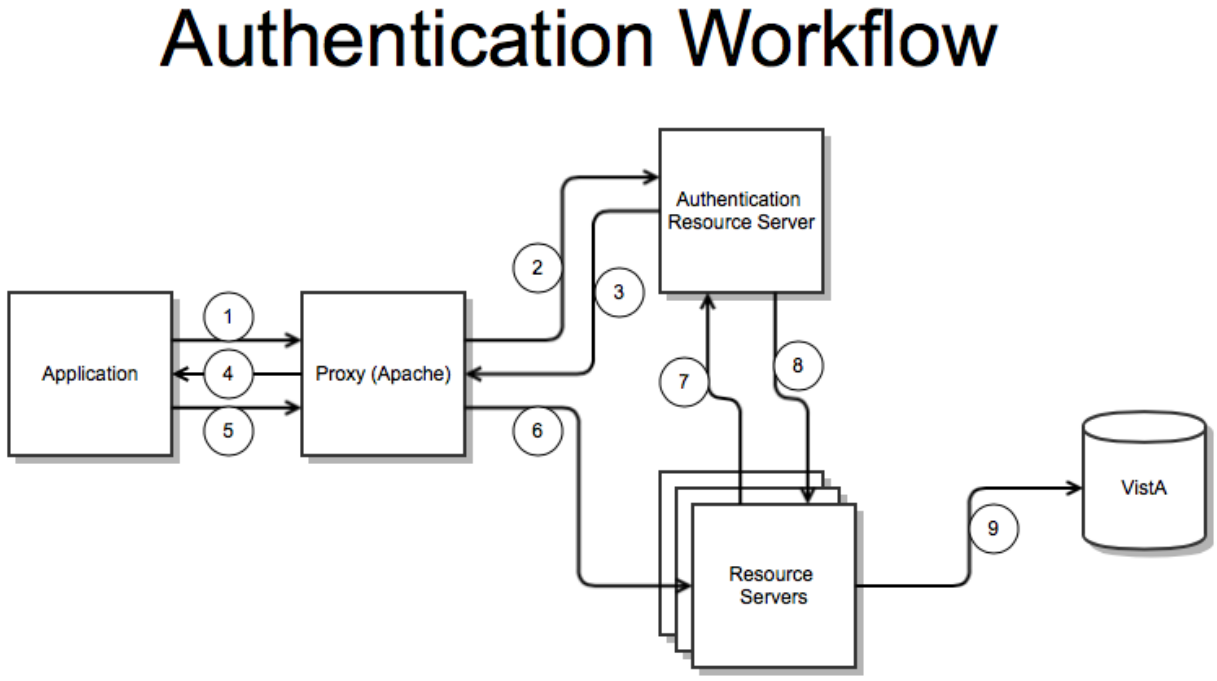
6.2.4.2. Authentication

User authentication includes several items.

First, the user can utilize their access and verify code with the selection of a site (VistA instance) in order to log on. For Release 1.0, the web services accept an access/verify code plus site as part of the HTTP

header consistent with HTTP Basic Authentication. The web service validates this against the VistA instance using an RPC call. The HTTP header is passed on all calls.

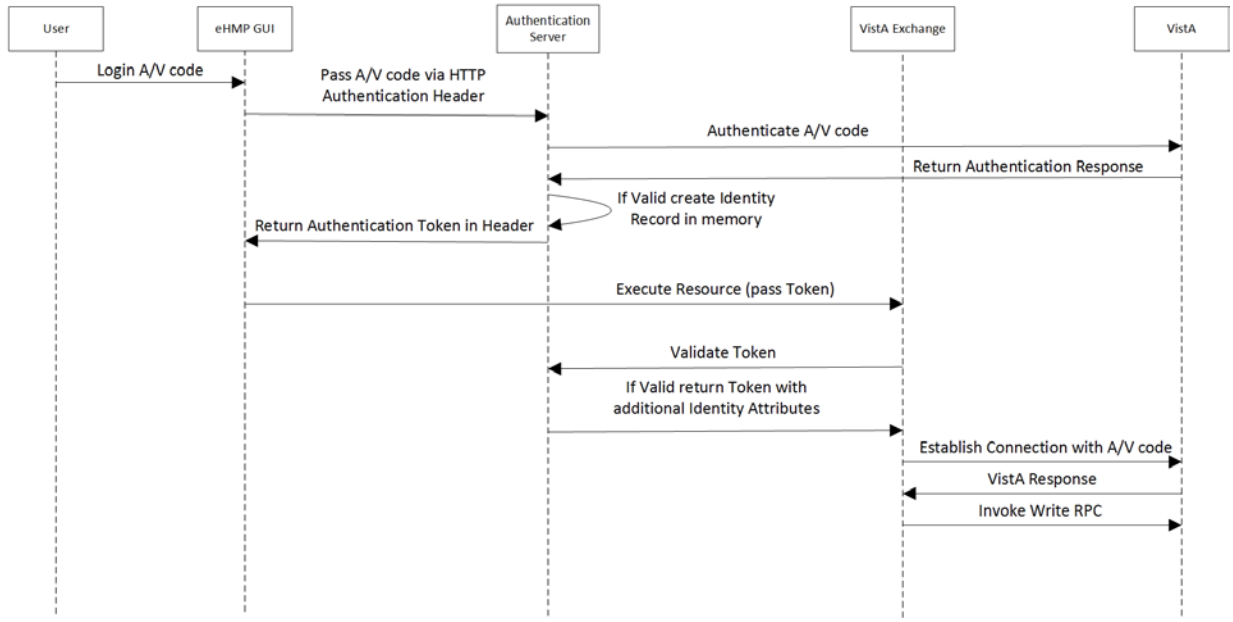
For release 1.1, the introduction of a token-based scheme works as follows.



1: Application presents access/verify code (via HTTP Basic authentication header) to authentication resource
2: Proxy (Apache) routes to authentication resource server. Authentication resource validates the access/verify code against VistA. If valid, creates storage (assume memory) for this session keyed by "token", with values for any user information that we need, including access/verify code, user's name, and anything we need for authorization downstream
3: Authentication Resource Server returns status 200 with token in a header value
4: Proxy returns to application. ADK holds onto token
5: Application attempts to access patient data by performing call to resource server. Call sent to proxy. SDK attaches token to header.
6: Proxy routes call to appropriate resource server
7: Resource Server authentication interceptor validates token by making call to resource server endpoint (passing token).
8: Authentication server returns indicator to state that token is valid/not valid. If valid, also returns info known about user, such as access/verify code
9: Resource Server establishes connection to VistA using access/verify code

