MCCF\_EDI\_TAS\_System\_Design\_Document

Medical Care Collection Fund (MCCF)

**Electronic Data Interchange Transaction Application Suite (EDI TAS)**



**November 2018**

**Version 0.9**

**Department of Veterans Affairs**

Revision History

| Date | Version | Description | Author |
| --- | --- | --- | --- |
| November 9, 2018 | 0.9 | Added clarifications to software architecture and corrected information as requested by PMO | Halfaker Team |
| May 21, 2018 | 0.8 | Added Wireframes | Halfaker Team |
| December 12, 2017 | 0.7 | Added VistA Data Access Services, TAS Reporting, | Halfaker Team |
| July 21, 2017 | 0.6 | Added Error Handling, Logging, Service Design, User Types, UI Design | Halfaker Team |
| May 8, 2017 | 0.5 | Added Technology Locations, Software Architecture and Identified Systems | Halkfaker Team |
| February 24, 2017 | 0.4 | Added tables of figures and tables, and addressed other tech writer review matters | Halfaker Team |
| February 14, 2017 | 0.3 | Technical Writer Review Complete. | Leidos PMO |
| February 2, 2017 | 0.2 | Revision based on MCCF EDI TAS Team review | Halfaker Team |
| January 27, 2017 | 0.1 | Initial Version of MCCF EDI TAS Architecture | Halfaker Team |

Artifact Rationale

This 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.

Table of Contents

[1. Introduction 1](#_Toc529518411)

[1.1. Scope 2](#_Toc529518412)

[2. Background 2](#_Toc529518413)

[2.1. Overview of the System 2](#_Toc529518414)

[2.2. Business Process Overview 3](#_Toc529518415)

[2.3. High-level Claims Process 4](#_Toc529518416)

[2.4. High-level Request for Additional Information Process 5](#_Toc529518417)

[2.5. High-level Pre-certification Process 6](#_Toc529518418)

[2.6. Overview of the Significant Requirements 7](#_Toc529518419)

[2.6.1. Architecture Platform Epics 7](#_Toc529518420)

[2.6.2. Overview of the Functional Workload/Performance Requirements 9](#_Toc529518421)

[2.6.3. Overview of Operational Requirements 11](#_Toc529518422)

[2.6.3.1. Scalability 11](#_Toc529518423)

[2.6.3.2. Availability 12](#_Toc529518424)

[2.6.3.3. Disaster Recovery (DR) 12](#_Toc529518425)

[2.6.4. Architecture Timeline 12](#_Toc529518426)

[3. Conceptual Design 13](#_Toc529518427)

[3.1. Conceptual Application Design 13](#_Toc529518428)

[3.1.1. User Profiles 16](#_Toc529518429)

[3.1.1.1. TASCore User Types 16](#_Toc529518430)

[3.1.2. Application Locations 18](#_Toc529518431)

[3.1.2.1. Identified Systems 18](#_Toc529518432)

[3.1.3. MCCF EDI TAS Automated Software Installation and Configuration 19](#_Toc529518433)

[3.2. Conceptual Data Design 27](#_Toc529518434)

[3.2.1. Project Conceptual Data Model 27](#_Toc529518435)

[3.2.1.1. FHIR Resources Needed for MCCF EDI TAS 27](#_Toc529518436)

[3.2.2. User Interface Data Mapping 29](#_Toc529518437)

[3.2.2.1. Application Screen Interface 29](#_Toc529518438)

[3.2.2.1.1 Mapping IB screens to FHIR resources 29](#_Toc529518439)

[3.3. Conceptual Infrastructure Design 30](#_Toc529518440)

[3.3.1. System Criticality and High Availability 30](#_Toc529518441)

[3.3.2. Special Technology 30](#_Toc529518442)

[3.3.3. Technology Locations 30](#_Toc529518443)

[3.3.4. Conceptual Infrastructure Diagram 31](#_Toc529518444)

[3.3.4.1. Conceptual Production String Diagram 32](#_Toc529518445)

[4. System Architecture 33](#_Toc529518446)

[4.1. Hardware Architecture 33](#_Toc529518447)

[4.2. Software Architecture 34](#_Toc529518448)

[4.2.1.1. MCCF VA Base Packages 34](#_Toc529518449)

[4.2.1.2. MCCF DEV Packages 34](#_Toc529518450)

[4.2.2. Node and Angular Shared Components 35](#_Toc529518451)

[4.3. Network Architecture 35](#_Toc529518452)

[4.4. Service Oriented Architecture / ESS 36](#_Toc529518453)

[4.5. Enterprise Architecture 37](#_Toc529518454)

[5. Data Design 39](#_Toc529518455)

[5.1. DBMS Files 39](#_Toc529518456)

[5.2. Non-DBMS Files 39](#_Toc529518457)

[5.3. Data View 39](#_Toc529518458)

[5.3.1. MCCF EDI TAS Report Data Element Mappings 40](#_Toc529518459)

[5.3.1.1. EDI Lockbox Report Data Element Mappings 40](#_Toc529518460)

[5.3.1.2. ECME OPECC Report Data Element Mappings 40](#_Toc529518461)

[6. Detailed Design 41](#_Toc529518462)

[6.1. Hardware Detailed Design 41](#_Toc529518463)

[6.2. Software Detailed Design 42](#_Toc529518464)

[6.2.1. Conceptual Design 42](#_Toc529518465)

[6.2.1.1. Presentation Layer Design 42](#_Toc529518466)

[6.2.1.2. Services Layer Design 45](#_Toc529518467)

[6.2.1.2.1. User Interfaces 56](#_Toc529518468)

[6.2.1.2.2. Communications Interfaces 74](#_Toc529518469)

[6.2.1.2.3. Orchestration Design 75](#_Toc529518470)

[6.2.1.3. Data Access Services Design 75](#_Toc529518471)

[6.2.1.3.1. NGINX 76](#_Toc529518472)

[6.2.1.3.2. HAPI FHIR Server 76](#_Toc529518473)

[6.2.1.3.3. VistA Access and Routing Message Flows 76](#_Toc529518474)

[6.2.1.3.4. VistA Instances 77](#_Toc529518475)

[6.2.1.4. Data Storage Design 77](#_Toc529518476)

[6.2.1.4.1. Azure Storage Mechanics 77](#_Toc529518477)

[6.2.1.4.2. Summary 79](#_Toc529518478)

[6.2.1.5. TAS Architecture Capabilities 79](#_Toc529518479)

[6.2.1.5.1. TAS Reporting Capability 79](#_Toc529518480)

[6.2.1.5.2. TAS FSC Interface Capability 82](#_Toc529518481)

[6.2.1.5.3. TAS Proxying 84](#_Toc529518482)

[6.2.1.5.4. TAS CORE Caching Strategy 86](#_Toc529518483)

[6.2.1.6. Dependencies and Constraints 91](#_Toc529518484)

[6.2.2. Specific Requirements 91](#_Toc529518485)

[6.2.2.1. Database Repository 91](#_Toc529518486)

[6.2.2.2. System Features 91](#_Toc529518487)

[6.3. Network Detailed Design 92](#_Toc529518488)

[6.4. Security and Privacy 93](#_Toc529518489)

[6.4.1. Security 93](#_Toc529518490)

[6.4.2. Privacy 93](#_Toc529518491)

[6.4.3. Security 94](#_Toc529518492)

[6.4.3.1. Authentication and Authorization with IAM 94](#_Toc529518493)

[6.4.3.1.1. Authorization 95](#_Toc529518494)

[6.5. Service Oriented Architecture / ESS Detailed Design 96](#_Toc529518495)

[6.5.1. Service Integration Flow 97](#_Toc529518496)

[6.6. Interface Design Rules 97](#_Toc529518497)

[7. External System Interface Design 97](#_Toc529518498)

[7.1. Interface Architecture 97](#_Toc529518499)

[7.1.1. TAS Web Development Ports 99](#_Toc529518500)

[7.1.2. CIT MAG Servers 102](#_Toc529518501)

[7.1.3. SQA MAG Servers 104](#_Toc529518502)

[7.1.4. UAT MAG Servers 105](#_Toc529518503)

[7.2. Interface Detailed Design 106](#_Toc529518504)

[7.2.1. eBilling ICD 106](#_Toc529518505)

[8. Human-Machine Interface 107](#_Toc529518506)

[8.1. Inputs 107](#_Toc529518507)

[8.2. Outputs 107](#_Toc529518508)

[8.3. Navigation Hierarchy 107](#_Toc529518509)

[8.3.1. Screens 107](#_Toc529518510)

[9. Appendix A: STAT Team Diagrams 108](#_Toc529518511)

[9.1. STAT Team Diagram Reference to Equivalent SDD Detailed Diagram 108](#_Toc529518512)

[10. Attachment A – Approval Signatures 112](#_Toc529518513)

[A.1. Identification of Technology and Standards 113](#_Toc529518514)

[A.2. Constraining Policies, Directives and Procedures 113](#_Toc529518515)

[A.3. Requirements Traceability Matrix 113](#_Toc529518516)

[A.4. Packaging and Installation 113](#_Toc529518517)

[A.5. Design Metrics 113](#_Toc529518518)

Table of Figures

[Figure 1: VA Revenue Cycle Business Process 4](#_Toc529518519)

[Figure 2: Transactions per Hour when Insurance Collected at Each Visit 10](#_Toc529518520)

[Figure 3: Transactions per Hour when Insurance Requested Once per Month per Unique Patient 11](#_Toc529518521)

[Figure 4: MCCF EDI TAS Conceptual Architecture 13](#_Toc529518522)

[Figure 4: MCCF EDI TAS Conceptual Architecture 13](#_Toc529518523)

[Figure 5: Servers Hosting the Software Components 19](#_Toc529518524)

[Figure 6: Test Environment Conceptual Infrastructure Diagram, Part 1 31](#_Toc529518525)

[Figure 7: Test Environment Conceptual Infrastructure Diagram, Part 2 32](#_Toc529518526)

[Figure 8: Conceptual Production String Diagram 33](#_Toc529518527)

[Figure 9 - MCCF EDI TAS Package Management Process 35](#_Toc529518528)

[Figure 10: MCCF EDI TAS Logical Architecture 37](#_Toc529518529)

[Figure 11: VA Future IT Vision Diagram (5-year) 38](#_Toc529518530)

[Figure 12 - UI Lazy Loading Sample Web Page 45](#_Toc529518531)

[Figure 13 - MCCF EDI TAS Load Balanced node.js Web System 45](#_Toc529518532)

[Figure 14 - MCCF EDI TAS Services Design 46](#_Toc529518533)

[Figure 15 - MCCF TAS node.js Logging Components 47](#_Toc529518534)

[Figure 16 - MCCF TAS Logging Configuration 53](#_Toc529518535)

[Figure 17 - MCCF TAS Health Monitoring System High-level Design 54](#_Toc529518536)

[Figure 18 - MCCF TAS Health Check Flow 55](#_Toc529518537)

[Figure 19 - USWDS Design Elements 57](#_Toc529518538)

[Figure 20 - MCCF TAS Portal Screen Mockup 58](#_Toc529518539)

[Figure 21 - USWDS Landing Page Template 59](#_Toc529518540)

[Figure 22 - USWDS Documentation Page Template 60](#_Toc529518541)

[Figure 23 - MCCF TAS Error Handling High-level Design 66](#_Toc529518542)

[Figure 24 - MCCF TAS Custom Error Message 69](#_Toc529518543)

[Figure 25 - MCCF TAS Email Notification Acknowledgement 70](#_Toc529518544)

[Figure 26 - MCCF TAS Error Email Notification 71](#_Toc529518545)

[Figure 27 - MCCF TAS Off-site Notification Flow 72](#_Toc529518546)

[Figure 28 - MCCF TAS Log User Interface 74](#_Toc529518547)

[Figure 29 - MCCF EDI TAS VistA Data Access Services Design 75](#_Toc529518548)

[Figure 30 – VistA Data Access Routing Architecture 77](#_Toc529518549)

[Figure 30 - TAS Reporting Design 79](#_Toc529518550)

[Figure 31 - Viewing Reports in Reporting Frontend 80](#_Toc529518551)

[Figure 32 - Editing configuration for loading VistA data into MCCF Reporting Database 81](#_Toc529518552)

[Figure 30 - TAS FSC Interface Design 83](#_Toc529518553)

[Figure 31 - Processing EDI Transactions through FSC Interface 83](#_Toc529518554)

[Figure 31 – FSC Interface EDI Transaction Log Entries 84](#_Toc529518555)

[Figure 33 - MCCF EDI TAS Physical Architecture Overview 92](#_Toc529518556)

[Figure 39: Service Integration Flow Diagram 97](#_Toc529518557)

[Figure 40: High-level Application Design 98](#_Toc529518558)

[Figure 41: MCCF EDI TAS Interface Architecture 98](#_Toc529518559)

Table of Tables

[Table 1: FY 2016 Transaction Volumes 11](#_Toc529518690)

[Table 2 - User Type by eBusiness Team 17](#_Toc529518691)

[Table 3 - MCCF EDI TAS FHIR Resources by Product Line 27](#_Toc529518692)

[Table 4 - MCCF EDI TAS Server Specifications 41](#_Toc529518693)

[Table 5 – Event Logfile to Table Row Mapping 73](#_Toc529518694)

[Table 6 - Exception Logfile to Table Row Mapping 74](#_Toc529518695)

[Table 7 - Azure Storage Shared Access Signature 78](#_Toc529518696)

[Table 8 - MCCF EDI TAS Reporting Nightly Loading Sequence Flow Diagram 81](#_Toc529518697)

[Table 9 - MCCF EDI TAS Business Service Proxying 85](#_Toc529518698)

[Table 10 - MCCF EDI TAS Caching by Layer 90](#_Toc529518699)

[Table 11: DEV MAG Servers 99](#_Toc529518700)

[Table 12: DEV MAG Connections 100](#_Toc529518701)

[Table 13: CI MAG Servers 101](#_Toc529518702)

[Table 14: CI MAG Servers 102](#_Toc529518703)

[Table 15: CI MAG Servers 104](#_Toc529518704)

[Table 16: CI MAG Servers 105](#_Toc529518705)

[Table 17: FSC Interface Design 106](#_Toc529518706)

[Table 18: Planned Nuance Claim Scrubber Interface Design 107](#_Toc529518707)

[Table 19: IAM Interface Design 107](#_Toc529518708)

# Introduction

The eBusiness Solutions Office manages the development, implementation and ongoing support of the Department of Veterans Affairs (VA) Electronic Data Interchange (EDI) applications within VistA in support of revenue generation from third party payers. By monitoring legislative, legally mandated, and other industry requirements, enacted under the Health Insurance Portability and Accountability Act (HIPAA) and subsequent regulations, eBusiness Solutions supports innovative enhancements to VA software ensuring the continued flow of revenue to meet the needs of our Nation’s Veterans.

eBusiness focuses upon:

* Assuring compliance with industry EDI standards setting organizations such as the Workgroup for Electronic Data Interchange (WEDI), the Accredited Standards Committee X12 (ASC X12), the Council for Affordable Quality Healthcare (CAQH), the National Council on Prescription Drug Plans (NCPDP), and the National Committee on Vital and Health Statistics (NCVHS).
* Working collaboratively with the Office of Information Technology (OIT) as the EDI business owner to develop and update the Veterans Health Administration (VHA) software, including eInsurance, eBilling, ePharmacy and ePayments. Ensures multi-year funding projections are included in VA submission to the President’s budget.
* Providing training to VA Medical Center (VAMC) and Consolidated Patient Account Center (CPAC) staff on the updated EDI software, including program support to staff and trading partners.
* Ensuring electronic connectivity to over 1,600 payers, including proactive outreach to ensure payer compliance with mandates, helping to educate payers on EDI standards, and reporting non-compliant payers to Centers for Medicare and Medicaid Services (CMS).
* Maintaining partnerships with Federal agencies and trading partners who assist with EDI business and work flow. This includes Federal partners such as Health & Human Services (HHS), the Internal Revenue Service (IRS) and Treasury; and EDI clearinghouses and commercial banking partners.

The vision of the Medical Care Collection Fund/Application Programming Interface (MCCF/API) 2.0 work effort is to modernize and evolve the systems used for EDI transactions. The MCCF program seeks to transition from a Veterans Health Information Systems and Technology Architecture (VistA) - based architecture to a Service Oriented Architecture (SOA). The end state is to transition business logic from VistA to a modernized solution while using VistA as the authoritative data source.

This SDD is a dual-use document that provides the conceptual design as well as the to-be design for the MCCF EDI Transaction Application Suite (TAS). This document will be updated as the product is built, to reflect the as-built product.

The intended audience of this document includes the eBusiness Solutions Office, Enterprise Program Management Office (ePMO), Product Engineering, Software Quality Assurance (SQA), the Chief Business Office (CBO), and staff at the Office of Information and Technology (OIT) at the Health Administration Center (HAC), and Financial Service Center (FSC).

## Scope

This SDD specifies the technical details for the MCCF) EDI TAS project.

The work effort will create a modern Web application design and architecture that:

* Maintains compliance with Designated Standard Maintenance Organizations (DSMO) related to healthcare EDI transactions (see list of following specific EDI transactions).
* Ports existing functionality of the current EDI applications to a new, modern Web application.
* Enhances the capabilities of EDI transaction processing, including increasing the ability to “prepopulate” processing of all EDI transactions from service data.
* Provides services that will be used to implement/Improve the user interface for EDI transaction processing.
* Enhances service status management and monitoring capabilities and reporting (e.g., dashboards).

The specific EDI transactions in scope for this effort will include:

* 837 claims (837-I, 837-P, and 837-D)
* 835 electronic remittance advice (ERA)
* Electronic Funds Transfer (EFT)
* 270 (Eligibility, Coverage or Benefit Inquiry)/271 (Eligibility, Coverage or Benefit Information)
* B1 (Pharmacy Claim)/B2 (Pharmacy Claim Reversal)/B3 (Pharmacy Claim Rebilling)/E1 (Pharmacy Eligibility Notification)
* 278 Health Care Services Review – Request for Review and Response (278x217), Inquiry and Response (278x215)
* 277 Request for Additional Information
* 275 Additional Information to Support a Health Care Claim or Encounter (future)

# Background

## Overview of the System

VHA’s MCCF EDI operations need to effectively comply with the mandated requirements of the Administrative Simplification provisions in Section 1104 of the Patient Protection and Affordable Care Act (PPACA) of 2010 and the Health Information Technology for Economic and Clinical Health Act (HITECH). These mandated requirements are further specified in the 1996 HIPAA as amended by Public Law (P.L.) 111-148 PPACA, Section 1104.

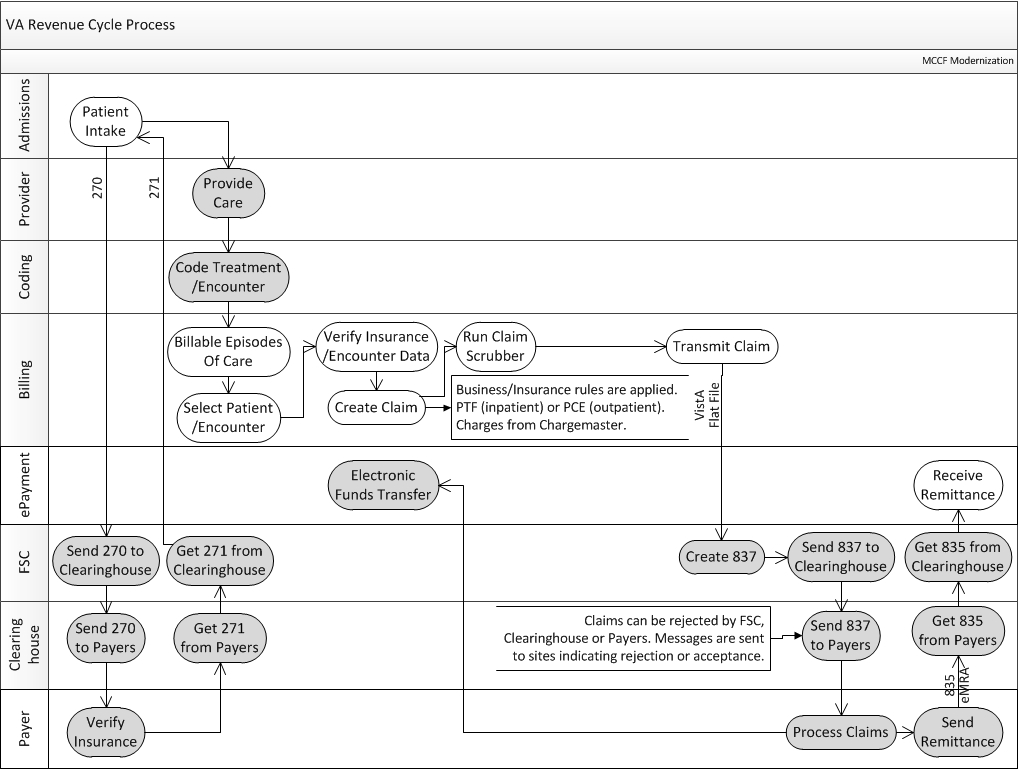
The eBusiness Solutions Office supports the MCCF EDI operations by assessing the impact of mandated healthcare regulatory requirements on these operations and the identification of needs to increase the percentage of reimbursements collected. As the VHA business process owner and Subject Matter Experts (SMEs) for industry health care EDI mandates, eBusiness Solutions defines the business needs and provides these needs to OIT Product Development (PD) and VA FSC for implementation of technology solutions that deliver both compliance and capabilities to increase reimbursements to VA for clinical services delivered to Veterans.

The IB module is used by Billing Supervisors and Clerks, and UR and Insurance Verification personnel. MCCF Modernization will provide a system with the functionality to address the full revenue lifecycle from insurance verification to claims to payment of the claims. For claims, the system will include the ability to generate auto populated (fully automated) claims, thus allowing more efficient generation of claims, more revenue, and allowing staff to work on individual claim issues. The system will include services that can also be used to deliver data to, and receive data from, a new Web application used by billing staff for claims that cannot be fully automated.

## Business Process Overview

The MCCF Modernization solution will automate the business processes used currently as part of the VA revenue cycle. This includes insurance verification, billing, and claims processing, payment, and remittance. These processes are tied to other processes that are out of scope, including documenting the care provided, coding treatment and encounters, and sending claims and receiving remittance to and from the clearinghouse. The figure that follows below, VA Revenue Cycle Business Process, shows the high-level business process starting from patient intake to receipt of payment for care that was provided. The other out-of-scope steps are displayed to show the overall context. The steps shaded in grey are out of scope for the MCCF Modernization effort.

Figure 1: VA Revenue Cycle Business Process



The process includes three steps:

1. Insurances are verified during the patient intake process.
2. After patient intake, care is delivered.
3. Once the care has been provided, the treatment or encounter is coded with the appropriate codes (ICD-10, CPT, etc.). For inpatient care, the Patient Treatment File (PTF) contains the documentation of the care. For outpatient care, the Patient Care Encounter contains the documentation of the care.

Some details regarding specific processes are detailed in the following sub-sections.

## ****High-level Claims Process****

1. Billing staff process third party claims using VistA IB, which is integrated with other VistA modules. The data used to process the claims comes from the inpatient and/or outpatient records within VistA.
2. When the billing staff has finished processing the claims, they authorize the claim for transmission to a third-party payer.
3. The claims are batch processed using site-specific settings, or the billing staff can manually send a claim using the IB option, [IBCE 837 MANUAL TRANSMIT] though this is not normal procedure.
4. The data for each claim is extracted from VistA and IB generates a flat file using the layout defined in Appendix A of the eBilling International Classification of Diseases (ICD), which also lists the FileMan file and field locations of each data element in VistA where the data is extracted from to create the flat file. Though the messages transmitted to FSC still contain some proprietary elements which are needed by Change Healthcare, the VA Healthcare Clearing House (HCCH) to print claims, the message content is modeled on the ASC X12 5010 standard and contains the data elements necessary for FSC to created compliant claims transmissions to electronic payers.
5. The flat files containing the claim data are sent to FSC using VistA Mailman messages.
6. FSC receives the Mailman messages and uses the Gentran system to translate the data in the flat file into a standard ASC X12N/005010 Health Care Claims (837) transmission, validates whether the data complies with HIPAA standards and then forwards the claim data to the VA Healthcare Clearing House (HCCH). NOTE: This step is out of scope for MCCF.
7. The HCCH transmits the 837 claims to payers who have agreements with the HCCH to receive electronic payments. If the payer cannot receive electronic claims, the claim is printed at the VAMC and mailed to the payer. NOTE: This step is out of scope for MCCF.
8. Once the payer processes the claim, data is sent from the payer to the HCCH, and then to FSC. Data is also returned to the sending VistA system from the FSC in Mailman messages. Refer to the ICD document X12 835 formats for EFT, ERA and MRA for details. NOTE: This step is out of scope for MCCF.

## ****High-level Request for Additional Information Process****

1. A claim is sent to a payer using the previously described process.
2. Once a payer receives a health care claim, it may decide that the basic electronic claim or paper claim does not contain enough information for its systems or personnel to determine how to adjudicate the claim. The payer can then place the claim in a pending status and send a 277 Request for Additional Information (RFAI) from the health care provider. NOTE: This step is out of scope for MCCF.
3. The HCCH receives the RFAI message, validates it and sends it to FSC. NOTE: This step is out of scope for MCCF.
4. The VLTrader system at FSC receives and validates X12N/5010 Health Care Claim Request for Additional Information (277RFAI) transactions received from the HCCH. The 277RFAI transactions are then converted to the Health Level Seven (HL7) v2 format and then delivered to the VAMC VistA site from which the original claim was sent to the payer. A Virtual Private Network (VPN) is used to communicate between FSC and the HCCH. FSC retains a copy of the HL7 transactions in a local Microsoft SQL Server relational database. NOTE: This step is out of scope for MCCF.
5. VistA IB receives the messages and places them on the RFAI Management Worklist (MRW) for the staff to process. The information used to respond to the RFAI comes from the patient clinical record and is usually provided by sending the payer clinical documentation and/or images.
6. Once the RFAI is processed by the staff, currently, the 277RFAI transactions are replied to manually (i.e., mail, Fax, email). In the future, the X12N/5010 Additional Information to Support a Health Care Claim or Encounter (275) transaction will be used to respond to 277RFAI transactions.

## ****High-level Pre-certification Process****

One of the standardize transactions for exchange of data is the ASC X12N Health Care Services Review – Request for Review and Response (278). The 278 transaction is designed to allow a provider to request authorization or certification of healthcare services from a Utilization Management Organization (UMO). Initiation of requests and receipt of responses are managed from within Claims Tracking. The 278 transaction is designed to support the following business events:

* Admission certification review requests and associated responses.
* Referral review requests and associated responses.
* Health care services certification review requests and associated responses.
* Extend certification review requests and associated responses.
* Certification appeal review requests and associated responses.
* Reservation of medical services review requests and associated responses.
* Cancellation of service reservations review requests and associated responses.

Certification review requests can be sent at many times including at or prior to registration for pre-certifications, or in association with referrals. The Claims Tracking module within VistA, is designed to be used by both Billing Personnel and UR staff. Regardless of when the request is sent, the process is the same and is outlined as follows.

1. VistA users (UR/RUR nurses) request authorization for healthcare events such as scheduling, admissions and clinic appointments for claims tracking events and manage insurance reviews and hospital reviews through the Claims Tracking module.
2. Entries are created on the Health Care Services Review (HCSR) Worklist.
3. RUR staff process entries on the HCSR Worklist.
4. The Integrated Billing module within VistA initiates the X12N 5010 HCSR/ Request for Review (278) messages sent to FSC.
5. The TLE at the FSC receives the HL7 278 Request message and translates the message into the proper X12 format and forwards the message on to the HCCH. NOTE: This step is out of scope for MCCF.
6. The X12N 5010 HCSR/ Request for Review (278) response message is sent back to FSC. NOTE: This step is out of scope for MCCF.
7. FSC translates the X12 formatted message back to HL7 and returns the HL7 formatted Response message to the originating VAMC.
8. The Integrated Billing module within VistA receives the X12N 5010 Health Care Service Review / Request for Review (278) response messages returned by the FSC and updates the Claims Tracking module.
9. Authorization for care numbers are then added to the claims creation process so that authorization numbers are submitted to the third-party payers as part of the claims.

## Overview of the Significant Requirements

### Architecture Platform Epics

The MCCF EDI TAS architecture platform will need to support the eBilling, eInsurance, ePayments, ePharmacy, and TASCore applications as defined in the User Stories for each of those work streams. To ensure that the MCCF EDI TAS architecture will meet this goal, a review of the User Stories for each of the work streams is performed regularly, and user stories are created for each work stream for the features and components in the Conceptual Architecture that would need to be implemented in the MCCF EDI TAS platform to enable the application functionality.

This review of the MCCF EDI TAS applications against the Conceptual Architecture validates that the components identified in that architecture will provide a platform that will enable the features and functionality defined for each MCCF EDI TAS application. It also provides traceability back to the requirements in the User Stories and will be used as input to guide future choices made for the components used in the MCCF EDI TAS Architecture layers. The Epics were also cross-referenced to the Veteran-focused Integrated Process (VIP) Compliance Epics, which are mapped to each MCCF EDI TAS Architecture Epic as acceptance criteria. The Epics that were created were then placed on a build plan to create a high-level sequence for the work.

The current set of epics and user stories are structured in a hierarchy with Initiatives at the highest level, Capabilities at the next level down, and Features at the level under Capabilities. User Stories are assigned to Features for the work needed to develop and deploy each Feature. The list below contains the Initiatives, Capabilities and Features defined by the eBusiness Solutions Office for MCCF EDI TAS. This SDD documents details for the Cloud First Initiative, so the hierarchy for that initiative is shown.

* I1 Maintain Compliance
* I2 Vista Enhancements
* I3 Cloud First Initiative (VA Enterprise Cloud)
  + C1 Reporting
    - F7 ECME OPECC Reports
    - F8 EDI Lockbox Reports
  + C2 Screens
    - F9 Insurance Company Editor
    - F19 ECME OPECC Screens
    - F47 eIV Linking & Activating Phase I
    - F60 ERA/APAR Worklist
    - F61 Receipt Profile Worklist
  + C3 Infrastructure
    - F32 VistA Data Access Infrastructure
    - F34 TAS Reporting Infrastructure
  + C7 No Touch
    - F10 eBilling No Touch
  + C8 Insurance Capture
    - F20 Electronic Insurance Coverage Discovery (EICD)
    - F50 Insurance Verification Phase I
  + C9 Financial Service Center (FSC) Interface
    - F11 837 Porting
    - F15 Interface with FSC – Inbound/Outbound 278
    - F28 277 STAT Porting
    - F29 278 Porting
    - F30 277 RFAI Porting
  + C11 Worklists
    - F37 ECME OPECC Workflow
  + C12 User/Role Management
    - F14 User Preferences
    - F16 User/Role Management
  + C13 Third Party Joint Inquiry (TPJI)
    - F17 Third Party Joint Inquiry (TPJI)
  + C14 Queue/Task Management
    - F18 Queue/Task Management
  + C16 Shared Component
    - F24 IAM Shared Component
    - F31 Non-VistA Database Shared Component
    - F33 TAS Services API Shared Component
    - F35 Rules Management System Shared Component
    - F51 TAS Portal Integrated Help
    - F52 Content Management System
    - F53 Worklist Management (Listman functionality)
    - F54 Site Search
    - F55 VistA File/Field Locking
    - F56 Artifact Repository

### Overview of the Functional Workload/Performance Requirements

The following data points are functional workload/performance requirements and statistics based on a comprehensive 2014 data pull.

* At maximum capacity; VA collects insurance at every visit, per patient, everywhere = averaging 40,000 entries per hour at peak
* More realistic capacity; VA collects insurance once per month, per patient, everywhere = averaging 16,000 entries per hour peak (data assumption: one patient has one appointment and one policy)
* Ideally each patient policy will contain a card image (front and back), which could result in a maximum of 350,000 images per day with typical 150,000 images per day Monday through Friday workweek
* Images are stored for 13 month rolling periods
* VA would not import/migrate existing card images to a new platform
* Approximately between 700 and 800 insurance verifiers work simultaneously
* Approximately between 8,000 and 10,000 insurance intake clerks work simultaneously
* There is an unknown number of “admin” only users, but approximately 300 to 500 exist who will not work simultaneously (Assumptions are two per facility and a few at each CPAC)
* An insurance verifier typically spends between three and five minutes verifying a policy
* An insurance intake clerk typically spends between one and two minutes scanning a card
* \* System availability is 24/7 but most patient appointments are scheduled events
* \* Times are ET, system would see typical user load begin at 7 am ET, peak at 11 am ET as all time zones are operational, then begin falling at 5 pm ET with minimal use 8 pm through 6 am next day.

Figure 2: Transactions per Hour when Insurance Collected at Each Visit

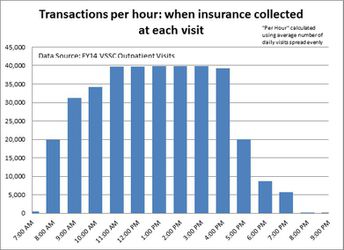
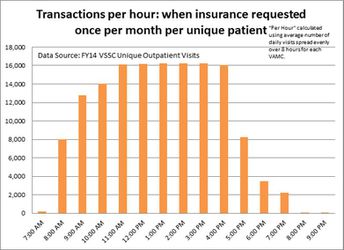


Figure 3: Transactions per Hour when Insurance Requested Once per Month per Unique Patient



The transaction volumes for FY 2016 are listed **Error! Reference source not found.**.

Table 1: FY 2016 Transaction Volumes

| FY 2016 Electronic Transaction Volumes | |
| --- | --- |
| eIV Inquiries Initiated | 9,241,808 |
| Primary & Secondary Claims: | 13,401,842 |
| NCPDP Payable Claims: | 108,283,375 |
| ERA 835: | 1,833,867 |
| EFTs Received: | 1,297,370 |

### Overview of Operational Requirements

#### Scalability

* MCCF EDI TAS allows for future functional requirements to be integrated into the system without major changes to the hardware and software package.
* MCCF EDI TAS will be fully integrated with existing systems within VistA and the VA Enterprise. VistA instances can be located locally at the facility or in remote data centers as either an integrated or a single instance per site database. Additionally, VistA instances can reside on servers running various operating systems including but not limited to Windows, VMS, and Linux running InterSystems Caché.

#### Availability

MCCF EDI TAS uses Web server availability strategies such as load balancing for failover and availability. Services will be deployed on the VDA platform, which include failure recovery features that are yet to be defined. MCCF TAS will be deployed into the Microsoft Azure GovCloud (MAG) environment for Production, which will include availability features and processes including synchronization across regions for Azure Storage and the Azure Load Balancer.

The MCCF EDI TAS operational requirements are as follows.

* The system shall provide sub-second performance for the execution of system connections and logic within the boundaries of the MCCF EDI TAS platforms.
* Maintenance including maintenance of externally developed software incorporated into the MCCF EDI TAS platforms shall use system (contingency/failover, etc.) redundancy to ensure that scheduled maintenance does not cause system down time.
* System implementation includes Continuity of Operations Plan (COOP)/Disaster Recovery (DR) and 24/7 support consistent with organizationally established expectations relative to system availability.
* A Continuity of Operations Plan will be created and provided to the technical and user community to follow if/when lapses in system availability occur despite the implementation of COOP/DR and 24/7 support arrangements.

#### Disaster Recovery (DR)

MCCF TAS will be deployed into the MAG environment for Production, which will include Disaster Recovery features and processes. MCCF EDI TAS will follow VA Enterprise application backup and restore procedures, including:

* Application servers
* Platform, code, and artifacts stored in source control
* Operational data supporting application server configurations
* DR will be implemented as per MAG and regional data center DR procedures and plans

### Architecture Timeline

The Build and Sprint schedule places the Epics, Sub-epics and User Stories from the previous section on a VIP Build Iteration timeline. The roadmap below shows the builds and the major feature release milestones associated with those builds.

1. Architecture Build 0 – Design/Infrastructure – (Timeline) (11/1/16-2/3/17)
2. Architecture Build 1 Architecture Platform Implementation – (Timeline) (2/6/17-4/28/17) (90-day VIP Cycle)
3. Architecture Build 2 VistA Data Access, DevOps, TAS Architecture Foundation – (Timeline) (5/1/17 - 7/21/17) (90-day VIP Cycle)

Due to the consolidation of the Architecture Team and the eAdmin team to TASCore, the architecture build schedule was realigned and resulted in TASCore Build 1, which started 7/3/2017.