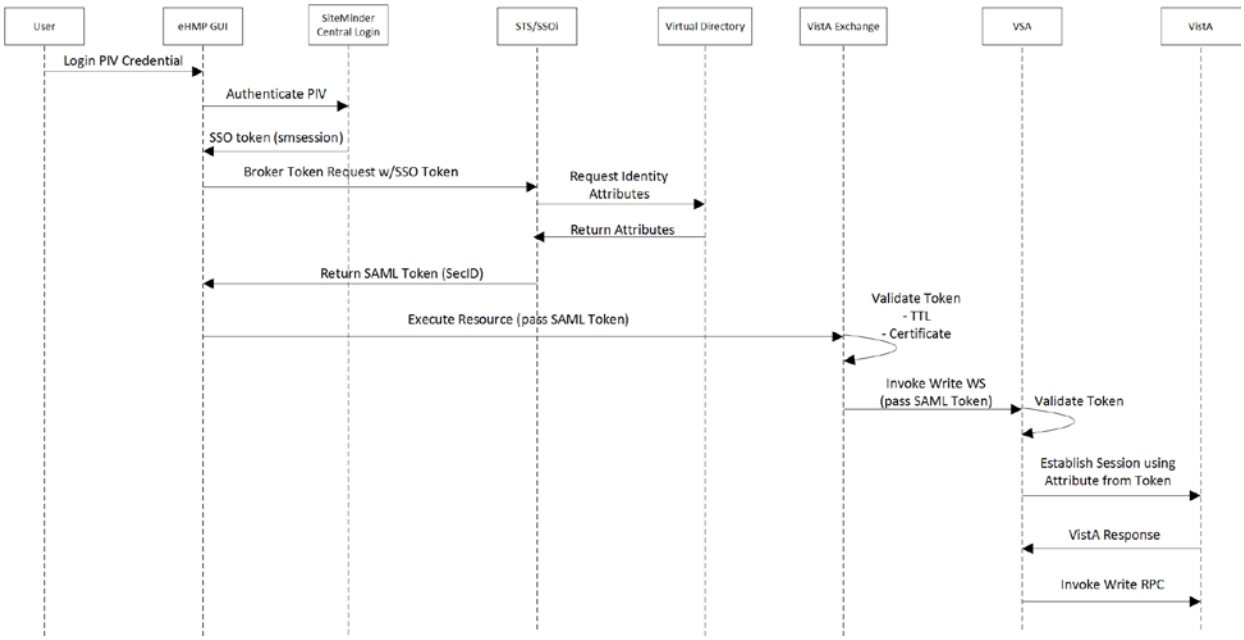
An alternate view of this workflow is listed below:

Interim Solution Access/Verify Code



A longer term approach relies upon IAM SSOi to provide mapping from the user's active directory identity to their VistA identity.

September 2015 Enterprise Solution



In addition, multi-factor user authentication will be employed as Personal Identity and Verification (PIV) cards are implemented throughout the VA staff/user population. It is anticipated that a single-factor user id and password may be required at the initial launch of the eHMP system until the entire VA staff/user population are issued PIV cards. Authentication of the web service API will be driven by the user's access

and verify code. When the user navigates to the web application, the web application static content (HTML/JavaScript/CSS) is accessed from the web server. The access to this content is not restricted to an authenticated user because, at this point, there is no data.

6.2.4.3. Access Control

Once a user is identified and authenticated, his or her access to system resources and application functionality will be based upon role-based access controls. When the web application attempts to access a protected resource such as patient data, the web service API's Policy Enforcement Point (PEP) gathers the information about the request (the user, the subject, the resource, and the action) and provides this information to the Policy Decision Point (PDP) via an Extensible Access Control Markup Language (XACML) request. Note that the PDP is implemented using Axiomatics.

The PDP then checks the request against registered rules and provides a permit or deny response, potentially with a set of obligations that the PEP must perform to allow access. The rules that will be implemented as part of this increment are to ensure that the user has a valid role such as a "CPRS User" or "CAPRI User" (flag within VistA). A "CPRS User" will have full read/write capabilities, while a "CAPRI User" will have read only capabilities. A check is also made to determine if the patient is identified as a "sensitive" patient at any of the VistA sites. If the patient is flagged as sensitive, then the obligation to "break the glass" is returned to the web application, prompting the user that they will be audited if they access the data. If the patient is flagged as sensitive because a user is attempting to access their own record, access to the patient record is denied without the option to "break the glass".

A "CPRS User" is defined as a VistA user that has at least one CPRS TAB ACCESS setting. Vista user management includes two (2) options for CPRS TAB ACCESS, CPRS UI "core" tabs (COR) and Reports tab (RPT). Access will be denied to any resource requiring "CPRS User" access if a user does not have either of these settings enabled. The Vista RPC used by CPRS to validate this setting is ORWU USERINFO. The RPC response includes the COR and RPT tab settings in fields 22 and 23 of the response as illustrated in figure 6-4 CPRS Tab Access.

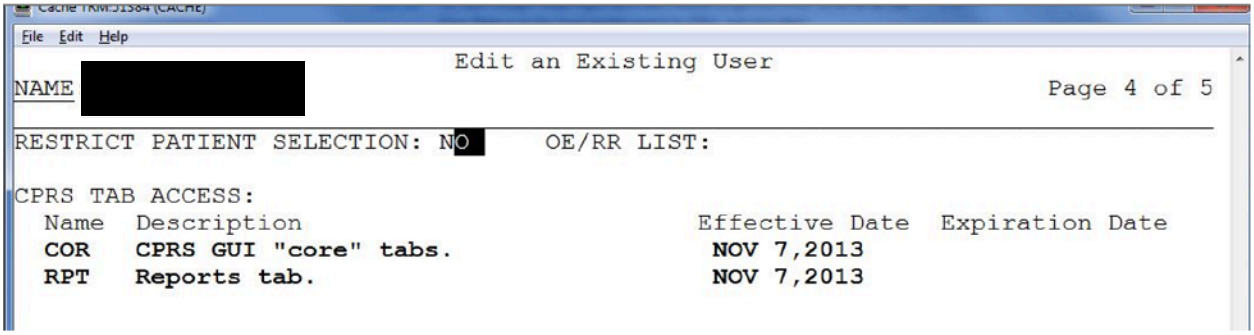


Figure 6-4 CPRS Tab Access

Patient sensitivity rules are based on a number of attributes that will continue to grow as new patient access control policies are defined and implemented. Many of these attributes are not centralized and exist in each VistA host. In order to fully leverage the benefit of an externalized access control solution and provide centralized access control policies for centralized patient records, all user and resource (e.g., patient) attributes required for defining access control rules and policies will need to be centralized.

The current HMP solution and the initial implementation of Axiomatics with the FHIR API will be based on access control attributes retrieved from a single VistA host using the existing patient sensitivity RPC interface. The VistA RPC used for patient sensitivity checks is VPRCRPC RPC with a command parameter of "isPatientSensitive." Figure 6-5 Patient Sensitivity Command Response depicts the

command response, which includes attributes of “sensitive”, “logAccess”, “mayAccess”, and “text” as defined below and shown in the following figure:

- sensitive – When exists, indicates that the patient is sensitive
- logAccess – When true, indicates that patient access auditing should occur
- mayAccess – When true, indicates that the user may access the sensitive patient with obligation
- text – Includes the obligation prompt text for the consuming application

Patient Sensitivity RPC Response


```
{
  "isPatientSensitive":{
   "sensitive":{
     "dfn":20,
     "logAccess":true,
     "mayAccess":true,
     "text":"\r\n***RESTRICTED RECORD***\r\n* * * *
```

The first will be related to sensitive patient access, in which the user chooses to "break the glass". Once a user has chosen to access a sensitive patient and has chosen to accept that they will be audited for these actions, the RDK-based resource server ensures that this access is audited. This audit information is written back to VistA via an existing RPC.

Second, all calls to the resource server will be recorded and stored within VistA Exchange. It is expected in a future increment for this data to be published to a central authority to maintain.

Third, each patient access is audited back to VistA using the Audit RPC call that is available, and VistA implemented by the VPR RPCs. This ensures that an audit trace is stored to VistA for each patient access that a given user performs, regardless of what actual data is viewed by that user.

6.2.4.5. Patient Search and Selection

Patient Search and Selection will be accomplished by locating a given patient within the JDS cache or if not found in the cache, then querying the primary VistA site. Subsequent releases will also perform searches against the VA MVI. There are three scenarios that will be supported.

First, the eHMP UI will perform a search against the JDS cache when an end user performs a patient search. This could be done by entering patient traits or identifiers. The search options that will be supported for patient traits are Last Name, First Name, and Social Security Number, consistent with VHA Business Requirements. The UI controls will require a minimum of three characters to be entered for performing trait-based searches. Selection of a patient using an identifier will support either the Patient ICN or the Patient DFN and Site. If a complete patient record is found in the JDS cache, then no additional search or synchronization calls will be made to the Primary VistA site. If the patient is not found in the cache (or if only a minimal demographic record is found), then the UI will retrieve the full demographic and clinical record using the Patient Sync Subsystem described below.

Second, the FHIR API is called to perform a patient search against the JDS cache when an external application or process wants to perform a patient search. This could be done by passing in patient traits or identifiers. The search options that will be supported for patient traits are Last Name, First Name, Social Security Number, Date of Birth, and Gender. Partial field entries will not be supported for performing trait-based searches. Selection of a patient using an identifier will support either the Patient ICN or the Patient DFN and Site. If a complete patient record is found in the JDS Cache, then no additional search or synchronization calls will be made to the primary VistA site. If the patient is not found in the cache (or if only a minimal demographic record is found), then the system will retrieve the full demographic and clinical record using the Patient Sync Subsystem described below.

Third, the FHIR API is called to search for Clinical Data (for a given domain such as Allergies) by patient using the ICN or FHIR Patient Subject Identifier. In this case, the calling application or process would pass in the exact ICN or FHIR Patient Subject Identifier. If the patient is found in the JDS cache along with the Clinical data, then the FHIR API would retrieve that data directly from the JDS cache. If the patient is not found in the cache, then the FHIR API would initiate a Patient Sync for that patient (using the ICN), and wait for the JDS cache to be populated with the Clinical Data before returning that to the calling application.

In addition to the approaches above for patient search, release 1.1 will add support for doing enterprise search, utilizing the VA's Mater Veteran Index (MVI). This will establish a SOAP connection from VistA Exchange passing the patients traits. Unlike the approaches above, this is intended to be used only for very narrow search results. The implementation will be worked out during the next increment.

6.2.5. Patient Sync Subsystem

The Patient Sync Subsystem is responsible for orchestrating patients being synchronized. The Patient Sync Subsystem provides a java API to request sync and to check if there is a sync being performed for the patient.

When a patient sync is requested for a given patient, the Patient Sync subsystem invokes a VistA RPC to initiate a subscription for the given patient. Within VistA, this retrieves all data for the patient and places into an M-based outbound queue within VistA. The VistA Exchange patient sync subsystem periodically checks for data available from the outbound VistA queue using a polling RPC check. As data becomes available, the patient sync subsystem retrieves the data. The patient sync process then checks the data against the known terminology lookup tables, and applies any of the standards based terms to the data.

The “enriched” data record is then stored into the JDS cache using an HTTP PUT command.
The supported terminology data includes:

- Allergies (UMLS CUI)
- Immunizations (CVX)
- Lab Results (LOINC)
- Medications (RxNorm)
- Notes (LOINC)
- Problems (SNOMED CT)
- Vitals (LOINC)

6.2.6. Patient Correlation Service

The patient correlation service is responsible for performing a patient search and for lookup of patient identifiers (including looking up the patient Data File Number (DFN)/site code for a given enterprise identifier). The patient correlation service is embedded within the web service API and the patient sync subsystem.

Pending assessment from Identity Access Management (IAM) via an existing service request, the Patient Correlation Service will perform real time request/response calls to the Master Veteran Index (MVI) to perform the search and get Corresponding IDs.

6.2.7. Cache

The clinical data is cached in a subsystem called JSON Data Store (JDS), which is an M-based document data store developed originally as part of the Health Management Platform (HMP) program. JDS is hosted within InterSystems Caché. JDS provides an HTTP based API to allow for VistA Exchange to read and update the data. The cached clinical data also gets stored within an instance of SOLR. This is optimized to support full text search.

When the Patient Synch Subsystem receives additional data for a patient, it adds/updates that data to the JDS and SOLR, independent of any end user access to that data. This “near real time” synchronization architecture keeps the cache updated for the set of patients that have been synchronized with one or more VistA host locations.

6.2.8. VistA Access Approach

There are several technology approaches used to integrate with VistA.

First, when the synchronization process initiates a subscription or checks for published events from VistA, it will connect to VistA using a direct connection to the RPC broker. These calls are done within the context of a system proxy account. These calls are responsible for: initiating a subscription from a VistA Exchange instance to a given VistA host for a specific patient; polling to receive published events representing availability of data subscribed patients; pull of operational data, such as lookup lists. This data is stored to the VistA Exchange cache (JDS).

Second, for retrieval of patient data from "remote" VistA hosts, the subscription process performs a request by invoking the CDS web service. The CDS web service then in turn invokes VistA instances to retrieve data for that patient. This data is stored to the VistA Exchange cache (JDS).

Third, under Release 1.1, VistA Exchange will utilize a direct RPC connection for performing writes. These writes will cover the domains of allergies, vitals, and problems. All these writes will be to VistA. This connection to VistA will be performed using the actual users access / verify code – no proxy user account will be used in this scenario.

Forth, under a future release when VSA is available, it is expected to migrate the writes from direct RPC broker connection to utilize VSA. This requires an ability to present a "token" to VSA as provided from IAM SSOi to identify the end user.

6.2.9. Patient Generated Data

VistA Exchange has the ability to pull data in from multiple sources. One of the planned sources of data is Patient Generated Data (PGD). This would allow for PGD to be presented alongside data from clinicians with indicators of the patient data source.

To demonstrate this capability, the Release 1.0 reaches out to a mock implementation of web services that represents the expected to-be state of the VLER DAS PGD instance. The synchronization process translates the patient data to the EDIPI, and performs a request/response fetch of data from the mock data. This data is then stored to the VistA Exchange cache.

7. External Interface Design

The Web Service API is the public API for accessing data within VistA Exchange. This web service API follows the FHIR specification[2]. The detailed interfaced architecture and design will be defined during the development of this increment.
The interfaces that eHMP consumes are Vista, CDS, MVI, and JMEADOWS.

8. Human-Machine Interface

8.1. Summary

As of the date of this document version, human-machine interfaces details are still being defined. There will be an eHMP Web Services delivered at the end of the eHMP Increment 1. The detailed interface

[2] FHIR Specification: http://www hl7.org/implement/standards/fhir/

design rules, inputs, and outputs, and navigation hierarchy are subject to change as additional customer elicitation is held.
In addition the application will have the following engineering standards and principles:

- 8.1.1 – Responsive Design
- 8.1.2 – Adaptive Design
- 8.1.3 – 508 Accessibility
- 8.1.4 – User Centered Design iterations with participation from the VA Human Factors Engineering team

8.2. JLV Functionality

The JLV functionality as included in eHMP will be known as Cover Sheet. Cover Sheet is an instance of a screen as defined in 6.2.1. The Cover Sheet is meant to be a role-based, preset of applets designed for initial patient EHR summary functions. It will blend VA and DoD data to fully represent the patient record. Additionally, there will be user customizable screen/views to accommodate user preference.