1. TASCore Build 1 – TAS Portal and ATO (Timeline) (7/3/17 - 9/22/17) (90-day VIP Cycle)
2. TASCore Build 2 – ePayment Reporting, VistA Data Access Services, IAM Integration (Timeline) (9/25/17 – 12/15/17) (90-day VIP Cycle)
3. TASCore Build 3 – FSC ICDs, ePayment Reporting, VistA Data Access Services, IAM Integration (Timeline) (12/18/17 – 3/9/18) (90-day VIP Cycle)

# Conceptual Design

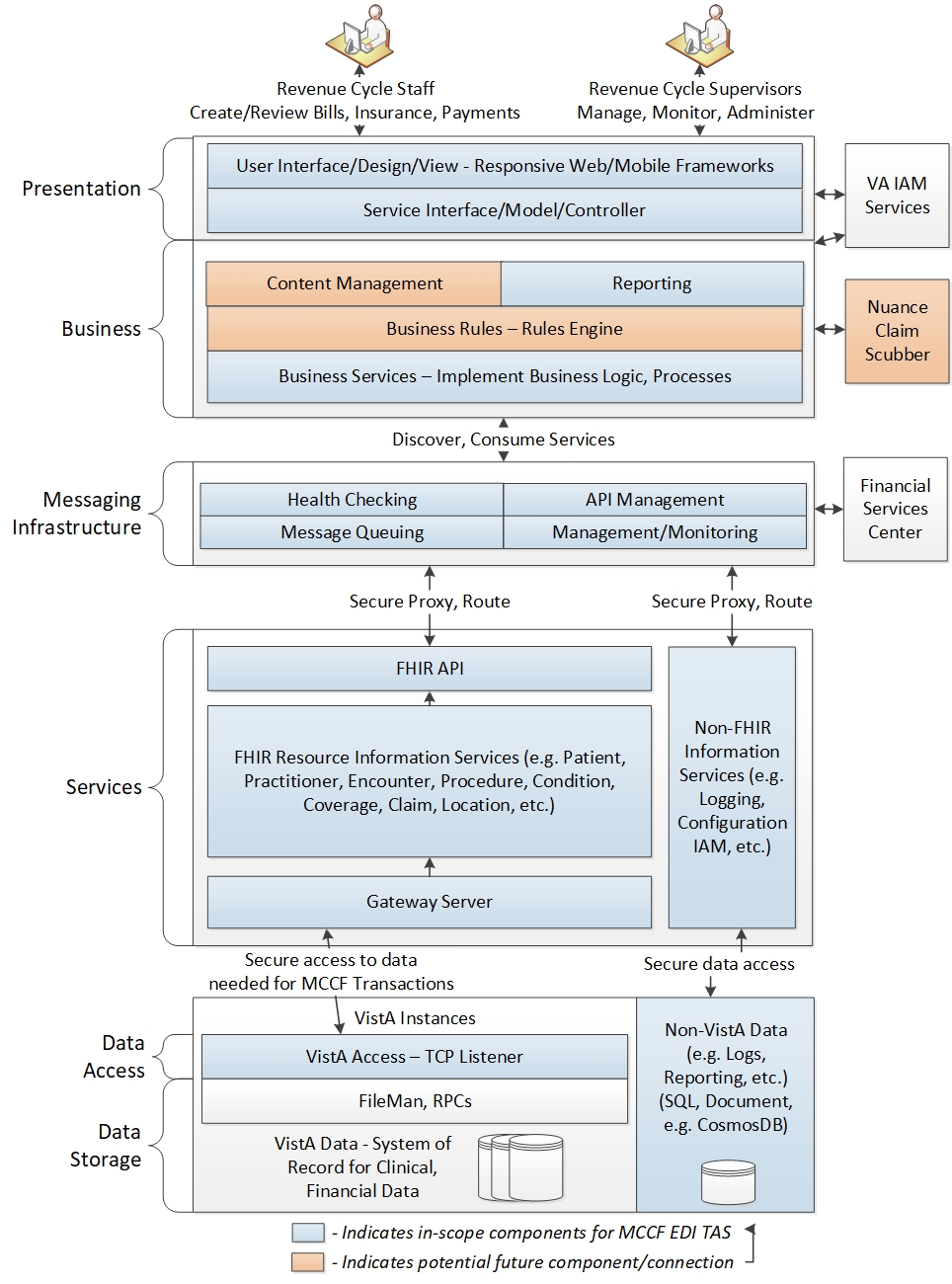
## Conceptual Application Design

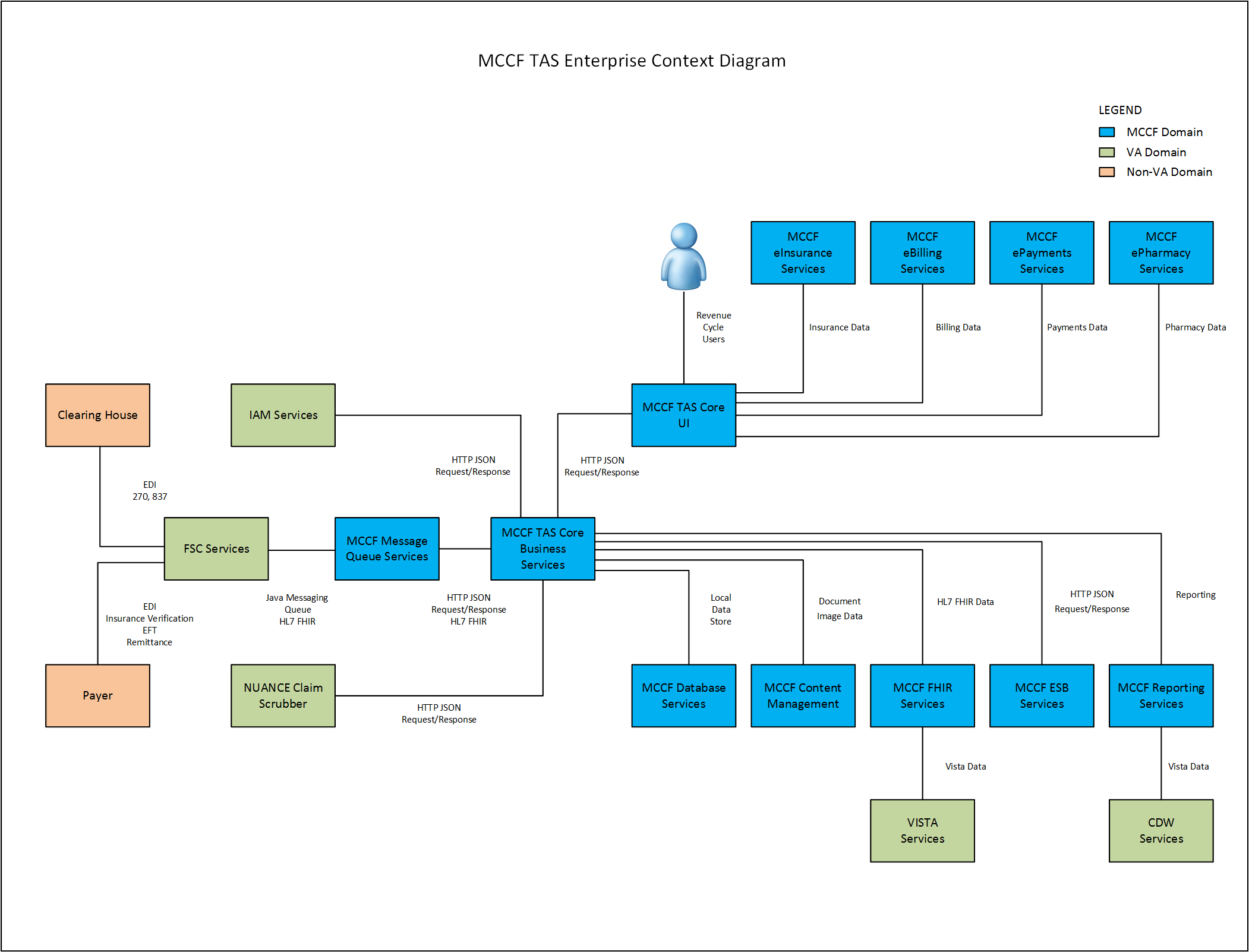
Figure : MCCF EDI TAS Conceptual Architecture

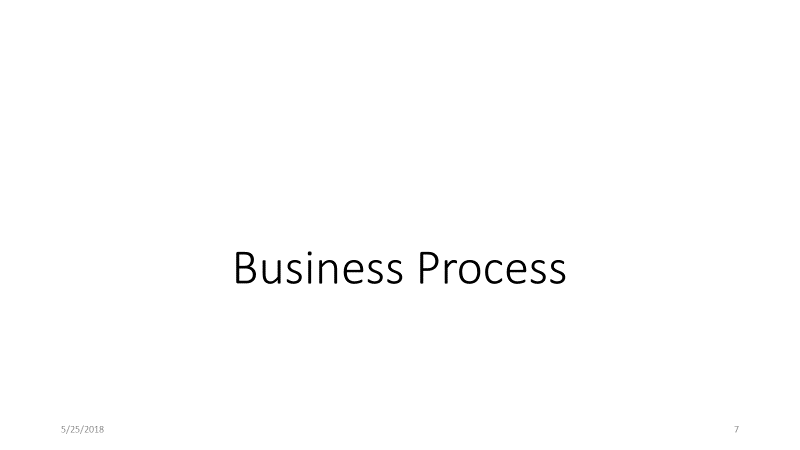
Figure 4 shows architecture components for MCCF EDI TAS based on technologies and products currently available. The components in scope for MCCF EDI TAS are color coded. Those components not in scope for MCCF represent dependencies and potential risks. Some of the dependencies can potentially be removed, and the associated risks mitigated, if MCCF EDI TAS implements these components within the MCCF EDI TAS implementation. MCCF EDI TAS will work with the other VA projects to determine if the components will be available within the timeframe needed and will determine whether each component should be implemented within MCCF EDI TAS to meet project requirements. One capability that will be implemented on an interim basis until the VA Enterprise capability is available is the VistA Data Access Services.

For the diagrams, potential technologies or products that will be used to implement the various components of the architecture are included in the Software Architecture section. The list includes details for current implementations, future planned implementations, and implementation by MCCF EDI TAS. All technologies or products listed, except as noted, are on the VA TRM.

Figure 4: MCCF EDI TAS Conceptual Architecture







### User Profiles

The different user types that will use TAS are listed below. These will be used to define roles and permissions within the TAS web application.

The current VistA Integrated Billing (IB) module is used by Billing Supervisors, Billing Clerks, and Utilization Review (UR) Nurses. VistA support personnel are responsible for monitoring status of the service interface as part of their normal job functions. The FSC receives claims, sends the claims to payers through a clearinghouse and sends remittance advice back for those claims. The MCCF Modernization service architecture will enable Billing Supervisors and Clerks, and UR Nurses to perform the same functions as they would in the VistA IB module through a new Web application user interface, which will be a consumer of the MCCF Modernization services.

#### TASCore User Types

The following user types were defined during the work performed to complete US1648 – UI Architecture: Develop User Types. Existing documentation for eBilling, eInsurance, ePayments, eAdmin, and ePharmancy was used as a basis for researching existing user types. The outcome of US1648 was reviewed by the business teams.

Table - User Type by eBusiness Team

| User Type | User Permissions |
| --- | --- |
| Insurance Intake Team | One or more of the following:  Read-Only  Contributor  Make changes to documents (not the information but the content in the actual document) |
| Insurance Verification Team | One or more of the following:  Read-Only  Contributor  Make changes to documents (not the information but the content in the actual document) |
| Utilization Review Team | One or more of the following:  Read-Only  Contributor  Make changes to documents (not the information but the content in the actual document) |
| Billing Team | One or more of the following:  Read-Only  Contributor  Make changes to documents (not the information but the content in the actual document) |
| Pharmacist Team | One or more of the following:  Read-Only  Contributor  Make changes to documents (not the information but the content in the actual document) |
| OPECC Team | One or more of the following:  Read-Only  Contributor  Make changes to documents (not the information but the content in the actual document) |
| Account Receivable Team | One or more of the following:  Read-Only  Contributor  Make changes to documents (not the information but the content in the actual document) |
| eBusiness Solution Staff Team | Administrative Privileges (eBusiness Solutions Staff)  Upload documents, links and dynamic content  Tag metadata and taxonomy information  Set priority (Sort)  Add, Edit or Delete Content Items  Manage user access/permissions  Ability to run reports |

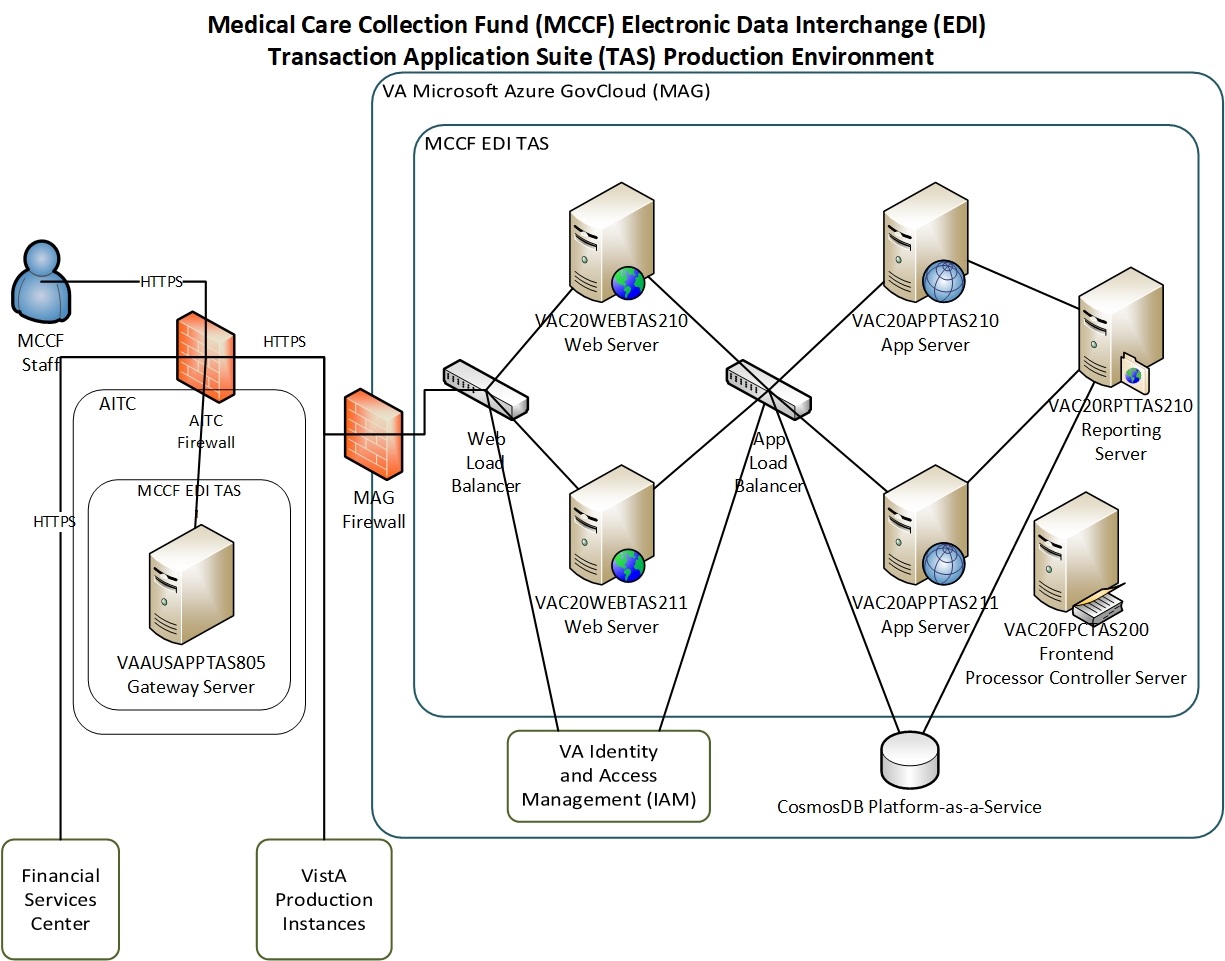
### Application Locations

#### Identified Systems

* Web application servers that host the MCCF EDI TAS application components will be located at the MAG Data Centers
* Database servers that host the non-VistA data will be located at the MAG Data Centers
* The receiving/sending system for EDI transaction data and messaging is at the FSC in Austin.
* The data access and storage are the VistA instances at VAMCs.

The figure below, Servers Hosting the Software Components, shows the primary MCCF EDI TAS non-VistA components of this environment. IAM/SSOi integration implements components in the Apacheserver in each of the MCCF environments.

Figure 5: Servers Hosting the Software Components



Non-VistA storage components are implemented in the CosmosDB instances in MAG. All other software components will be installed on the Web application servers.

### MCCF EDI TAS Automated Software Installation and Configuration

The software packages that will be used and the environments where each will be used are listed in the table below.

The servers located in each environment are defined in Section 6.3

If a software package is required to be used in an environment, all other related environments must also incorporate that package prior to the promotion process. For example, if a package is needed in production, it will also be needed in all other environments.

Automated software installation and configuration will be implemented using Vagrant and Ansible in the development team environment and on developer workstations. In the VA MAG, automated software installation and configuration will be implemented using Ansible.

Below is the list of packages for each environment. Details for each tool including descriptions and links to the tool web site are included on the VA TRM site.