8.3. Patient Search

Patient search has been decoupled from the Patient Record in accordance with VA Patient Safety Office requirements.

- Four search types are expected:
  - Global Search – Search is issued against MVI to return patients from across the DoD and VA enterprise. User can search using Name, DOB, and SSN criteria.
  - My Site (All) – Allows for Name, SSN search against current VistA (as defined by the facility identification required at login)
  - My Site (Clinic) – Returns all patients for a selected clinic location with ability to add data range and name filter qualifiers
  - My Site (Ward) – Returns all patients for a selected ward location with ability to add name filter qualifiers
- Patient selection will include a confirmation giving enough patient information further qualify the selection
- Patient selection will elicit sensitivity flags and 'break the glass' scenarios. These flags will be presented as part of the patient selection and will require a confirmation.

8.4. Patient Information:

Will be presented in a vertical and will persist across all patient-centric actions as to always keep the patient in context.

8.4.1 – CWAD – Postings: to include:

- 8.4.1.1 – Allergies
- 8.4.1.2 – Crisis Notes
- 8.4.1.3 – Directives
- 8.4.1.4 - Flags

Patient Record Fields:

- 8.4.2 - Last Name, First Name
- 8.4.4 - SSN (555-55-5555)
- 8.4.5 - DOB (01/01/1900)
- 8.4.6 - Age (80y)
- 8.4.6 - Gender (M)
- 8.4.7 - Provider
- 8.4.8 - Ward (if applicable)
- 8.4.9 - Bed (if applicable)

- 8.4.10 - Primary Care Team
- 8.4.11 - Attending Provider (if applicable)
- 8.4.12 - Inpatient Provider (if applicable)

8.5. HMP Functionality

The following functional areas from HMP will be retained:

- Cover sheet (as described in 8.1)
- Medications Review – and aggregate of patient medications both inpatient and outpatient with date range qualifiers to limit the data set.
- News Feed – A historical view of past events in the medical record. To include Problem List, active medications and current patient tasks.
- Documents – A listing of all document types with date range qualifiers to limit the data set. This set includes but is not limited to:
  - Progress Notes
  - Radiology Exams
  - C-CDAs
  - Discharge / Essentris Notes
  - Surgery Reports
  - Consults
- Text Search – To be known as "Search Record". This function gives deep exposure to the EHR through free-text search, as well as predefined lookups into domains, labs, and LOINC codes.
  - Unlike HMP, Text search is not the standard method for data retrieval, and will act a secondary path to discover patient data.
- Tasks – (PSI 6-8) Tasks will be displayed, and given that they are in-progress, can be initialized and completed using the orders and note writer applets (PSI 6-8)
- Add Task – (PSI 6-8) The ability to create a new Note or Order against the patient record

8.6. Header

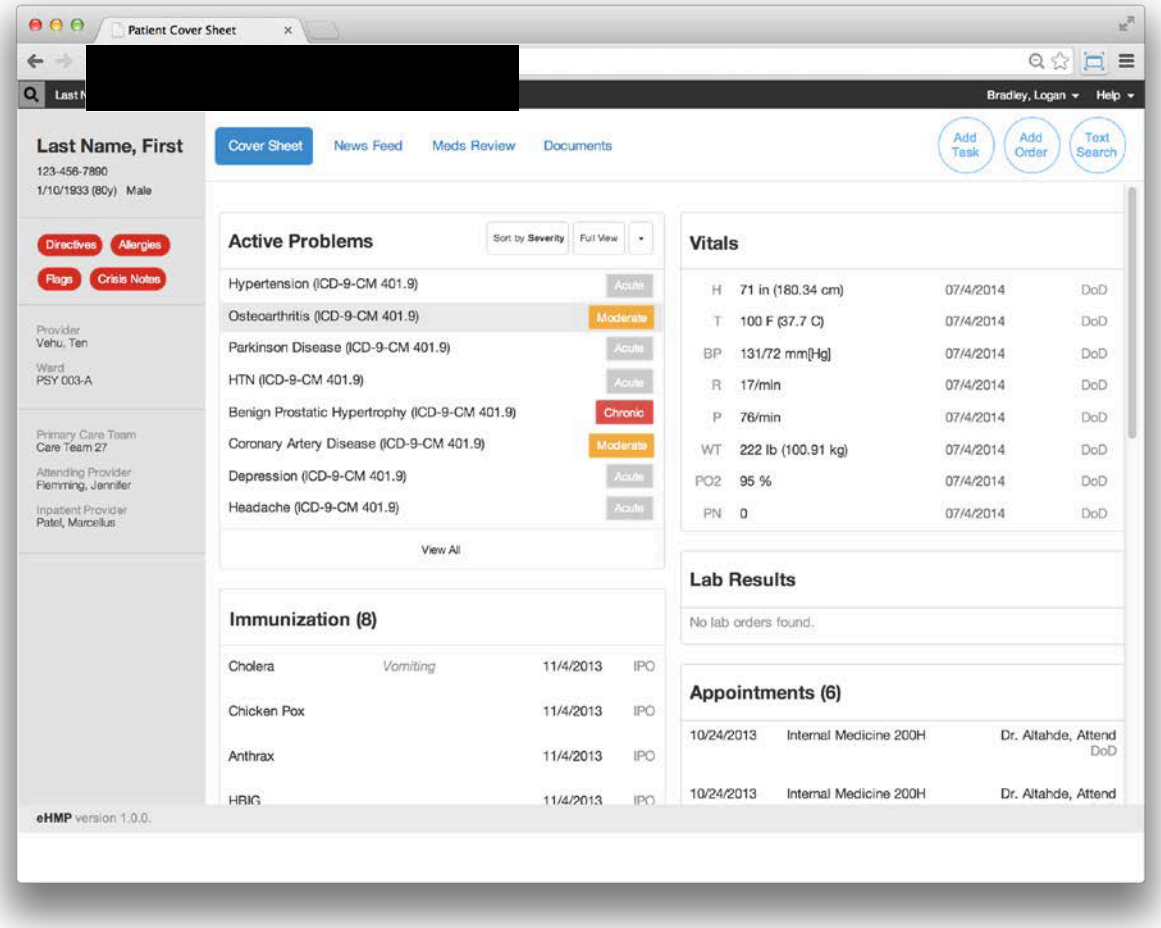
Consists of the following:

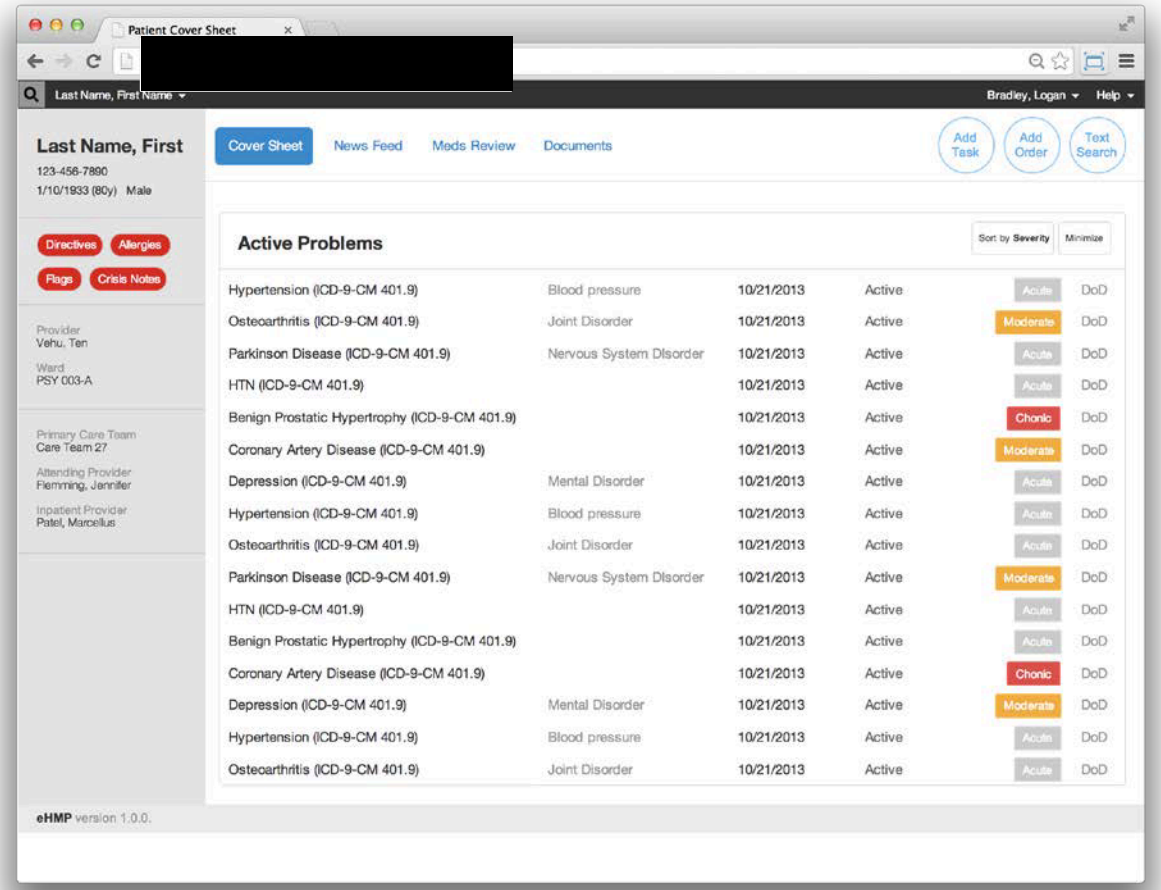
- Patient Selection – Will allow access to the patient selection applet
- Last 20 patient selection – (PSI 6) Will give list of last 20 patients viewed and will allow for hot swapping of patient context (along with confirmation and sensitivity flags)
- Display of User information: To show the current logged in user, and the facility in which they have logged into.
- Access to help features for any given screen (PSI 7)

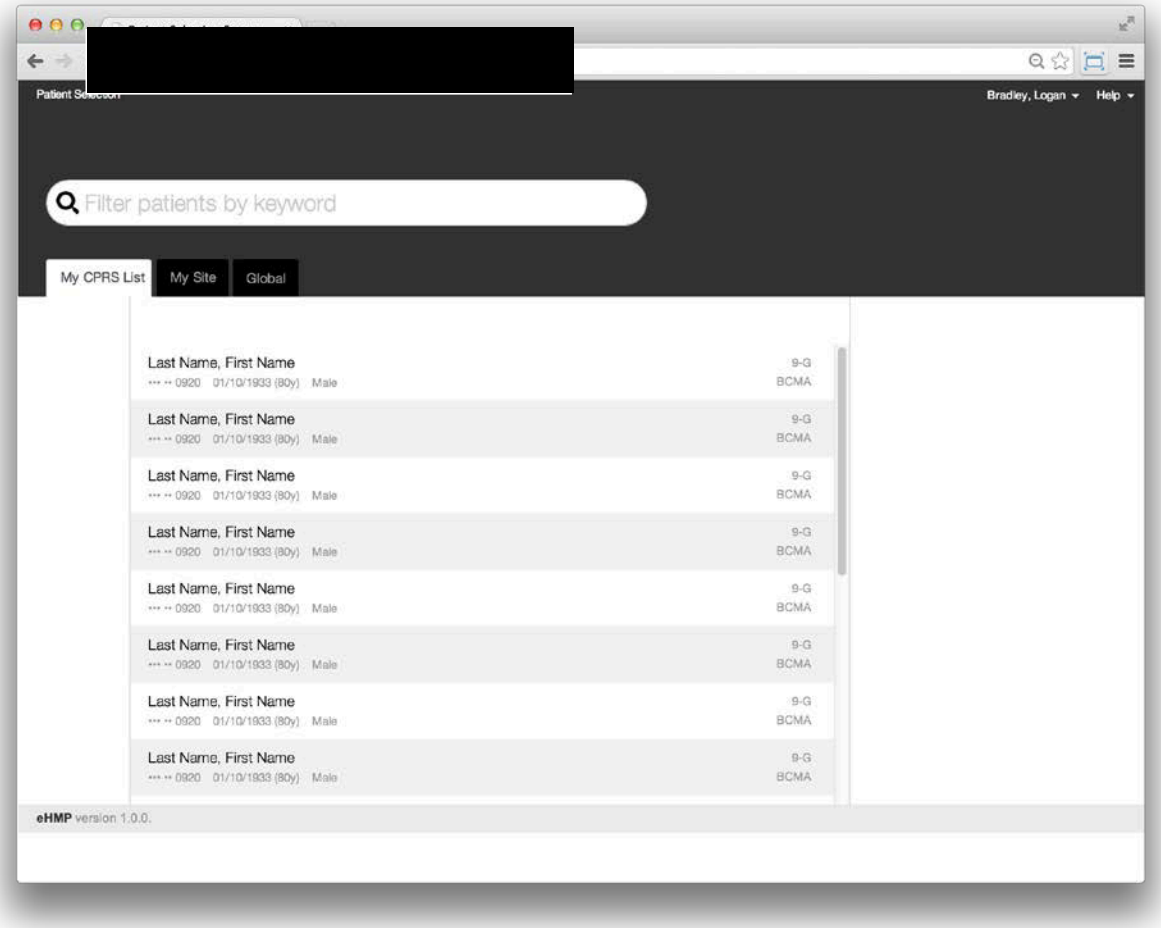
8.7. Footer

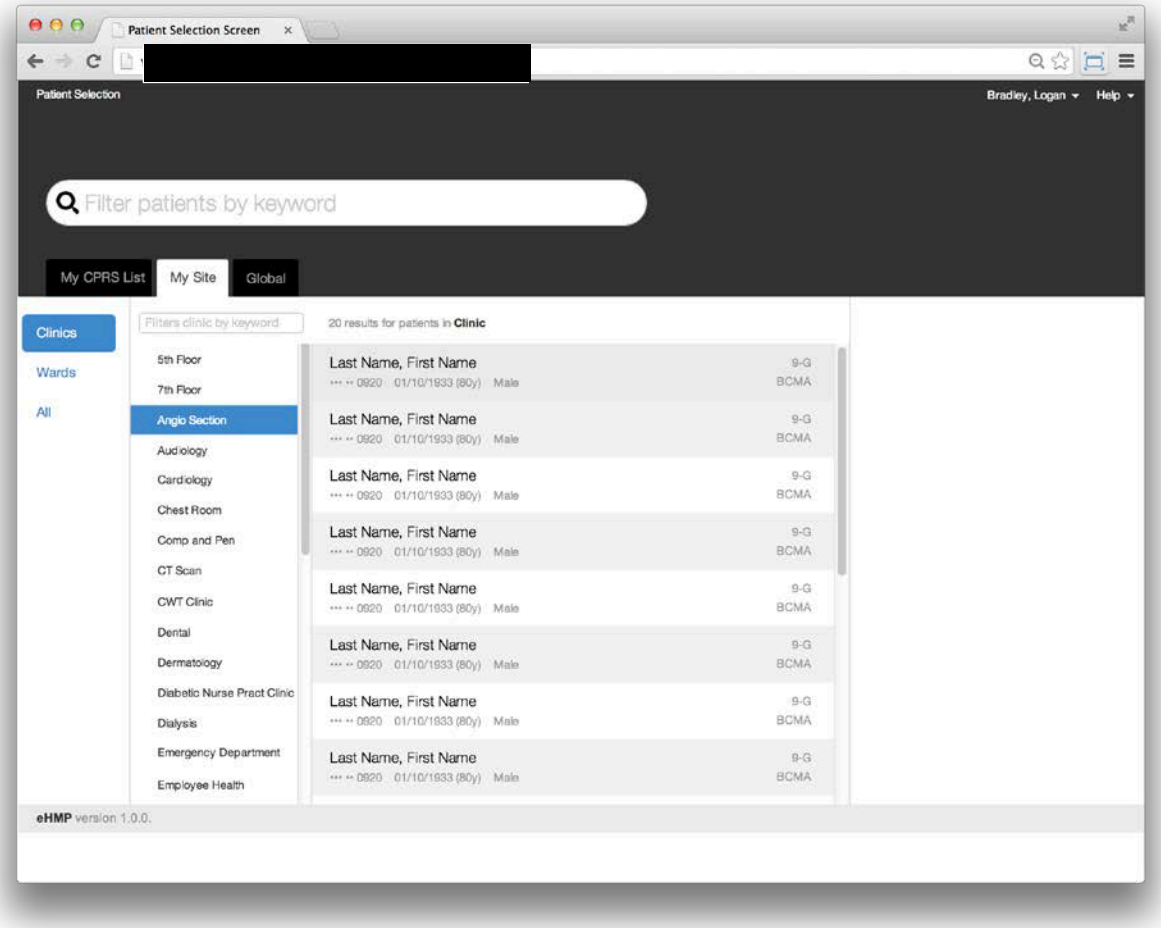
Consists of the following:

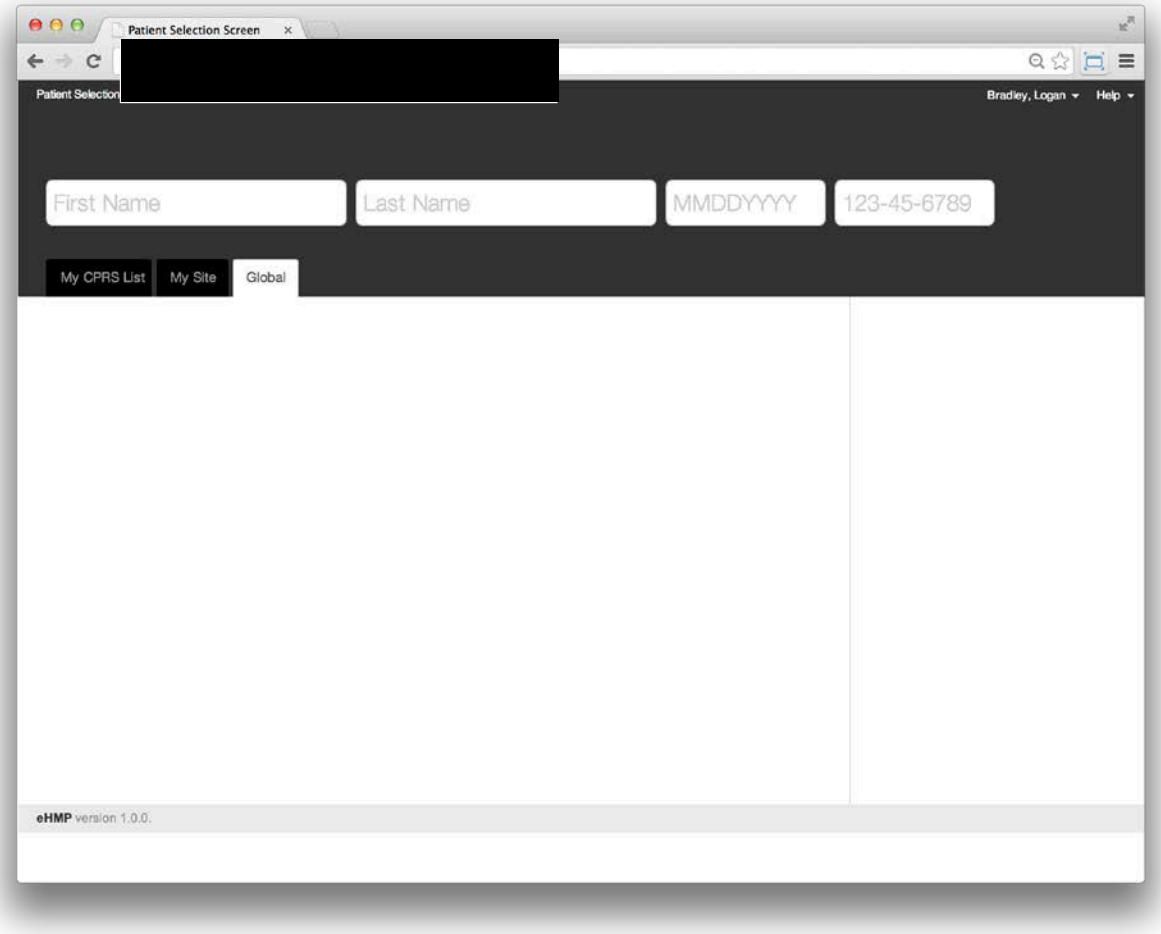
- Application Name (eHMP)
- Application version (v##.##.##)


Patient Cover Sheet
[redacted]
Bradley, Logan
Help
Last Name, First
123-456-7890
1/10/1933 (80y) Male
Directives
Allergies
Flags
Crisis Notes
Provider
Vehu, Ten
Ward
PSY 003-A
Primary Care Team
Care Team 27
Attending Provider
Flemming, Jennifer
Inpatient Provider
Patel, Marcellus
Cover Sheet
News Feed
Meds Review
Documents
Add Task
Add Order
Text Search
Active Problems
Sort by Severity
Full View
Hypertension (ICD-9-CM 401.9) Acute
Osteoarthritis (ICD-9-CM 401.9) Moderate
Parkinson Disease (ICD-9-CM 401.9) Acute
HTN (ICD-9-CM 401.9) Acute
Benign Prostatic Hypertrophy (ICD-9-CM 401.9) Chronic
Coronary Artery Disease (ICD-9-CM 401.9) Moderate
Depression (ICD-9-CM 401.9) Acute
Headache (ICD-9-CM 401.9) Acute
View All
Immunization (8)
Cholera Vomiting 11/4/2013 IPO
Chicken Pox 11/4/2013 IPO
Anthrax 11/4/2013 IPO
HBIG 11/4/2013 IPO
Vitals
H 71 in (180.34 cm) 07/4/2014 DoD
T 100 F (37.7 C) 07/4/2014 DoD
BP 131/72 mm[Hg] 07/4/2014 DoD
R 17/min 07/4/2014 DoD
P 76/min 07/4/2014 DoD
WT 222 lb (100.91 kg) 07/4/2014 DoD
PO2 95 % 07/4/2014 DoD
PN 0 07/4/2014 DoD
Lab Results
No lab orders found.
Appointments (6)
10/24/2013 Internal Medicine 200H Dr. Altahde, Attend DoD
10/24/2013 Internal Medicine 200H Dr. Altahde, Attend
eHMP version 1.0.0.