| Tool | Environment | | MCCF Version | | TRM Version | TRM ID | TRM Date | TRM Waiver Status | MCCF Review Date |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Health Level 7 (HL7) Application Programming Interface (API)- Fast Healthcare Interoperable Resources (FHIR) | Production | | 2.2 | | 2.x | 8870 | 06/02/2017 | Approved | 12/06/2017 |
|  |
| NPM | Production | | 5.x | |  |  |  | Not Required, INCLUDED IN FRAMEWORK | 11/02/2017 |
| Postman | Development | |  | |  |  |  |  |  |
|  |  | |  | |  |  |  |  |  |
| zone.js | Production | | ^0.7.2 | |  |  |  | Not Required, INCLUDED IN Angular FRAMEWORK | 11/22/2017 |
| errorhandler | production | | 1.5.x | |  |  |  | Submitted to PMO CM | 11/22/2017 |
|  |  | |  | |  |  |  |  |  |
| Font Awesome | Production | | 4.7.x | | 5.x | 10342 | 11/29/2017 | Approved w/Constraints | 12/01/2017 |
| primeNG | Production | | 4.2 | | 4.x | 11566 | 11/27/2017 | Approved w/Constraints | 12/01/2017 |
| TypeDoc | Production | | 4.2 | | 0.8.0 | 11563 | 11/27/2017 | Approved w/constraints | 12/01/2017 |
| azure-storage-node | Production | | 2.6.x | |  |  |  | Submitted to PMO CM | 12/01/2017 |
|  |  | |  | |  |  |  |  |  |
| AngularJS | Production | | 2.4 | | 4.0.x | 7842 | 06/02/2017 | Approved w/Constraints | 12/06/2017 |
| Bower | Production | | 1.8.x | | 1.8.x | 8183 | 08/02/2017 | Approved with constraints | 12/06/2017 |
| core-js | Production | | ^2.4.1 | | 2.4.x | 10276 | 11/30/2016 |  | 12/06/2017 |
| TypeScript | Production | | 2.1 | | 2.x.x | 9883 | 05/30/2017 | Approved | 12/06/2017 |
| Ansible | Production | | 2.4 | | 2.3 | 8196 | 10/25/2017 | submitted to PMO CM | 12/06/2017 |
| Apache HTTP Server | Production | | 2.4.6-45 | | 2.4.x | 5009 | 06/19/2017 | Approved w/Constraints | 12/06/2017 |
| Apache Tomcat | Production | | **7.0.76** | | 9.x | 5451 | 02/14/2017 | Divest | 12/06/2017 |
| Application Insights for Node.js | Production | | 0.21.x | | 0.21.x | 11328 | 09/11/2017 | Approved | 12/06/2017 |
| axe-core | SQA | | 2.3.1 | | 2.x | 11518 | 10/25/2017 | Approved | 12/06/2017 |
| axe-webdriverjs | SQA | | 1.1.x | | 1.1.x | 11334 | 09/11/2017 | Approved | 12/06/2017 |
| Body-Parser | Production | | **1.16** | | 1.15.x | 8242 | 06/24/2016 | **\*\*\* This entry has been marked as ARCHIVED \*\*\*** | 12/06/2017 |
| Bootstrap | Production | | 3.3.7 | | 3.3.x | 7795 | 06/21/2017 | Approved with constraints | 12/06/2017 |
| CentOS Linux | Production | | 7 | | 7.x | 6460 | 05/03/2017 | Approved w/Constraints | 12/06/2017 |
| Debug | Production | | 3.1.x | | 2.3.x | 8967 | 12/13/2016 | Approved w/Constraints | 12/06/2017 |
|  |  | |  | |  |  |  |  |  |
|  |  | |  | |  |  |  |  |  |
| Embedded JavaScript Templating (EJS) | Production | | 2.5.5 | | 2.5.x | 10913 | 06/07/2017 | Approved | 12/06/2017 |
| Express JS | Production | | 4.14 | | 4.x | 7729 | 11/30/2016 | Not Required | 12/06/2017 |
| Google Chrome | Production | | **60.x** | | 58.x | 5618 | 11/23/2016 | Approved | 12/06/2017 |
| Google GSON | Production | | 2.7.x | | 2.7.x | 7348 | 11/23/2016 | Not Requested | 12/06/2017 |
| Groovy | Production | | 2.4.9 | | 2.4.x | 9262 | 11/10/2016 | Not Requested | 12/06/2017 |
| Grunt | Production | | 1.0.1 | | 1.0.x | 8162 | 05/28/2017 | Approved | 12/06/2017 |
| JavaScript (JS) Beautifier | Production | | 1.6.x | | 1.6.x | 11329 | 09/11/2017 | Approved | 12/06/2017 |
| JMESPath JS | Production | | 0.15.0 | | 0.15.x | 11497 | 10/25/2017 | Approved | 12/06/2017 |
| jQuery | Production | | **3.1.1** | | 2.2.x | 6706 | 06/28/2016 | **\*\*\* This entry has been marked as ARCHIVED \*\*\*** | 12/06/2017 |
| jsdom | Production | | 10.1.0 | | 10.x, 11.x | 11176 | 07/28/2017 | Approved | 12/06/2017 |
|  |  | |  | |  |  |  |  |  |
|  |  | |  | |  |  |  |  |  |
|  |  | |  | |  |  |  |  |  |
| Morgan | Production | | 1.7 | | 1.7.x | 8256 | 06/24/2016 | **\*\*\* This entry has been marked as ARCHIVED \*\*\*** | 12/06/2017 |
|  |  | |  | |  |  |  |  |  |
| Nexus | Production | | 3.5.2 | | 3.5.x | 6411 | 09/15/2017 | Approved w/Constraints | 12/06/2017 |
| NGINX | Production | | 1.11 | | 1.12.x | 6605 | 06/20/2017 | Approved w/Constraints | 12/06/2017 |
| ngx-clipboard | Production | | 8.0.x | | 8.0.x | 11327 | 09/11/2017 | Approved | 12/06/2017 |
| ngx-datatable | Production | | 10.0.x | | 10.0.x | 11333 | 09/11/2017 | Approved | 12/06/2017 |
| Node.js | Production | | 8.x | | 6.x | 6716 | 08/18/2017 | Approved w/constraints | 12/06/2017 |
| node-rules | Production | | 3.1.0 | | 3.1.x | 10857 | 05/29/2017 | Approved | 12/06/2017 |
| PM2 | Production | | 2.4 | | 2.4.x | 10869 | 05/30/2017 | Approved | 12/07/2017 |
| Request | Production | | 2.83.x | | 2.73.x | 8033 | 11/17/2017 | **\*\*\* This entry has been marked as ARCHIVED \*\*\*** | 12/07/2017 |
| Simple Logging Facade for Java (SLF4J) | Production | | 1.7.21 | | 1.7.x | 7735 | 01/18/2017 | Approved w/Constraints | 12/07/2017 |
| socket.io | Production | | 2.0.x | |  | 8947 | 04/13/2017 | **\*\*\* This entry has been marked as ARCHIVED \*\*\*** | 12/07/2017 |
| Swagger User Interface (UI) | Development | | 2.1.x | | 3.1.x | 8231 | 09/15/2017 | Approved w/Constraints | 12/07/2017 |
| Vagrant | Production | | 1.9.x | | 1.9.x | 7306 | 07/31/2017 | Approved w/Constraints | 12/07/2017 |
| winston | Production | | 2.4.x | | 2.2.x | 9857 | 06/30/2016 | Archived | 12/07/2017 |
|  |  | |  | |  |  |  |  |  |
| Open Web Application Security Project (OWASP) Dependency-Check | Development (Jenkins) | | 3.0.1 | | 1.4.x | 8298 | 04/18/2017 | Approved w/Constraints | 12/07/2017 |
|  |  | |  | |  |  |  |  |  |
| angular-mocks | Development | | ^1.5.0 | | **Not on TRM** |  |  | Not Required, INCUDED IN FRAMEWORK | 10/25/2017 |
| check-dependencies | Development | | **1.1.0** | |  |  |  | Submitted to PMO CM | 11/22/2017 |
|  |  | |  | |  |  |  |  |  |
| grunt-nodemon | development | | 0.4.x | |  |  |  | Submitted to PMO CM | 11/22/2017 |
| angular-cli | Development | | 1.0.0-beta.28.3 | | 1.x | 11153 | 07/28/2017 | Approved | 12/06/2017 |
| Atom | Development | | 1.22.1 | | 1.x.x | [7721](http://trm.oit.va.gov/ToolPage.aspx?tid=7721) | 04/03/2017 | Not Requested | 12/06/2017 |
| dependency-check | Development | | **2.8** | |  |  |  | submitted to PMO CM | 12/06/2017 |
| Eclipse Classic | Development | | **4.6.3** | | 4.6 | 6316 | 11/17/2016 |  | 12/06/2017 |
| Git | Development | | 2.11 | | 2.11.x | 6396 | 12/08/2016 | Not Required | 12/06/2017 |
| Maven | Development | | 3.3.9 | | 3.3.x | 1101 | 11/23/2016 | Not Required | 12/06/2017 |
| MUnit | Development | |  | | 7.3.x | 10284 | 11/30/2016 | Approved w/Constraints |  |
|  |  | |  | |  |  |  |  |  |
| RxJS | CIT | | 5.1.0 | | **Not on TRM** |  |  | Not Required, INCUDED IN FRAMEWORK. | 11/22/2017 |
| sinon | CIT | | 2.2.0 | | **Not on TRM** |  |  | Not Required, INCUDED IN FRAMEWORK | 11/22/2017 |
|  |  | |  | |  |  |  |  |  |
| Codelyzer | CIT | | 3.1.x | | 3.1.x | 11175 | 07/28/2017 | Approved | 12/06/2017 |
| JUnit | CIT | | 4.12 | | 4.12 | 38 | 07/25/2017 | Approved w/constraints | 12/06/2017 |
| junit-viewer | CIT | | 4.11.1 | | 4.x | 11181 | 07/28/2017 | Approved | 12/06/2017 |
|  |  | |  | |  |  |  |  |  |
| mocha-junit-reporter | CIT | | 1.13.0 | | 1.13.0 | 11126 | 07/25/2017 | Approved | 12/06/2017 |
| mocha-logger | CIT | | 1.0.5 | | 1.0.x | 11174 | 07/28/2017 | Approved | 12/06/2017 |
|  |  | |  | |  |  |  |  |  |
| SoapUI | CIT | | **5.3.0** | | 5.3.x | 6252 | 06/19/2017 | Approved w/Constraints |  |
| tslint | CI | | ^4.1.1 | | **Not on TRM** |  |  | Not Required, INCUDED IN FRAMEWORK. | 11/22/2017 |
|  |  | |  | |  |  |  |  |  |
| webpack | CI | | ^1.12.6 | | **Not on TRM** |  |  | Not Required, INCLUDED IN FRAMEWORK. | 11/22/2017 |
| engine.io | CI | | 3.1 | | 3.1.x | 8931 | 11/27/2017 | Approved w/Constraints | 12/01/2017 |
| Protractor | CI | | 5.1.2 | | 5.2.x | 11589 | 11/28/2017 | Approved w/Constraints | 12/012017 |
| Selenium | CI | | 2.20.0 | | 3.6.x | 6440 | 11/27/2017 | Not Required | 12/01/2017 |
| Karma | CI | | 1.1.x | | 1.1.x | 8882 | 07/28/2016 | Not Requested |  |
| Chai | CI | | 3.5.0 | | 3.5.x | 10221 | 11/29/2016 | Not Required | 12/06/2017 |
| cross-os | CI | | 1.1.2 | | 1.1.x | 11520 | 10/25/2017 | Approved w/Constraints | 12/06/2017 |
| JAVA | CI | | 8 (aka 1.8) | | 8 (aka 1.8) | 8 | 06/19/2017 | Not Required | 12/06/2017 |
| Jenkins Continuous Integration Server | CI | | 2.73.1 | | 2.x | 6397 | 11/17/2016 | Not Required | 12/06/2017 |
| Mocha | Production | | 3.5.x | | 3.2.x | 8248 | 06/02/2017 | submitted to PMO CM | 12/06/2017 |
|  |  | |  | |  |  |  |  |  |
| node-json2html | CI | | 1.1.1 | | 1.x | 11519 | 10/25/2017 | Approved w/Constraints | 12/06/2017 |
| PhantomJS-prebuilt | CI | | 2.7.1 | | 2.1 | 11564 | 11/27/2017 | Further Inquiry To Be Made | 12/07/2017 |
| protractor-jasmine-2-html-reporter | CI | | 0.0.7 | | 0.0.7 | 11545 | 10/30/2017 | Approved | 12/07/2017 |
| selenium-standalone | CI | | 6.0.1 | | 6.x.x | 10860 | 05/29/2017 | Approved | 12/07/2017 |
| U.S. Web Design Standards Framework | Production | | 1.1.0 | | 1.x.x | 11196 | 07/28/2017 | Approved | 12/07/2017 |
| VirtualBox | CI | | 5.1.x | | 5.1.x | 6679 | 09/09/2016 | Not Required | 12/07/2017 |
| Zed Attack Proxy (ZAP) | CI | | 2.6.0 | | 2.6 | 8271 | 10/30/2017 | Approved w/Constraints | 12/07/2017 |
| Micro Focus - HP Fortify Static Code Analyzer | CI | | 17.10 | | 17.2 | 6429 | 12/05/2017 | Approved | 12/08/2017 |
| ts-node |  | 3.0.4 | | **Not on TRM** |  |  | Not Required, INCLUDED IN FRAMEWORK. | 11/22/2017 |

## Conceptual Data Design

### Project Conceptual Data Model

MCCF EDI TAS will use FHIR STU version 3 as the target data model. This aligns with the work that was done by the VistA API 2.0 Team. Details of the FHIR resource requirements for these transactions is described in the following sections.

#### FHIR Resources Needed for MCCF EDI TAS

The set of FHIR resources needed for processing of claims, payments, and insurance in the MCCF EDI TAS user interface, in reports, and in transactions to FSC are listed below. The columns to the right of the FHIR resource indicate whether that resource is needed for the reports, screens and transactions in that product team.

Table - MCCF EDI TAS FHIR Resources by Product Line

| MCCF EDI TAS FHIR Resources | eBilling | ePayments | eInsurance | ePharmacy | TASCore |
| --- | --- | --- | --- | --- | --- |
| Terminology/value list service |  |  |  |  | X |
| Account FHIR resource/service |  | X |  | X |  |
| Claim FHIR resource/service | X | X |  | X |  |
| ClaimResponse FHIR resource/service | X |  |  | X |  |
| CodeSystem FHIR resource/service |  |  |  |  | X |
| Communication FHIR resource/service | X | X |  | X |  |
| ConceptMap FHIR resource/service |  |  |  |  | X |
| Condition FHIR resource/service | X | X | X | X |  |
| Coverage FHIR resource/service | X | X | X | X |  |
| Device FHIR resource/service |  |  |  |  | X |
| DiagnosticReport FHIR resource/service |  |  | X |  |  |
| DocumentResource FHIR resource/service |  |  |  |  | X |
| EligibilityRequest FHIR resource/service |  |  | X |  |  |
| EligibilityResponse FHIR resource/service |  |  | X |  |  |
| Encounter FHIR resource/service | X | X | X | X |  |
| EnrollmentRequest FHIR resource/service |  |  | X |  |  |
| EnrollmentResponse FHIR resource/service |  |  | X |  |  |
| EpisodeOfCare FHIR resource/service | X | X |  | X |  |
| ExpansionProfile FHIR resource/service |  |  |  |  | X |
| ExplanationOfBenefit FHIR resource/service | X | X |  | X |  |
| HealthcareService FHIR resource/service | X | X |  | X |  |
| Location FHIR resource/service | X | X | X | X |  |
| Media FHIR resource/service |  |  |  |  | X |
| MedicationOrder FHIR resource/service | X | X |  | X |  |
| MessageHeader FHIR resource/service | X | X | X | X |  |
| NamingSystem FHIR resource/service |  |  |  |  | X |
| OperationOutcome FHIR resource/service |  | X | X |  |  |
| Organization FHIR resource/service | X | X | X | X |  |
| Patient FHIR resource/service | X | X | X | X |  |
| Person FHIR resource/service | X |  |  |  |  |
| Practitioner FHIR resource/service | X | X | X | X |  |
| Procedure FHIR resource/service | X | X | X | X |  |
| ProcessRequest FHIR resource/service | X | X |  | X |  |
| ProcessResponse FHIR resource/service | X | X |  | X |  |
| ReferralRequest FHIR resource/service | X | X |  | X |  |
| RelatedPerson FHIR resource/service | X | X | X | X |  |
| PaymentReconciliation FHIR resource/service |  | X |  |  |  |
| PaymentNotice FHIR resource/service |  | X |  |  |  |
| ValueSet FHIR resource/service |  |  |  |  | X |

### User Interface Data Mapping

#### Application Screen Interface

##### 3.2.2.1.1 ****Mapping IB screens to FHIR resources****

In addition to needing all the data elements required to generate the data for each transaction for transmission to FSC, the MCCF EDI TAS will need to present all the data needed by MCCF staff to process transactions. It is assumed that the MCCF EDI TAS will use the current IB screens as a baseline for functionality. The data required for the IB screens will be used to define the set of FHIR resources that will be needed to deliver data to the MCCF EDI TAS user interface for claims processing. The mapping performed by the API 2.0 team maps the IB screens and data elements to FHIR resources. Mapping for other screens will be performed as the user stories are defined for those screens.

The mapping file below from the API 2.0 SDD starts with the list of IB screens and fields and maps those to FHIR resources as well as to VistA files and fields and the 837 flat file data elements in the eBilling ICD.



The FHIR resources identified as being needed to deliver the data elements for the IB screens and fields used by billing staff to process claims are:

1. Claim
2. ClaimResponse
3. Coverage
4. Encounter
5. EpisodeOfCare
6. HealthcareService
7. MedicationOrder
8. Organization
9. Patient
10. Practitioner
11. Procedure

## Conceptual Infrastructure Design

### System Criticality and High Availability

The MCCF EDI TAS follows the standard procedures used for all critical systems. For example:

* System backups
* VistA disaster recovery

In addition, the high availability features within Azure such as synchronizing across MAG regions will be leveraged to ensure that the TAS application is available. More details can be found in the MCCF EDI TAS Information Systems Continuity Plan (ISCP) and Disaster Recovery Plan (DRP).

### Special Technology

Details of the technologies used in MCCF EDI TAS are included in the MCCF EDI TAS Software Packages table earlier in this document. MCCF EDI TAS modernization will use current TRM approved packages. In cases where newer versions are required or versions that are not on the TRM are required, we will submit a justification for waiver or evaluation in parallel to our development, to the TRM group. Once approved, we will provide the technology baseline update in a future VIP build of the modernization to align with our desired architectural goals.

### Technology Locations

The table in section 3.1.2 shows the components used in MCCF EDI TAS and the locations where each component will be located as well as their TRM status.

### Conceptual Infrastructure Diagram

The following two figures, Test Environment Conceptual Infrastructure Diagram, Part 1. and Test Environment Conceptual Infrastructure Diagram, Part 2 show the architecture of the development and test environments used for MCCF EDI TAS. The first diagram shows both the Development Team Amazon Web Services (AWS) Virtual Private Cloud (VPC) and the VA MAG environments that will be used. The second diagram shows the details of the AWS environment that will be used for development of the MCCF EDI TAS.

Figure 6: Test Environment Conceptual Infrastructure Diagram, Part 1

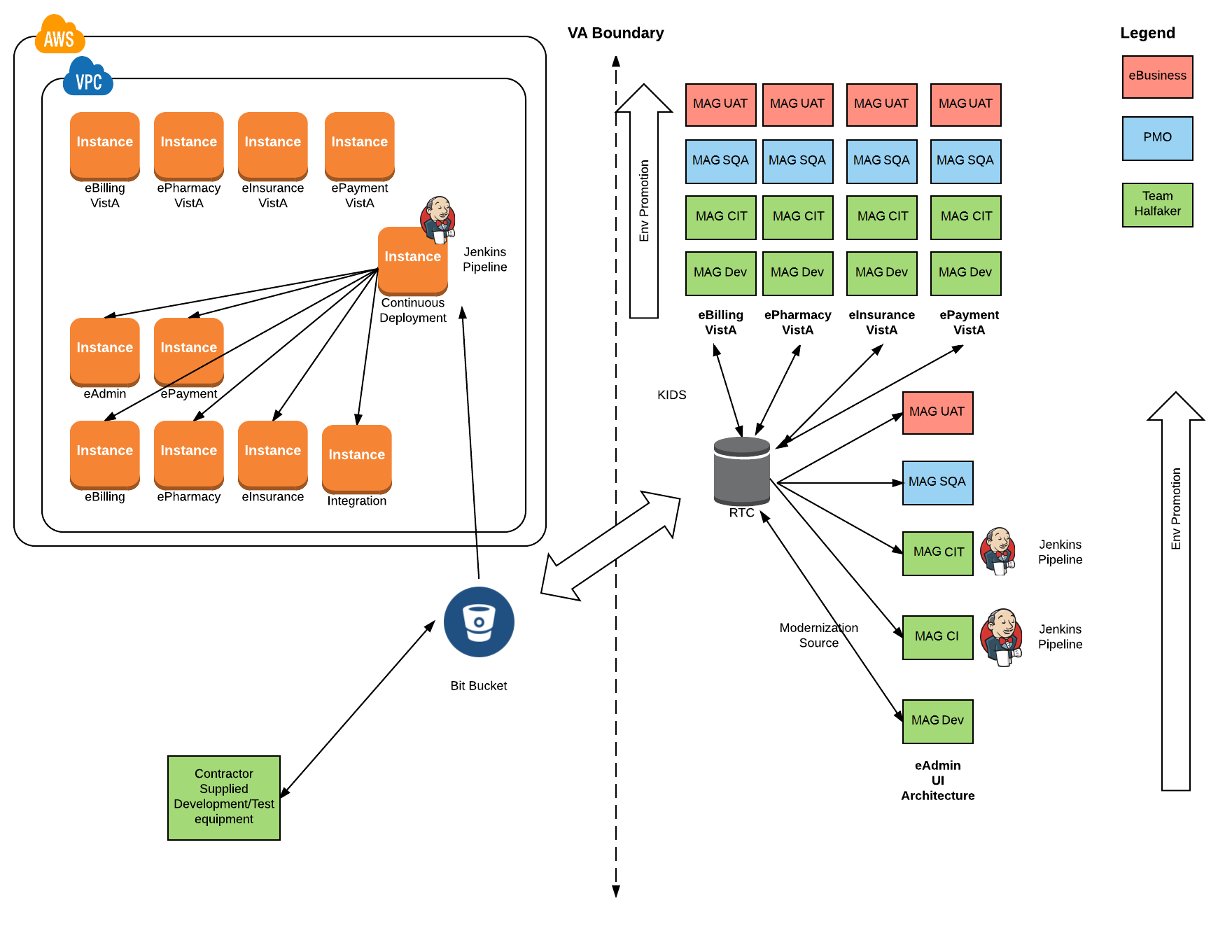
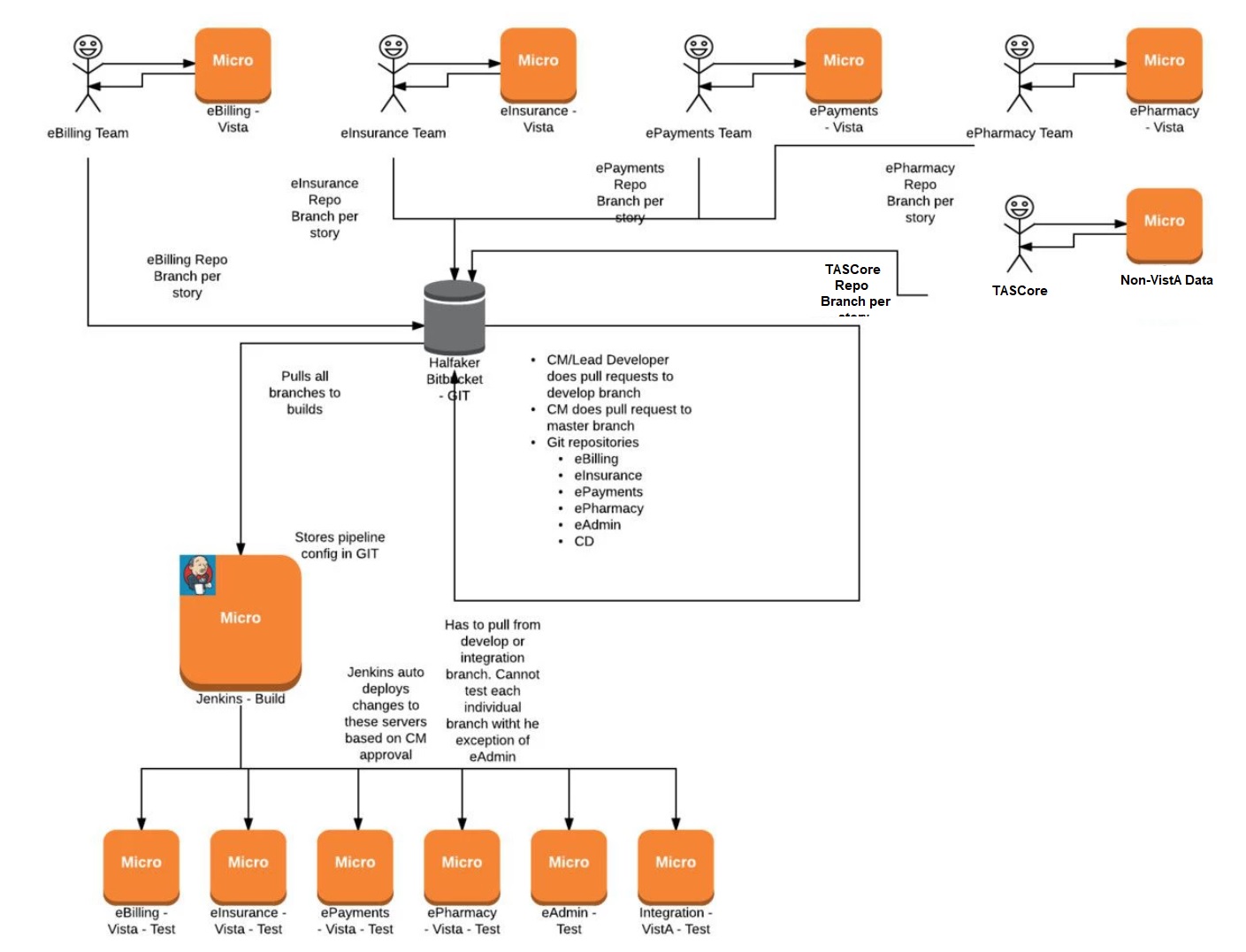


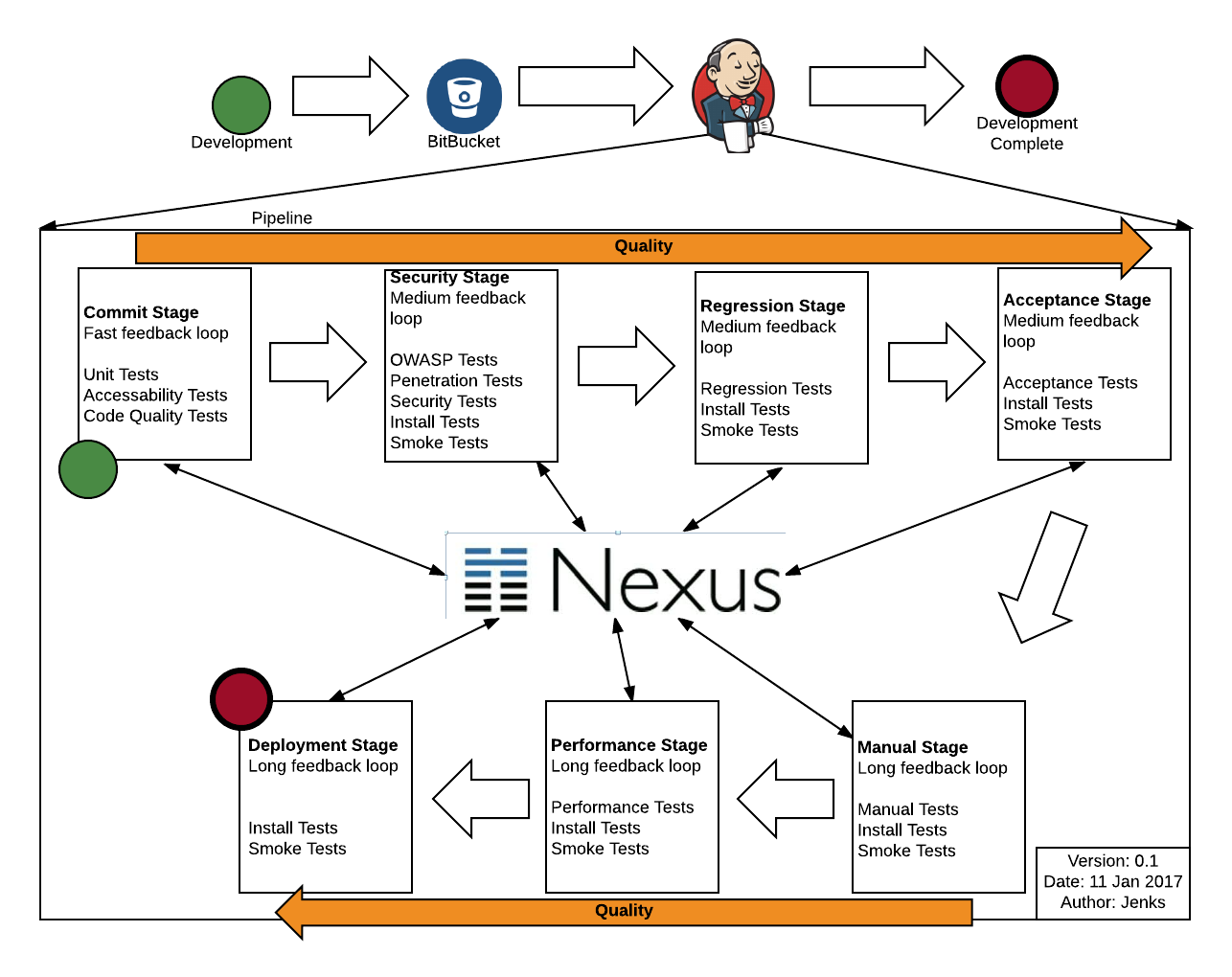
Figure 7: Test Environment Conceptual Infrastructure Diagram, Part 2



#### Conceptual Production String Diagram

The diagram below shows the process used to deploy the TAS application into production. The process shows moving from development through to application build and deployment. The quality tests performed at various stages in the process are shown as is the Nexus artifact repository that will be used to manage production deployment artifacts.

Figure 8: Conceptual Production String Diagram



# System Architecture

## Hardware Architecture

MCCF EDI TAS provides a Web application that reads and writes data to existing VA VistA instances distributed throughout the VA Enterprise, so the overall architecture is distributed. The architecture takes advantage of consolidation of VistA instances into regional data centers. MCCF EDI TAS is based on node.js and as such runs on a platform that supports node.js, including Red Hat Linux. MCCF EDI TAS components will be deployed on a standard VA baseline Red Hat Linux image. The TAS web application components will be deployed in MAG for national use, and the MUMPS code needed to enable VistA Data Access will be deployed in the VA VistA instances.

MCCF EDI TAS components will run effectively on virtual platforms provided by VA using VA baseline images. Existing VA data protection and backup techniques will be effective for the MCCF EDI TAS platform. Capacity planning will be performed using the transaction volumes included in section 2.6.2. The hardware architecture allows for scaling by adding servers to handle additional load.

## Software Architecture

The software used for each component in the MCCF EDI TAS Architecture is listed in the table in section 3.1.2.

For the MCCF EDI TAS architecture, VistA data access is available to the Business and Presentation layers, as well as other consumers such as FSC, via a FHIR API implementation, which delivers FHIR resources using a FHIR Profile, and enables all search, query, and fetch operations defined in the FHIR specification.

MCCF EDI TAS Package Management Process

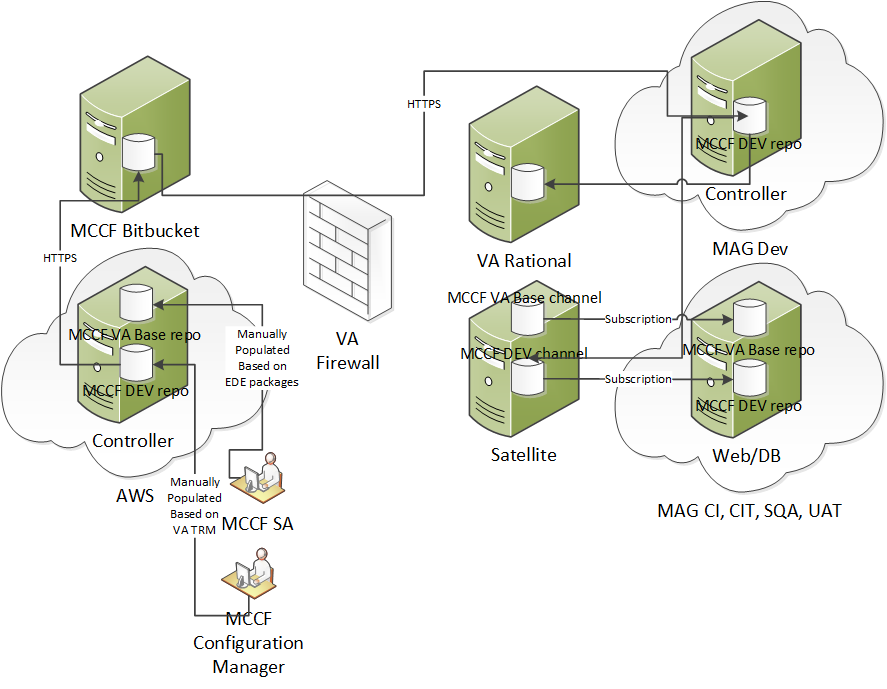
#### MCCF VA Base Packages

1. MAG version availability is confirmed by the MCCF System Administrator (SA)
2. AWS Controller server MCCF VA Base repository is populated by the MCCF SA based on the MAG availability
3. Dev MAG Controller server subscribes to existing Satellite channel(s) for environment packages
4. Satellite channel based on Dev MAG Controller MCCF VA Base repository is used for provisioning within CI, CIT, SQA, and UAT MAG environments

#### MCCF DEV Packages

1. Packages are identified on the baseline MCCF EDI TAS configuration
2. TRM version availability is confirmed by the MCCF Configuration Manager
3. RPMs are created for NPMs
4. RPMs for TRM approved versions are manually added to the MCCF DEV repo on the AWS controller by the MCCF Configuration Manager
5. Packages in the MCCF DEV repo are checked in to Bitbucket
6. Packages are synched from Bitbucket to Dev MAG Controller
7. Packages from Dev MAG Controller MCCF DEV repo are checked into RTC
8. Satellite server updates MCCF DEV channel from Dev MAG Controller MCCF DEV repo
9. Dev MAG Controller server subscribes to Satellite channel(s) for MCCF DEV packages
10. Satellite channel based on Dev MAG Controller MCCF DEV repository is used for provisioning within CI, CIT, SQA, and UAT MAG environments

Figure 9 - MCCF EDI TAS Package Management Process



### Node and Angular Shared Components

Creating shared modules using node and Angular will allow common features or functionality to be created once and reused by all product teams for MCCF EDI TAS. Common UI components such as the TAS eheader and footer will be created using Angular. Common services likewise will be created as shared components using node.

## Network Architecture

MCCF EDI TAS relies on existing VA local area networks (LANs) and Wide Area Networks (WANs). Connections are over VA-standard implementations of HTTP/HTTPS and TCP. Logical and Physical network isolation through firewalls and port restrictions are not affected by MCCF EDI TAS. As the VA configures TCP ports to run MCCF EDI TAS services, these TCP ports and endpoints will need to be configured in network isolation devices and software. The MAG environment enables communication between components within each MAG environment as well as between MAG regions. TAS will make use of this internal MAG infrastructure as well as the connection between MAG and the VA network. Detailed diagrams and tables depicting the communications path(s) between the system and subsystem modules are included in sections 6.3 and 7.1 of this document.

## Service Oriented Architecture / ESS

MCCF EDI TAS includes, and is based on, the following Architectural Characteristics:

* Layered Architecture with abstraction at each layer.
* Web Application using Angular SPA in the Presentation Layer.
* Web Services API Architecture using Node.js in the Services Layer.
* Standards-based Microservices architecture compliant with OpenAPI Specification.

VistA Data Access components implemented within TAS enable aggregated access to all VA VistA instances via an HL7 FHIR API.

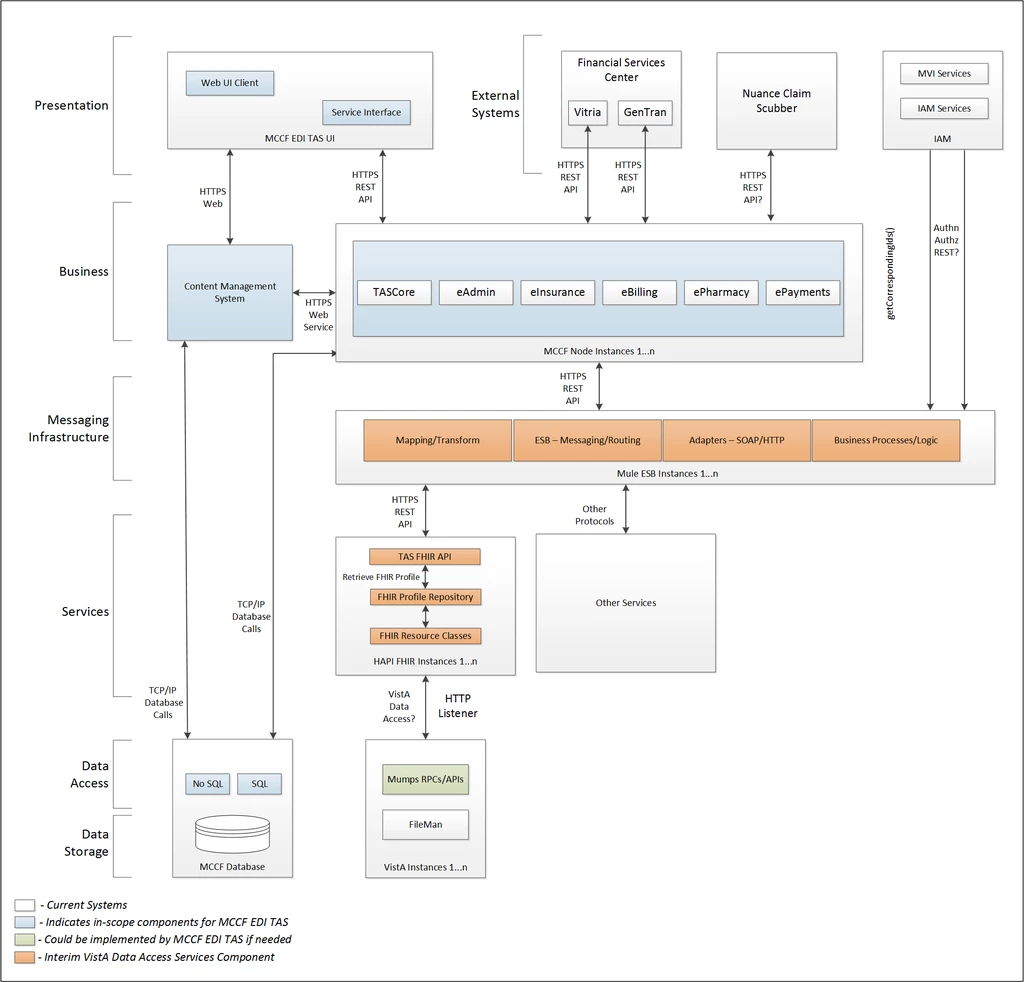
The diagram below shows architecture components for MCCF EDI TAS based on technologies and products currently included in the TAS technical baseline.

. The components in scope for MCCF EDI TAS are color coded. Those components not in scope for MCCF represent dependencies and potential risks. Some of the dependencies can potentially be removed, and the associated risks mitigated, if MCCF EDI TAS implements these components within the MCCF EDI TAS implementation.

One example of this is the VistA Data Access VDA services needed for TAS. TAS will implement VistA Data Access services within the TAS system boundary and will use those until a VA Enterprise VistA Data Access service solution is made available. The MCCF EDI TAS team will work to ensure the interim TAS VDA implementation will align with a future VA Enterprise VDA solution.

The MCCF EDI TAS team will work with the other VA projects to determine if the components will be available within the timeframe needed and if each component should be implemented within MCCF EDI TAS to meet project requirements. All technologies or products listed, except as noted, are on the VA TRM.

Figure 10: MCCF EDI TAS Logical Architecture



## Enterprise Architecture

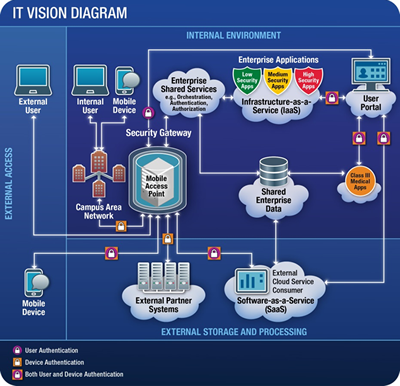
The VA Enterprise Architecture standards, strategies, and guidelines establish the fundamental technologies enabling the VA to meet its business and information system goals. By using these standards, the VA can:

* Promote interoperability, portability, and adaptability within systems
* Promote quality assurance
* Place the VA in a position to use current technology
* Provide a framework for IT application and infrastructure development.

The MCCF EDI TAS will employ the VA Enterprise Architecture as described in the OneVA Enterprise Technical Architecture. MCCF EDI TAS will use a layered approach, which includes services for VistA and non-VistA data access and for implementing specific functionality and capabilities. This approach allows for separation of the UI from the business logic and the data.

The VA IT Vision Diagram from the Office of Technology Strategies (TS) below shows how Enterprise Shared Services access Shared Enterprise Data and are used by Enterprise Applications to provide information to users. Enterprise Shared Services are also critical to interoperability and provide access to information for external partners and applications.

Figure 11: VA Future IT Vision Diagram (5-year)



The MCCF EDI TAS team is working with Enterprise projects including IAM, to ensure that application is implemented following the VA governance and implementation guidelines, and adhere to VA Enterprise Design Patterns, including the Service-Oriented Architecture, Enterprise Cloud Service Management, and Microservices patterns[[1]](#footnote-2). The VA has established design patterns to guide the design and development of Web services.

The MCCF EDI TAS will only use software that is approved by the VA TRM.

MCCF EDI TAS VistA Data Access services will be deployed in the TAS architecture. The VDA services are intended to provide a set of core services for access to data in existing VA VistA instances distributed throughout the VA Enterprise, so the overall architecture will be distributed. The architecture takes advantage of consolidation of VistA instances into regional data centers.

VA Enterprise services used within TAS include:

* Authorization/Authentication (leveraging IAM)
* Basic patient services (leveraging MVI) to provide patient and domain lookups
* RPC services

# Data Design

This section describes the design of the database management system (DBMS) and non-DBMS files associated with MCCF EDI TAS.

## DBMS Files

MCCF EDI TAS uses the existing VA FileMan and Cache databases and their underlying global and file structure for the EDI Screens, Reports and Transaction Processing.

For non-EDI transactions, a NoSQL database will be used to store data. This will primarily be associated with the TASCore features and functionality in the system (such as logging and content management) as well as security data including user, role and permission information.

## Non-DBMS Files

MCCF EDI TAS has not identified the need to store non-DBMS files at this time. All MCCF EDI TAS data will be stored in VistA or the non-VistA data storage.

## Data View

MCCF EDI TAS is using HL7 FHIR Release 3 resources as the data model for the X12 EDI transactions. HL7 had considered a change request to add mappings from FHIR resources to the X12 transactions as part of the FHIR specification, and said that mapping was underway, but they have not included that in the current Release 3 version and have said that it will be considered for future use <http://gforge.hl7.org/gf/project/fhir/tracker/?action=TrackerItemEdit&tracker_item_id=7705>.