Patient Cover Sheet
[redacted]
Last Name, First Name
Bradley, Logan
Help
Last Name, First
123-456-7890
1/10/1933 (80y) Male
Directives
Allergies
Flags
Crisis Notes
Provider
Vehu, Ten
Ward
PSY 003-A
Primary Care Team
Care Team 27
Attending Provider
Flemming, Jennifer
Inpatient Provider
Patel, Marcellus
Cover Sheet
News Feed
Meds Review
Documents
Add Task
Add Order
Text Search
Active Problems
Sort by Severity
Minimize
Hypertension (ICD-9-CM 401.9) Blood pressure 10/21/2013 Active Acute DoD
Osteoarthritis (ICD-9-CM 401.9) Joint Disorder 10/21/2013 Active Moderate DoD
Parkinson Disease (ICD-9-CM 401.9) Nervous System Disorder 10/21/2013 Active Acute DoD
HTN (ICD-9-CM 401.9) 10/21/2013 Active Acute DoD
Benign Prostatic Hypertrophy (ICD-9-CM 401.9) 10/21/2013 Active Chonic DoD
Coronary Artery Disease (ICD-9-CM 401.9) 10/21/2013 Active Moderate DoD
Depression (ICD-9-CM 401.9) Mental Disorder 10/21/2013 Active Acute DoD
Hypertension (ICD-9-CM 401.9) Blood pressure 10/21/2013 Active Acute DoD
Osteoarthritis (ICD-9-CM 401.9) Joint Disorder 10/21/2013 Active Acute DoD
Parkinson Disease (ICD-9-CM 401.9) Nervous System Disorder 10/21/2013 Active Moderate DoD
HTN (ICD-9-CM 401.9) 10/21/2013 Active Acute DoD
Benign Prostatic Hypertrophy (ICD-9-CM 401.9) 10/21/2013 Active Acute DoD
Coronary Artery Disease (ICD-9-CM 401.9) 10/21/2013 Active Chonic DoD
Depression (ICD-9-CM 401.9) Mental Disorder 10/21/2013 Active Moderate DoD
Hypertension (ICD-9-CM 401.9) Blood pressure 10/21/2013 Active Acute DoD
Osteoarthritis (ICD-9-CM 401.9) Joint Disorder 10/21/2013 Active Acute DoD
eHMP version 1.0.0.


[redacted]
Patient Selection
Bradley, Logan
Help
Filter patients by keyword
My CPRS List
My Site
Global
Last Name, First Name
••• •• 0920 01/10/1933 (80y) Male
9-G
BCMA
eHMP version 1.0.0.


Patient Selection Screen
Patient Selection
Bradley, Logan
Help
Filter patients by keyword
My CPRS List
My Site
Global
Clinics
Wards
All
Filters clinic by keyword
20 results for patients in Clinic
5th Floor
7th Floor
Angio Section
Audiology
Cardiology
Chest Room
Comp and Pen
CT Scan
CWT Clinic
Dental
Dermatology
Diabetic Nurse Pract Clinic
Dialysis
Emergency Department
Employee Health
Last Name, First Name
01/10/1933 (80y) Male
9-G
BCMA
eHMP version 1.0.0.



9. System Integrity Controls

The eHMP system design will comply with the NIST 800-53 Rev4 [3] Security Controls as described in the eHMP System Security Plan (SSP). This plan will be available within the VA's GRC RiskVision tool.

Encryption of data at rest will be protected using Federal Information Professing Standard (FIPS) 140-2 compliant encryption algorithms and products to protect patient data. A Caché database will be used to store data in multiple files (or UNIX raw disk partitions) called *databases*. All database reads and writes occur in 8192-byte blocks. This includes application data as well as metadata such as indices, bitmaps, pointers, allocation maps, and incremental backup maps. All blocks (with the exception of a single initial label block) are always encrypted on disk.

The actual encryption and decryption is done at a low-level in the Caché system code, and uses the United States Government Advanced Encryption Standard (AES) [FIPS 197] in Cipher Block Chaining (CBC) mode [NIST 800-38A], with a 128-, 192-, or 256-bit database encryption key. The initialization vector for

[3] NIST 800-53 Rev4: http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r4.pdf

each block is generated by encrypting a big-endian representation of the logical block number using AES in Electronic Codebook (ECB) mode [NIST 800-38A & Addendum], using the database encryption key.

Data at rest stored in other areas of the system such as log files that could hold sensitive data will be encrypted using Red Hat Encrypted File System (EFS), which utilizes FIPS 140-2 compliant encryption algorithms.

Further details of the design for system integrity controls will be completed during increment 1.

9.1. Transmission Encryption

The eHMP system design will employ the secure Transport Layer Security (TLS) protocol to protect the integrity of data in transit. TLS will establish FIPS-compliant, encrypted communications sessions on connections outside of the eHMP environment. These connections will be established based upon VA issued Digital Encryption Certificates installed on each of the eHMP servers. This will protect the confidentiality and integrity of the data being transmitted.

Multi-factor user authentication will be employed as the Personal Identity and Verification (PIV) cards are implemented throughout the VA Staff user population. It is anticipated that single-factor user id and password may be required at the initial launch of the eHMP system until the entire VA Staff user population are issued PIV cards. The multifactor authentication will consist of something a user has (PIV card) and something they know (PIN). This will enhance the integrity controls by making it exponentially more difficult for a user account to be compromised. Once a user is identified and authenticated his or her access to system resources and application functionality will be based upon role based access controls.

9.2. System Administrator Access

The eHMP production environment will be hosted within the VA AITC EO cloud computing environment. System administrator access necessary to manage eHMP resources will employ the following technical resources and methods to ensure the integrity of the production environment. Administrator access will require a user account that is authorized and granted administrator privileges for the eHMP system resources. Further, administrators will be limited to accessing only those resources necessary to accomplish their assigned tasks. Administrator access will also be subject to auditing of all privileged actions to include Logon, Logoff, user account management, and administrative access to system servers and sensitive resources such as database access.

A user assigned administrative duties will require authentication via a VA issued PIV as described in section 6.2.3.2 and 9.1 above. To establish an administrative connection with the eHMP system, a user must be using VA Citrix Access Gateway (CAG) and a valid VA issued PIV. Once authenticated, the administrative user will be joined to the local VLAN hosting the eHMP system resources and the VPN client for Citrix Receiver will establish a FIPS 140-2 compliant encrypted session that will enforce isolation of the administrator's laptop or desktop to prevent compromise of the production environment.

9.3. Flaw Remediation

The eHMP system will handle remediation of software flaws by utilizing a number of tools to detect and remediate vulnerabilities such as Nessus, HP Fortify, and SonarQube. A yum repository that is controlled by AITC will exist within the environment to download and apply to appropriate patches to the instances of Red Hat Linux OS. Patching of additional open source software components will be done manually and any vulnerabilities will be identified through monthly VA NSOC Nessus scans. VA NSOC Nessus scan results will be provide to the DevOps and Security teams for review and remediation. Any findings that require further testing and evaluation will be tracked in a Plan of Actions and Milestones (POA&M) until closed.