As previously described, some of the capabilities in MCCF EDI TAS that will require VistA Data Access include screens, reports, and interfaces with FSC that will be used to transmit HIPAA EDI transactions. The detailed data standards, formats, objects, and elements that will be used in MCCF EDI TAS transactions. for the HIPAA EDI transactions are included in the Interface Control Documents (ICDs) for each of the transactions. The ICDs are available at https://vaww.oed.portal.va.gov/pm/hape/ipt\_5010/EDI\_Portfolio/TAS%20Interim%20Repository/Forms/AllItems.aspx

The mappings of MCCF EDI TAS Reporting Data Elements from FHIR to VistA FileMan are included below. These will be updated with additional mappings for reports and screens throughout the development of MCCF EDI TAS.

### MCCF EDI TAS Report Data Element Mappings

The mappings included below are for two of the report features defined for MCCF EDI TAS. They include mapping for the report data elements from FHIR Resources to VistA FileMan files and fields.

#### EDI Lockbox Report Data Element Mappings



#### ECME OPECC Report Data Element Mappings



# Detailed Design

## Hardware Detailed Design

Node.js applications will operate on a VA-standard build hardware server configured with RedHat Linux. RedHat will be at the most current VA-supported version. Node is a Web server/app server. It was designed for a high volume of concurrent users and demanding workload. CPU intensive jobs were not the primary criteria for the design, but process clustering supports modern multi-core CPUs.

Load and performance testing is being performed to validate the hardware design. When results from performance testing are available additional RAM or upgraded processors may be implemented to meet performance requirements. The following is implemented:

Table - MCCF EDI TAS Server Specifications

| Servers | Qty | CPUs | RAM | OS Version | OS Storage | Data Storage |
| --- | --- | --- | --- | --- | --- | --- |
| DEV Web/App | 1 | 4 | 16 GB | RHEL 7.3 | 60 GB | 100 GB |
| DEV Web/App | 1 | 4 | 16 GB | RHEL 7.3 | 60 GB | 100 GB |
| DEV Database | 1 | 4 | 16 GB | RHEL 7.3 | 60 GB | 100 GB |
| CI Web/App | 1 | 4 | 16 GB | RHEL 7.3 | 60 GB | 100 GB |
| CI Web/App | 1 | 4 | 16 GB | RHEL 7.3 | 60 GB | 100 GB |
| CI Database | 1 | 4 | 16 GB | RHEL 7.3 | 60 GB | 100 GB |
| CIT Web/App | 1 | 4 | 16 GB | RHEL 7.3 | 60 GB | 100 GB |
| CIT Database | 1 | 4 | 16 GB | RHEL 7.3 | 60 GB | 100 GB |
| SQA Web/App | 1 | 4 | 16 GB | RHEL 7.3 | 60 GB | 100 GB |
| SQA Database | 1 | 4 | 16 GB | RHEL 7.3 | 60 GB | 100 GB |
| UAT Web/App | 1 | 4 | 16 GB | RHEL 7.3 | 60 GB | 100 GB |
| UAT Database | 1 | 4 | 16 GB | RHEL 7.3 | 60 GB | 100 GB |
| DEV-UAT NFS | 1 | 4 | 32 GB | RHEL 7.3 | 60 GB | 800 GB |
| Pre-Prod Web | 1 | 4 | 16 GB | RHEL 7.3 | 60 GB | 100 GB |
| Pre-Prod Web | 1 | 4 | 16 GB | RHEL 7.3 | 60 GB | 100 GB |
| Pre-Prod App | 1 | 4 | 16 GB | RHEL 7.3 | 60 GB | 100 GB |
| Pre-Prod App | 1 | 4 | 16 GB | RHEL 7.3 | 60 GB | 100 GB |
| Pre-Prod Database | 1 | 4 | 32 GB | RHEL 7.3 | 60 GB | 200 GB |
| Production Web | 1 | 4 | 16 GB | RHEL 7.3 | 60 GB | 100 GB |
| Production Web | 1 | 4 | 16 GB | RHEL 7.3 | 60 GB | 100 GB |
| Production App | 1 | 4 | 16 GB | RHEL 7.3 | 60 GB | 100 GB |
| Production App | 1 | 4 | 16 GB | RHEL 7.3 | 60 GB | 100 GB |
| Production Database | 1 | 4 | 32 GB | RHEL 7.3 | 60 GB | 200 GB |
| Production NFS | 1 | 4 | 32 GB | RHEL 7.3 | 60 GB | 800 GB |

## Software Detailed Design

### Conceptual Design

The subsections below contain the details of the MCCF EDI TAS Conceptual Design.

#### Presentation Layer Design

MCCF TAS employs a web application architecture that includes an Angular Single Page Application (SPA) for presentation layer and services implementation. The detailed design in this section describe specifics about the MCCF TAS software implementation.

**Angular**

“Lazy Loading” is angular2’s way of delivering components or pages only when they are called. This saves memory and makes the system faster.

The first part of Lazy Loading is make sure components are independent of the main page. This type of component is called “shared component”.

**Creating a shared Components**

There are 3 parts to shared components. Modules, which are a collection of Components, must be defined first. Second, Components, comprise the main building block, and Routes are the third part to shared components

These three parts taken together allow the html to find the component.

1. Create a new module scaffolding by running ng g module [NewModuleName]

2. In module directory, create a new Component by running ng g component

[NewComponentName]

3. Make a module shared by replacing the imported BrowserModule with CommonModule inside module ts file

4. Import new module into any desired parent modules (ng seems to do this automatically)

5. Add new module and component routes to their parent routing arrays

**Making Lazy Loaded Components**

(https://angular-2-training-book.rangle.io/handout/modules/lazy-loading-module.html)

lazy, we are going to lazy load a module conveniently called LazyModule. Look closely at the

definition of that route:

{ path: 'lazy', loadChildren: 'lazy/lazy.module#LazyModule' }

There are a few important things to notice here:

1. We use the property loadChildren instead of component.

2. We pass a string instead of a symbol to avoid loading the module eagerly.

3. We define not only the path to the module but the name of the class as well.

There's nothing special about LazyModule other than it has its own routing and a component

called LazyComponent.

app/lazy/lazy.module.ts

import { NgModule } from '@angular/core';

import { LazyComponent } from './lazy.component';

import { routing } from './lazy.routing';

@NgModule({

imports: [routing],

declarations: [LazyComponent]

})

export class LazyModule {}

If we define the class LazyModule as the default export of the file, we don't need to define the

class name in the loadChildren property as shown above.

The routing object is very simple and only defines the default component to load when

navigating to the lazy path.

app/lazy/lazy.routing.ts

import { ModuleWithProviders } from '@angular/core';

import { Routes, RouterModule } from '@angular/router';

import { LazyComponent } from './lazy.component';

const routes: Routes = [

{ path: '', component: LazyComponent }

];

export const routing: ModuleWithProviders = RouterModule.forChild(routes);

Notice that we use the method call forChild instead of forRoot to create the routing object. We

should always do that when creating a routing object for a feature module, no matter if the

module is supposed to be eagerly or lazily loaded.

Finally, our LazyComponent is very similar to EagerComponent and is just a placeholder for

some text.

app/lazy/lazy.component.ts

import { Component } from '@angular/core';

@Component({

template: '<p>Lazy Component</p>'

})

export class LazyComponent {}

View Example: https://plnkr.co/edit/vpCqRHDAj7V6mlN1AknN?p=preview

When we load our application for the first time, the AppModule along with the AppComponent

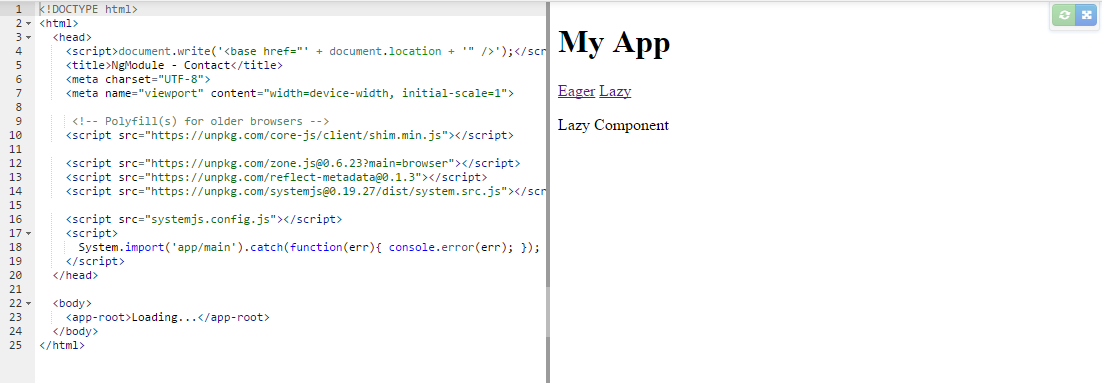
will be loaded in the browser and we should see the navigation system and the text "Eager

Component". Until this point, the LazyModule has not being downloaded, only when we click

the link "Lazy" will allow the needed code to be downloaded and we will see the message "Lazy

Component" in the browser.

Figure 12 - UI Lazy Loading Sample Web Page

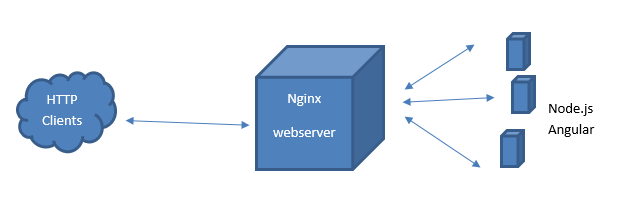


#### Services Layer Design

The diagram below shows the design for the load-balanced node.js web servers implemented in MCCF TAS. It shows the Virtual Machines for a load-balanced node.js web system.

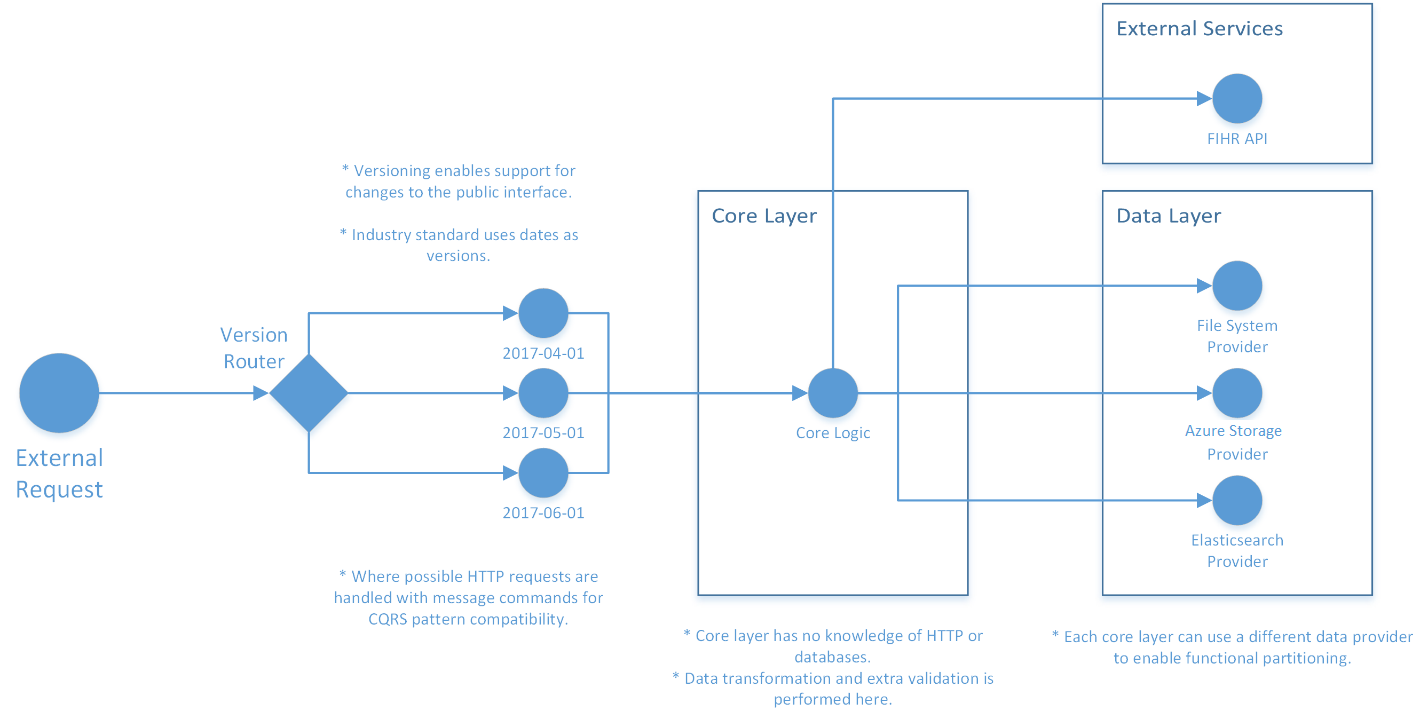
ref: <https://www.keithcirkel.co.uk/load-balancing-node-js/>

Figure - MCCF EDI TAS Load Balanced node.js Web System



The figure below shows the services design for MCCF TAS. Versioning is implemented through the version router, which enables support for multiple versions to be used at the same time so that new versions can be implemented without downtime. The Core Layer can access external services such as the planned FHIR API for VistA Data Access, as well as data sources in the data layer.

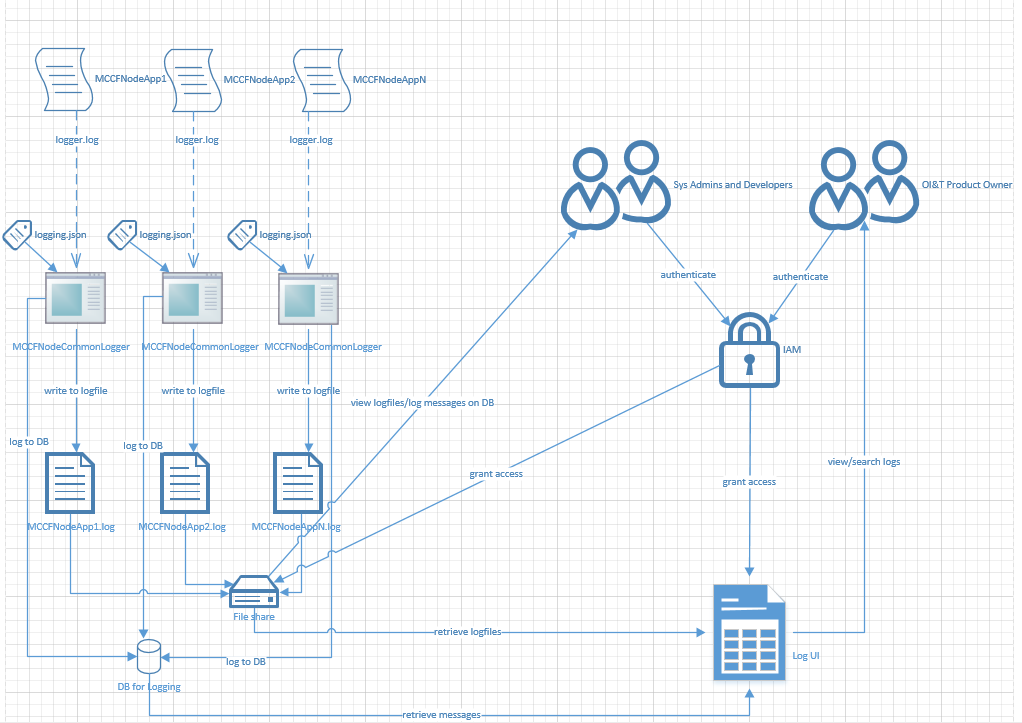
Figure - MCCF EDI TAS Services Design

****

**Logging in Node.js for MCCF EDI TAS**

The following diagram shows the different components that are needed to implement logging for Node.js applications and their relationships.

Figure 15 - MCCF TAS node.js Logging Components



**Logging Features:**

* Common Logging capability for MCCF TAS applications
* Ability to configure:
  + Log levels
  + Log level styles
  + Log locations
* Integration with IAM
* Integration with MCCF TAS notification/alerting system
* Log UI capability
  + search

**MCCF TAS Common Logging Component for Node.js applications**

A Node.js module named “MCCFNodeCommonLogger” will be developed. It will be used as a dependency by MCCF TAS Node.js modules for logging purposes. Thus, each MCCF TAS Node.js module will have its own instance of MCCFNodeCommonLogger.

Each module that uses the MCCFNodeCommonLogger will provide a configuration file (logging.json) that will be in the logging folder in the module.

The config file will be used by MCCFNodeCommonLogger and defines the following:

* Application name
* Log level for the application,
* Log level style – npm or syslog
* URI for the MongoDB instance where logfiles will be stored
* Path and name of the log file (one per log level)
* Logfile where unhandled exceptions will be logged to
* If logging will take place in a logfile on the fileshare or in the database (db or file)

This way, log level, log level style and the path to the log file/URI for the logging database can be changed without a new deployment.

An example of the logging.json could look like the following:

***[***

***{***

***“appName”: “nameOfApp”,***

***“logLevel”: “info”,***

***“logLevelStyle”: “npm”,***

***“filenameINFO”: “MCCFApp1INFO.log”,***

***“filenameWARN”: “MCCFApp1WARN.log”,***

***“filenameDEBUG”: “MCCFApp1DEBUG.log”,***

***“filenameERROR”: “MCCFApp1ERROR.log”,***

***“filenameExceptions”: “MCCFApp1Exceptions.log”,***

***“filepath”: “//server/path/to/logfile”,***

***“dbUri”: “mongodb://user:pass@host:port/MCCFApp1”,***

***“errorLoggingMethod”: “db”,***

***“infoLoggingMethod”: “ file”,***

***“warnLoggingMethod”: “file”,***

***“debugLoggingMethod”: “file”***

***}***

***]***

A log level could be

* one of the *syslog* logleves: emerg, alert, crit, error, warning, notice, info, or debug or
* one of the *npm* log levels: error, warn, info, verbose, debug, or silly

MCCFNodeCommonLogger will provide two methods called “**log**”.

One method will have two parameters (Log level and Message). (used by non Angular apps)

i.e. ***MCCFNodeCommonLogger.log(“error”, “Error Message”)****.*

The other method will have three parameters (Log Level, App Identifier, and Message (used to log Angular apps).

i.e. ***MCCFNodeCommonLogger.log(“error”, “angularApp1”, “Error Message”)****.*

The file logging.json will be read every time the .log method is called

MCCFNodeCommonLogger will use the NPM module “winston” to impelement the actual logging feature (<https://www.npmjs.com/package/winston>).

A log message created by winston will look like the following:

***{"level":"info","message":"this is a test","timestamp":"2017-05-11T14:13:11.006Z"}***

Not only will MCCFNodeCommonLogger provide logging when the log method is called, it will also provide logging in case of an uncaught exception (winston provides this feature and it can be implemented using the winston.handleExceptions function).

The following is an example of a message created in the exception logfile by winston after an unhandled exception occurred:

***{***

***"date":"Thu May 11 2017 10:50:48 GMT-0400 (Eastern Daylight Time)",***

***"process":{***

***"pid":15712,***

***"uid":null,***

***"gid":null,***

***"cwd":"C:\\Users\\smaer\\bitbucketrepos\\mccf-sandbox\\loggingtest",***

***"execPath":"C:\\Program Files\\nodejs\\node.exe",***

***"version":"v6.10.3",***

***"argv":[***

***"C:\\Program Files\\nodejs\\node.exe",***

***"C:\\Users\\smaer\\bitbucketrepos\\mccf-sandbox\\loggingtest\\log.js"***

***],***

***"memoryUsage":{***

***"rss":21315584,***

***"heapTotal":10522624,***

***"heapUsed":4795968,***

***"external":89982***

***}***

***},***

***"os":{***

***"loadavg":[***

***0,***

***0,***

***0***

***],***

***"uptime":7883.7613615***

***},***

***"trace":[***

***{***

***"column":1,***

***"file":"C:\\Users\\smaer\\bitbucketrepos\\mccf-sandbox\\loggingtest\\log.js",***

***"function":"",***

***"line":5,***

***"method":null,***

***"native":false***

***},***

***{***

***"column":32,***

***"file":"module.js",***

***"function":"Module.\_compile",***

***"line":570,***

***"method":"\_compile",***

***"native":false***

***}***

***],***

***"stack":[***

***"ReferenceError: conole is not defined",***

***"    at Object.<anonymous> (C:\\Users\\smaer\\bitbucketrepos\\mccf-sandbox\\loggingtest\\log.js:5:1)",***

***"    at Module.\_compile (module.js:570:32)",***

***"    at Object.Module.\_extensions..js (module.js:579:10)",***

***"    at Module.load (module.js:487:32)",***

***"    at tryModuleLoad (module.js:446:12)",***

***"    at Function.Module.\_load (module.js:438:3)",***

***"    at Module.runMain (module.js:604:10)",***

***"    at run (bootstrap\_node.js:390:7)",***

***"    at startup (bootstrap\_node.js:150:9)",***

***"    at bootstrap\_node.js:505:3"***

***],***

***"level":"error",***

***"message":"uncaughtException: conole is not defined",***

***"timestamp":"2017-05-11T14:50:48.639Z"***

***}***

MCCFNodeCommonLogger will only log a message if the log level defined in the file logging.json and the log level in the log message match.

There will be a log file for each log level that is defined in the file logging.json. The filename will be in the format “Appname-Loglevel.log” i.e. “MCCFApp1-INFO.log”. Winston transports can be used to implement this feature.

The logfiles will be stored on a yet to be defined file share and the module/machine that writes the log files will need write access to this file share.

For each log level there will also be a transport for logging into the database.

**Fileshare for Log Files**

The fileshare where the logfiles will be stored will have a folder for each application that writes logfiles. Each application folder will have a folder named “archive” where logfiles will be moved once they are rotated.

Rotation interval and/or file size quota will have to be defined.

Rotation of files will be done by the server that manages the fileshare. (set up by operations)

Access to logfiles and the fileshare will only be possible by select people (System Administrators, Developers, and Product Owners). An authentication/authorization concept will have to be developed. Utilization of IAM will have to be researched during development.

**Logging DB**

There will be a database for each application for logging. There could also be one database with multiple collections, one for each application. This will have to be decided.

**MCCF Node Apps**

Each MCCF Node.js app that uses the MCCFNodeCommonLogger will need to add it to its dependencies and will provide a logging.json file.

**Log UI**

There will be a UI for viewing the logfiles. It will be specified in chapter 6.2.1.1.1.

Each Node.js app (i.e. MCCFNodeApp1) has a dependency to the shared module “MCCFNodeCommonLogger”. Each Node.js app provides a file called logging.json and utilizes the .log method for logging purposes.

Depending on the configuration in logging.json, the logfile gets written to the file share, which rotates and archives the logfiles or the log message written to the database.

System administrators and developers will have access to the file share/database and will be able to view logfiles/log messages for their applications. Nobody else will have access to the file share/logfiles. Thus, before accessing the file share, the user will have to be authenticated via IAM. The user will also need to authenticate himself/herself to be able to access the database.

Product owners will be able to view the logfiles in the log UI. They will also have to be authenticated in order to log into the log UI (IAM). They will only be able to see data which they have access to.

**Logging for Angular applications**

Angular dependency injection will allow a developer to get an instance of the LoggerService from anywhere in the application.

See below:

*constructor(private configService: ConfigService, private loggerService: LoggerService) { }*

Early in the Angular boot-process (e.g. app.component.ts), the log level and the output provider will need to be set:

*this.loggerService.setlogLevel(this.configService.load(Config.LogLevel))*

*this.loggerService.setOutputProvider(BrowserConsoleOutputProvider)*

In this example, the log level is set by the config service.

Later, at any point in Angular, the LoggerService can be sent information regarding various log levels:

*this.loggerService.error("(sample error)", "(app id)")*

*this.loggerService.warn("(sample warn)", "(app id)")*

*this.loggerService.info("(sample info)", "(app id)")*

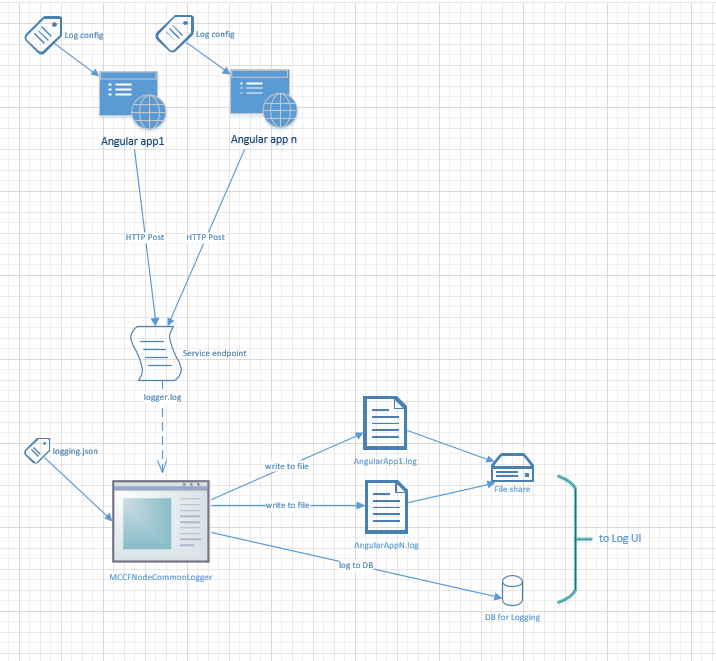
*this.loggerService.debug("(sample debug)", "(app id)")*

Each Angular application will provide its identifier (i.e. application name) in the logging message.

Once a service-endpoint is setup to listen for logs messages, an HttpOutputProvider would be created to send log messages to the service-endpoint via HTTP POST /api/log. The details of the service are black-boxed from the perspective of the client-side application.

This express-driven node service will be using the MCCFNodeCommonLogger to do the server-side logging for Angular applications. It will have **one** logging.json file which will be used to configure the logging for each Angular application.

Figure 16 - MCCF TAS Logging Configuration

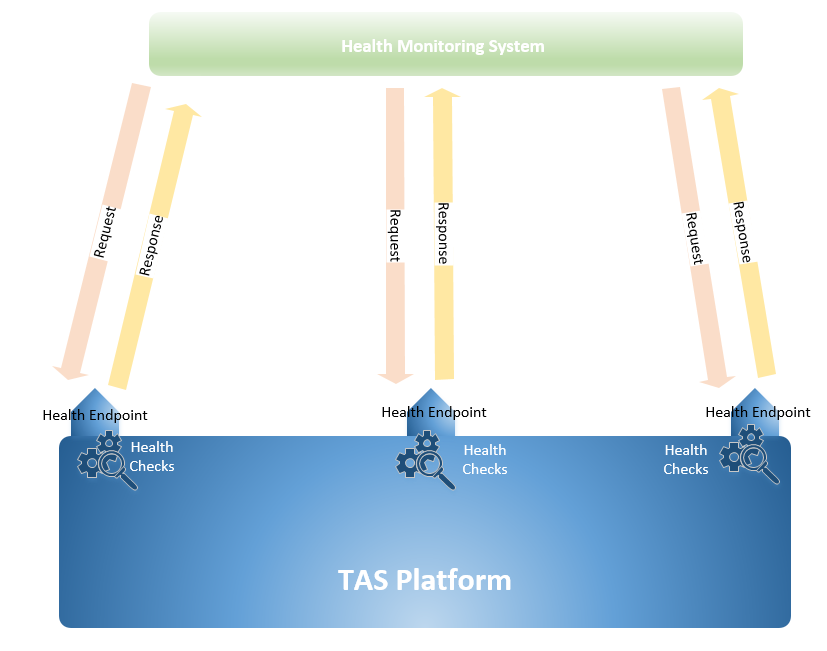


**Health Monitoring System interaction with the TAS platform**

The health monitoring system will check health endpoints which will be provided by the TAS platform.

The health monitoring system sends a request to the health endpoint. The health endpoint performs necessary checks to determine if the application is healthy. Once the health endpoint’s health check is finished it sends a response back to the health monitoring system.

Figure 17 - MCCF TAS Health Monitoring System High-level Design



The response will either be successful (e.g. HTTP 200) or not successful (e.g. != HTTP 200). If the endpoint doesn’t respond, the health monitoring system’s request will time out (see configuration section).

**Health Monitoring System configuration (adding and editing a health endpoint)**

To be able to access a health endpoint, the health monitoring system must configure it. The following items must be configured:

* Name: Name of the endpoint in the health monitoring system configuration
* Address/Path to the health endpoint
* Protocol used: HTTP, HTTPS, etc.
* Request interval (in seconds)
* Timeout: Interval in seconds (how long to wait for response from endpoint)
* Failure threshold
* Email address: Email address to notify in case of unhealthy endpoint

**Health check flow**

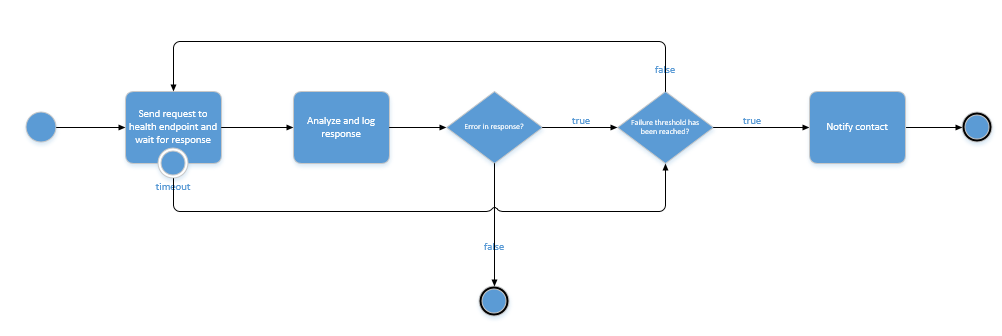
The health monitoring system will send requests to the health endpoint in the interval defined in the configuration.

If the health endpoint responds with a success message (e.g. HTTP 200), the health endpoint is considered healthy and no further action is required.

If the health endpoint doesn’t respond in the timeout interval defined in the configuration or if the endpoint returns an error (e.g. != HTTP 200), the health monitor will count the number of consecutive requests that the health endpoint hasn’t responded to or for which an error has been returned. If the health endpoint responds successfully, the counter for the failure threshold will be reset to 0.

Once the failure threshold has been reached, the health endpoint is considered unhealthy and an email will be sent to the email address in the configuration.

Figure 18 - MCCF TAS Health Check Flow



**Database for retrospective reporting**

Every request sent to a health endpoint and every response received by the health monitoring system will be saved in the non-VistA TAS data storage (it must yet be defined which database and/or collection) for retrospective reporting. Until the decision on the database has been made, the data will be stored in a log file.

A request/response object stored in the database (in the interim a log file) will have the following attributes:

* Health endpoint name
* Request
* Response: If timeout, then blank
* Successful: true/false (true if timeout)
* Failure threshold count
* Timestamp

The retention period for a request/response object varies based on the type of data in each object. An approach for deleting objects in the database that have reached their retention period must be defined to meet these business needs.

##### User Interfaces

This section identifies the standard layout and design to be used on each webpage developed for the TAS portal which will be based upon the U.S. Web Design Standards (USWDS) and addresses the above user stories.

This reference document defines USWDS typography, grid, UI components, headers etc.

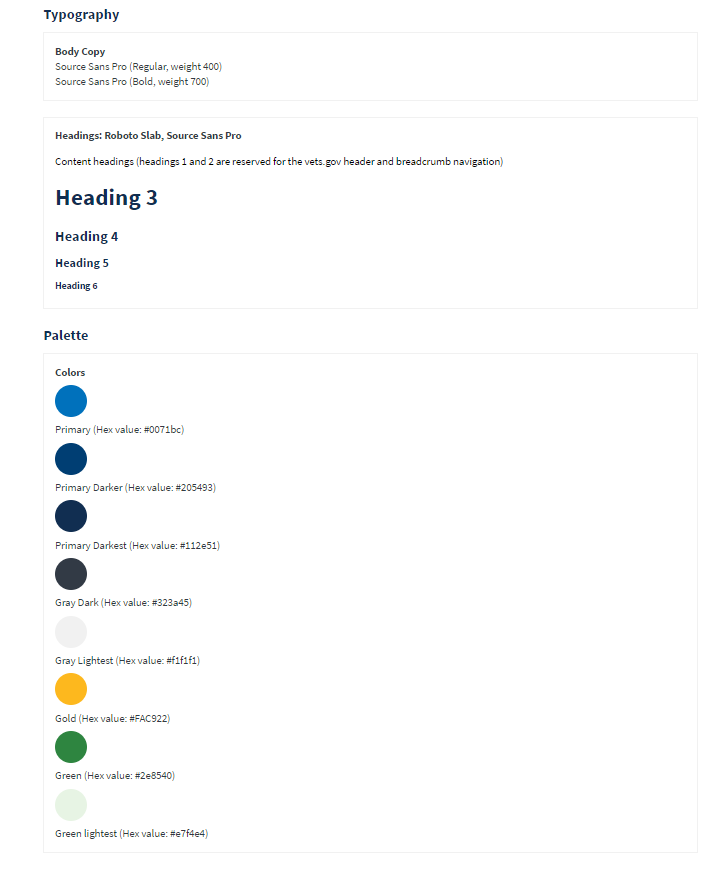


This reference document defines more clearly USWDS typography lists, form controls, buttons, side navigation etc.



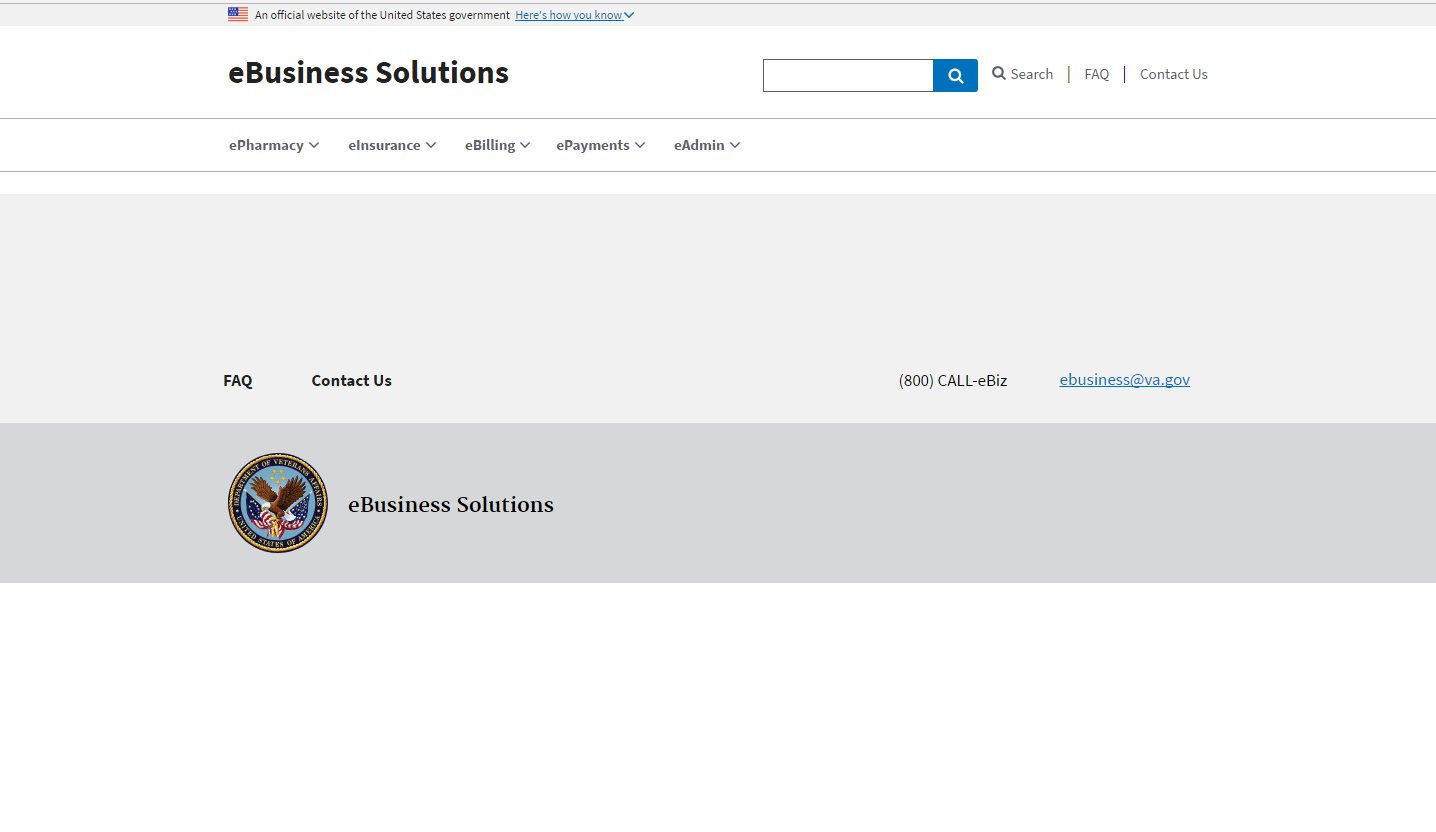
An example of design using USDWS could include:

Figure 19 - USWDS Design Elements



Example prototype of the TAS portal main page has been preliminarily designed using USWDS and will be used as a design template for each of the platform pages including the MCCF EDI home page and the product landing pages - eBilling, eInsurance, ePayments, ePharmacy, eAdmin.

Figure 20 - MCCF TAS Portal Screen Mockup



Other high-level requirements such as calendar, news and announcements, alerts / update section, current date / time, breadcrumbs, help and contact list link can be added to the main content section of the page using USWDS components and formatting.

The prototype has FAQ and Contact Us in both the Header and Footer as placeholders.