The eHMP source code will be scanned utilizing HP Fortify static code analyzer (SCA), prior to each build by the development team and all critical and high vulnerabilities will be remediated before moving to the Enterprise Development Environment for testing on the VA network. Code quality will also be evaluated using tools built into the SonarQube software. The eHMP program will support 3 environments to include test, pre-production and production. All changes to the source code will be tested thoroughly in each of the environments before to identify and eliminate the risk of flaws prior to migrating to production. All servers within the test, pre-production and production environments are required by the VA to have McAfee Antivirus and Host Intrusion Protection Systems (HIPS) for continuous monitoring and protection from any malware/malicious code. Remediation processes for related projects such as JLV will be evaluated for incorporation of best practices.

VA NSOC performs monitoring on all AITC networks utilizing IDS/IPS sensors and also monitors the gateways, firewalls and routers to conduct near real time analysis using HP OpenView software. eHMP will utilize a suite of open source tools that include Sensu, ElasticSearch, Kibana and FluentD that will serve as a monitoring and logging solution for the environment.

9.4. Configuration Management (CM)

Please refer to the eHMP Configuration Management Plan and Process Module document for further detail.

10. Approval Signatures

The signature below is an acknowledgement that the signatory understands the purpose and content of this document.

Signed: _____

Integrated Project Team Chair Date

Signed: _____

Business Sponsor Date

Signed: _____

IT Program Manager Date

Signed: _____

Project Manager Date

Signed: _____

Enterprise Architecture Date

Signed: _____

Service Delivery and Engineering Date

Appendix A – Acronym List

A&A	Assessment and Authorization
ACID	Atomicity, Consistency, Isolation, Durability
AITC	Austin Information Technology Center
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ASM	ASM Research
BHIE	Bidirectional Health Exchange
BSON	Binary JSON
CAG	Citrix Access Gateway
CBC	Cipher Block Chaining
CCB	Change Control Board
CCOW	Clinical Context Object Workgroup
CDA	Clinical Document Architecture
CDS	Clinical Data Service
CDW	Corporate Data Warehouse
CEM	Clinical Element Model
CHDR	Clinical Health Data Repository
CLIN	Contract Line Item Number
CMDB	Configuration Management Database
CPMP	Contractor Project Management Plan
CPRS	Computerized Patient Record System
CR	Change Request
DAO	Data Access Object
DAS	Data Access Service
DB	Database
DBMS	Database Management System
DCO	Data Center Operations
DICOM	Digital Imaging and Communications in Medicine
DFN	Data File Number
DMZ	Demilitarized Zone
DoD	Department of Defense
ECB	Electronic Codebook
EDE	Enterprise Development Environment
EHR	Electronic Health Record
EO	Enterprise Operations
eHMP	Enterprise Health Management Platform
eVPR	Enterprise Virtual Patient Record
FHIM	Federal Health Information Model

FHIR	Fast Healthcare Interoperability Resources
FIPS	Federal Information Processing Standards
GUI	Graphical User Interface
HDR	Health Data Repository
Hi2	Health Informatics Initiative
HIPS	Host Intrusion Protection Systems
HITSP	Healthcare Information Technology Standards Panel
HL7	Health Level Seven International
HMP	Health Management Platform
HTTP	Hypertext Transfer Protocol
IAM	Identity Access Management
ICN	Integration Control Number
iEHR	Integrated Electronic Health Record
JAR	Java Archive
JDS	JSON Data Store
JLV	Joint Legacy Viewer
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
LOINC	Logical Observation Identifiers Names and Codes
MBI	Moderate Risk Background Investigation
MDHT	Model Driven Health Tools
MDWS	Medical Domain Web Services
MVI	Master Veteran Index
MU	Meaningful Use
MVC	Model View Controller
NDAA	National Defense Authorization Act
NIEM	National Information Exchange Model
NwHIN	Nationwide Health Information Network
OHT	Open Health Tools
OI&T	Office of Information Technology
ONC	Office of National Coordinator
OSEHRA	Open Source Electronic Health Record Agent
PB	Petabyte
PITC	Philadelphia Information Technology Center
PDF	Portable Document Format
PMAS	Project Management Accountability System
POA&M	Plan of Actions and Milestones
PSI	Potentially Shippable Increment
QA	Quality Assurance
RDF	Resource Description Framework

REST	Representational State Transfer
RHEL	Red Hat Enterprise Linux
RPC	Remote Procedure Call
SAM	Scaled Agile Methodology
SAN	Storage Area Network
SCA	Static Code Analyzer
SDD	System Design Document
SDK	Software Development Kit
SDM	Service Desk Manager
SNOMED	Systematized Nomenclature of Medicine
SOA	Service-Oriented Architecture
SOR	System of Record
TCP/IP	Transmission Control Protocol/Internet Protocol
TLS	Transport Layer Security
TRM	Technical Reference Manual
UAT	User Acceptance Testing
URL	Uniform Resource Locator
UX	User Experience
VA	Department of Veterans Affairs
VDEF	VistA Data Extraction Framework
VHA	Veterans Health Administration
VistA	Veterans Health Information Systems and Technology Architecture
VLER	Veteran Lifetime Electronic Record
VM	Virtual Machine
VPR	Virtual Patient Record
WAR	Web Application Archive
W3C	World Wide Web Consortium
XACML	Extensible Access Control Markup
XML	Extensible Markup Language

.

Appendix B – FHIR Mappings

The most current version of the mappings to FHIR can be found on the project wiki [redacted].

An example of an allergy represented via FHIR is listed below.


```
{
 "resourceType":"AdverseReaction",
 "extension": [
  {
   "url":[redacted]
   "value":"allergy"
  },
 "extension": [
  {
   "url[redacted],
   "value":"2013-12-11T12:13:00"
  }
 ],
 "text":{
  "status":"generated",
  "div":"<div>PENICILLIN</div>"
 },
 "contained":[
  {
   "resourceType":"Practitioner",
   "id":"3702dba0-68cf-11e3-949a-0800200c9a66",
   "text":{
    "status":"generated",
    "div":"<div>PROVIDER,ONE</div>"
   },
   "type":{
    "text":"PROVIDER,ONE"
   }
  },
  {
   "resourceType":"Substance",
   "id":"b9f845b6-cce1-4f3b-bb9d-e87e9a7b9eb6",
   "text":{
    "status":"generated",
    "div":"<div>PENICILLIN</div>"
   },
   "type":{
    "coding":{
     "system":"urn:oid:2.16.840.1.113883.6.233",
     "code":"urn:va:vuid:4019880",
     "display":"PENICILLIN"
    },
    "text":"PENICILLIN"
```

```
   }
  }
 ],
 "identifier":{
  "use":"official",
  "value":"urn:va:B362:1:art:1"
 },
 "subject":{
  "reference":"Patient/1"
 },
 "didNotOccurFlag": false,
 "recorder": {
  "reference":"#3702dba0-68cf-11e3-949a-0800200c9a66"
 },
 "exposure":[
  {
   "substance": {
    "reference":"#b9f845b6-cce1-4f3b-bb9d-e87e9a7b9eb6"
   }
  }
 ]
}
```