Figure 21 - USWDS Landing Page Template

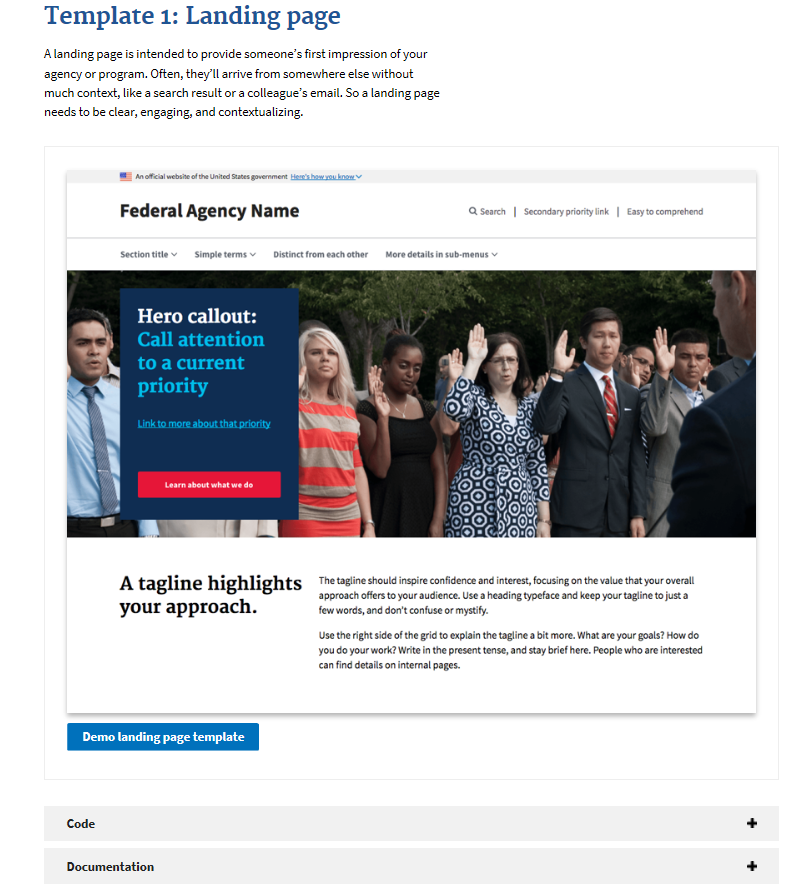
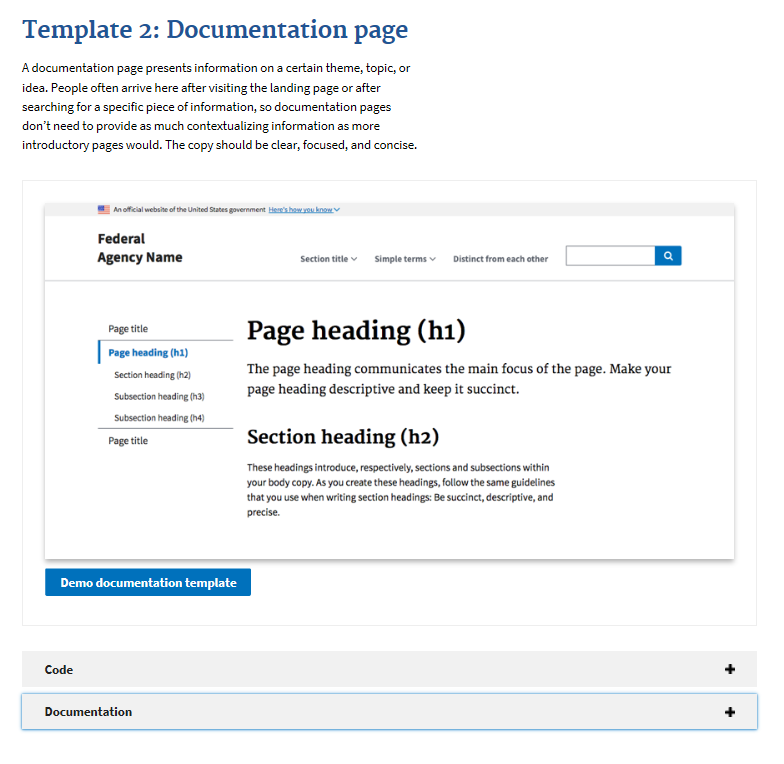


Figure 22 - USWDS Documentation Page Template



**Wireframes**

|  |  |
| --- | --- |
| **C:\TAS Wireframes\5_11_20118\FINAL DESIGN NEW FOOTER.png** | **C:\TAS Wireframes\5_11_20118\eInsurance Landing Page copy.png** |
| **C:\TAS Wireframes\5_11_20118\ePayments Landing Page.png** | **C:\TAS Wireframes\5_11_20118\ePayments Report.png** |
| **C:\TAS Wireframes\5_11_20118\ePharmacy Landing Page.png** | **C:\TAS Wireframes\5_11_20118\ePharmacy Modal.png** |
| **C:\TAS Wireframes\5_11_20118\ePharmacy Display Report.png** | **C:\TAS Wireframes\5_11_20118\eRevenue Resource Modal.png** |
| **C:\TAS Wireframes\5_11_20118\Non-MCCF Landing Page.png** | **C:\TAS Wireframes\5_11_20118\Alerts.png** |
| **C:\TAS Wireframes\5_11_20118\NPI Landing Page.png** | **C:\TAS Wireframes\5_11_20118\FAQ.png** |
| **C:\TAS Wireframes\5_11_20118\Edit Reports Config.png** | **C:\TAS Wireframes\5_11_20118\Contact Us.png** |
| **C:\TAS Wireframes\5_11_20118\eAdmin.pngC:\TAS Wireframes\5_11_20118\System Home.png** | **C:\TAS Wireframes\5_11_20118\Help Page.pngC:\TAS Wireframes\5_11_20118\IAM Manager.png** |
|  |  |

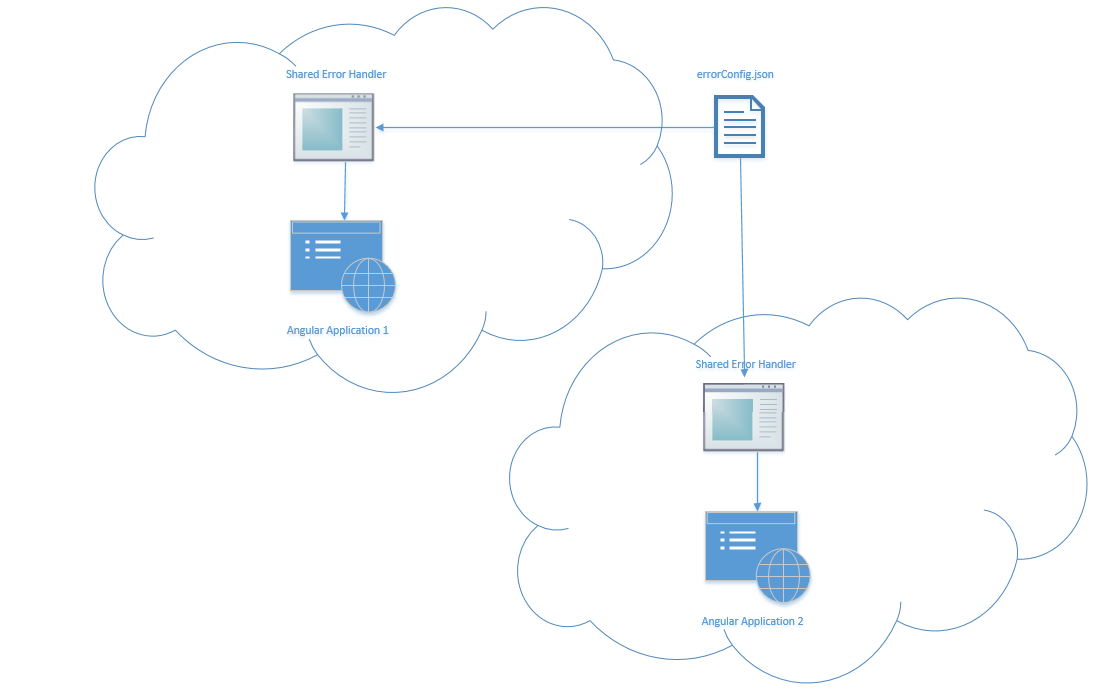
**Webpage Error Handling**

A shared Angular error handling component (shared component) will be developed which will be used by MCCF TAS Angular applications to handle UI errors. (Each application will include its own instance.)

The shared component could be implemented in a backend service later (the tradeoff will be more backend calls) instead of implementing it in the presentation layer (If page load times become an issue, it might need to be prioritized what is loaded in the UI).

The shared component will use a configuration file which determines how it will handle UI errors and which information to display to the user.

Figure 23 - MCCF TAS Error Handling High-level Design



The configuration file will be stored centrally on a file share (location must be defined yet) or in the database. This way it will be accessible by every application that uses the shared component. Editing of the configuration file could be done through a planned content management capability in the future to allow assigning permissions, tracking changes, and approvals.

The configuration file will be in the JSON notation and will have the following structure:

* errors[]
  + errorCode
  + errorMessageUser
  + errorMessageTechnical
* contacts[]
  + application
  + contact
    - name
    - email

The name of the configuration file will be ***errorConfig.json***

Example configuration file:

***{***

***"errors": [***

***{***

***"errorCode": "HTTP 500",***

***"errorMessageUser": "good error message that denotes what is occurring",***

***"errorMessageTechnical": "Internal Server Error"***

***},***

***{***

***"errorCode": "HTTP 501",***

***"errorMessageUser": "good error message that denotes what is occurring",***

***"errorMessageTechnical": "Not Implemented"***

***},***

***{***

***"errorCode": "timeoutVistA",***

***"errorMessageUser": "A timeout occured while accessing VistA. Please try again later.",***

***"errorMessageTechnical": "timeout technical message"***

***},***

***{***

***"errorCode": "default",***

***"errorMessageUser": "An error occurred",***

***"errorMessageTechnical": "An error occurred - default"***

***}***

***],***

***"contacts": [***

***{***

***"application": "eBilling",***

***"contact": {***

***"name": "First Name LastName",***

***"email": "first.last@va.gov"***

***}***

***},***

***{***

***"application": "eInsurance",***

***"contact": {***

***"name": "First Name LastName",***

***"email": "first.last@va.gov"***

***}***

***}***

***]***

***}***

The error messages will be the same for all applications. The contacts will be specific to each individual application (The shared component will need to determine which application ran into the error and map the correct contact information).

If an application wants to use the shared component, a system admin (yet to be defined) must be notified, so he/she can add the contact information to the errorConfig.json file.

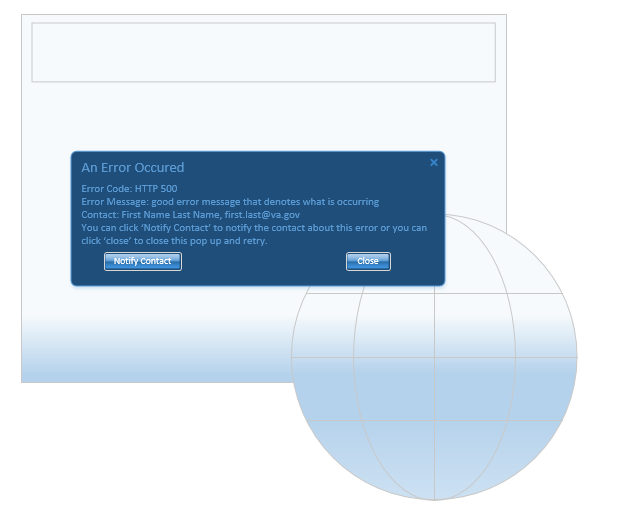
The configuration file must also define a default error element in case the error that occurred cannot be found in the configuration.

The shared component will be loaded the first time the application is called. The shared component will then read the configuration file. If there is a change to the configuration file, the shared component must detect the change and load the new configuration file.

If a UI error occurs the application must catch that error and pass it to its instance of the shared component. The shared component will create a pop-up window that will inform the user that an error occurred. The main Angular page will remain untouched and will not display any error related information (i.e. stack trace).

Based on the configuration file, the pop up will contain information regarding the error that has occurred.

Figure 24 - MCCF TAS Custom Error Message



When clicking the ‘Notify Contact’ button on the pop up, the shared component will send an email to the contact (errorConfig.json - errorConfig.contact.email). Email messaging will use the SMTP server implemented in the TAS Platform. When clicking the ‘Close’ button, the pop up will close.

The email that will be sent to the contact (errorConfig.json - contacts.contact.email) will contain the following information:

Email subject:

An error occurred in application: *see table below*

Email body:

Error Code: *see table below*

ErrorDescription: *see table below*

Stack Trace: *see table below*

User: *see table below*

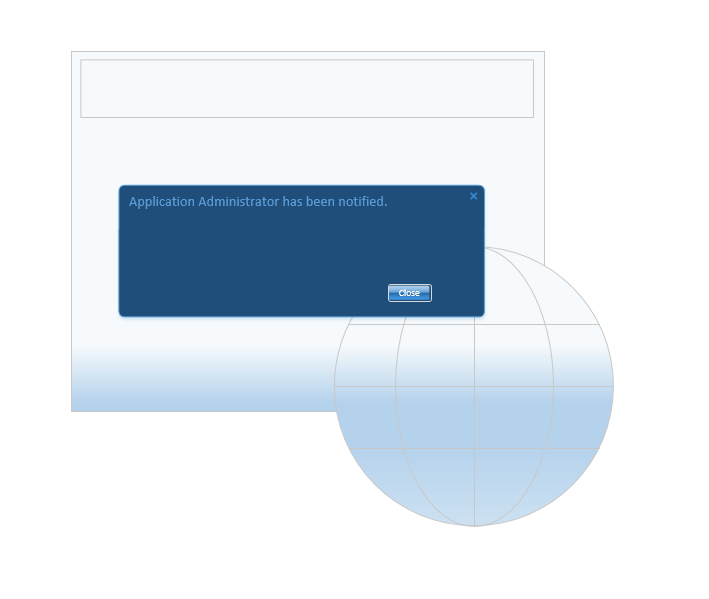
Date/Time: *see table below*

| Email content | Object |
| --- | --- |
| An error occurred in application | errorConfig.json – contacts.application |
| Error Code | errorConfig.json – errors.errorCode |
| ErrorDescription | errorConfig.json – errors.errorMesssageTechnical |
| Stack Trace | Stack Trace provided by the system |
| User | Reporting User |
| Date/Time | Timestamp when the error occurred |

If the email sending was successful, the text in the pop up will change to:

**Application Administrator has been notified.**

Figure 25 - MCCF TAS Email Notification Acknowledgement



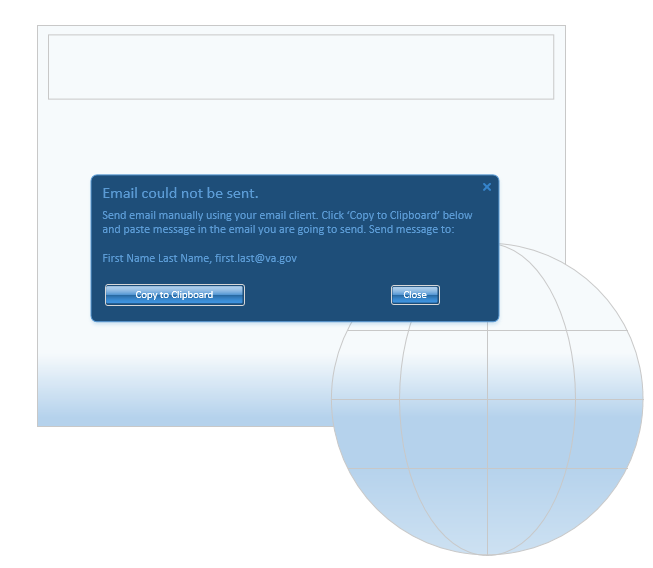
If an error prevents the email from being sent successfully, the pop up text will change to:

**Email could not be sent.**

**Send email manually using your email client. Click ‘Copy to Clipboard’ below and paste message in the email you are going to send. Send message to:**

**First Name Last Name,** [first.last@va.gov](mailto:first.last@va.gov)

Figure 26 - MCCF TAS Error Email Notification



The pop up will contain a button ‘Copy to Clipboard’.

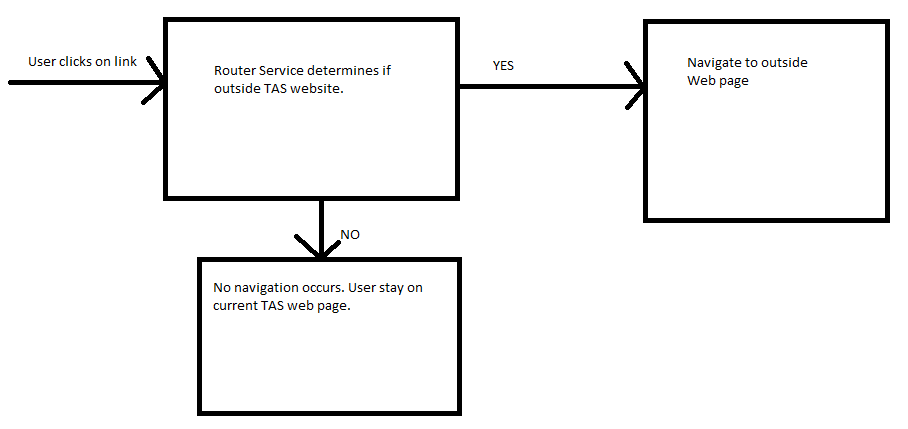
Clicking the button will copy the relevant information to the clipboard (same information that would be provided via email – see above).

**MCCF TAS Off-site notification**

TAS users need to be notified when they click a link that takes them outside the TAS system. TAS users can click on links to navigate to websites outside of the TAS system. The standard styling and formatting for notifications and other UI elements is defined in the MCCF EDI TAS Style Guide.

The Angular 2 Node.js framework provides a Routing service for handling user navigation using URL addresses. The Router service should catch clicks on URL addresses that are links leading to webpages outside the TAS system. The Router service should present a visual message notifying users that they are leaving the TAS website. See Angular 2 Router service. <https://angular.io/docs/ts/latest/guide/router.html>

Figure 27 - MCCF TAS Off-site Notification Flow



Popup notice window.

**NOTICE**

You are about to leave the TAS website

Cancel

Continue

**MCCF TAS Log UI**

The Log UI is a user interface that will be used be the product owner to view and search for log files. The user will have to log in to the Log UI and authenticate themselves (Utilization of IAM will have to be researched during development).

The user will be able to search for messages using the following search criteria:

* Date from (date/time picker)
* Date to (date/time picker)
* Log level (drop down menu, multiple selections possible)
* Text search for message (input text)

There is a search button that will retrieve log messages from the log files on the files share/log database that match the search criteria. Each log message is in the JSON format.

There will be a reset button that clears the search criteria. When no search criteria are selected, all log messages will be returned.

There will be a log out button that logs out the current user.

The log messages will be displayed in a table that is filterable and sortable.

The table that displays the log messages that are returned will have 3 columns:

* Timestamp
* Log level
* Message

The following mapping will be applied:

Table – Event Logfile to Table Row Mapping

| Element in message from logfile | Element in table row |
| --- | --- |
| level | Log level |
| timestamp | Timestamp |
| message | Message |

Exceptions in the exception log file will also be displayed when ‘error’ debug level is selected in the search criteria.

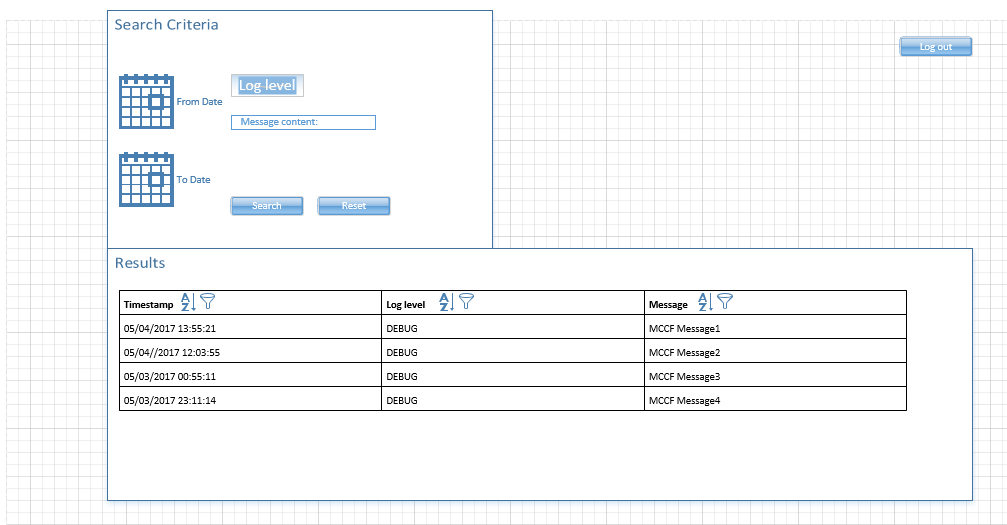
The following mapping will be applied for exception messages:

Table - Exception Logfile to Table Row Mapping

| Elements in message from logfile | Element in table row |
| --- | --- |
| level | Log level |
| timestamp | Timestamp |
| process + os + trace + stack + message | Message |

The following diagram shows the design of the Log UI

Figure 28 - MCCF TAS Log User Interface

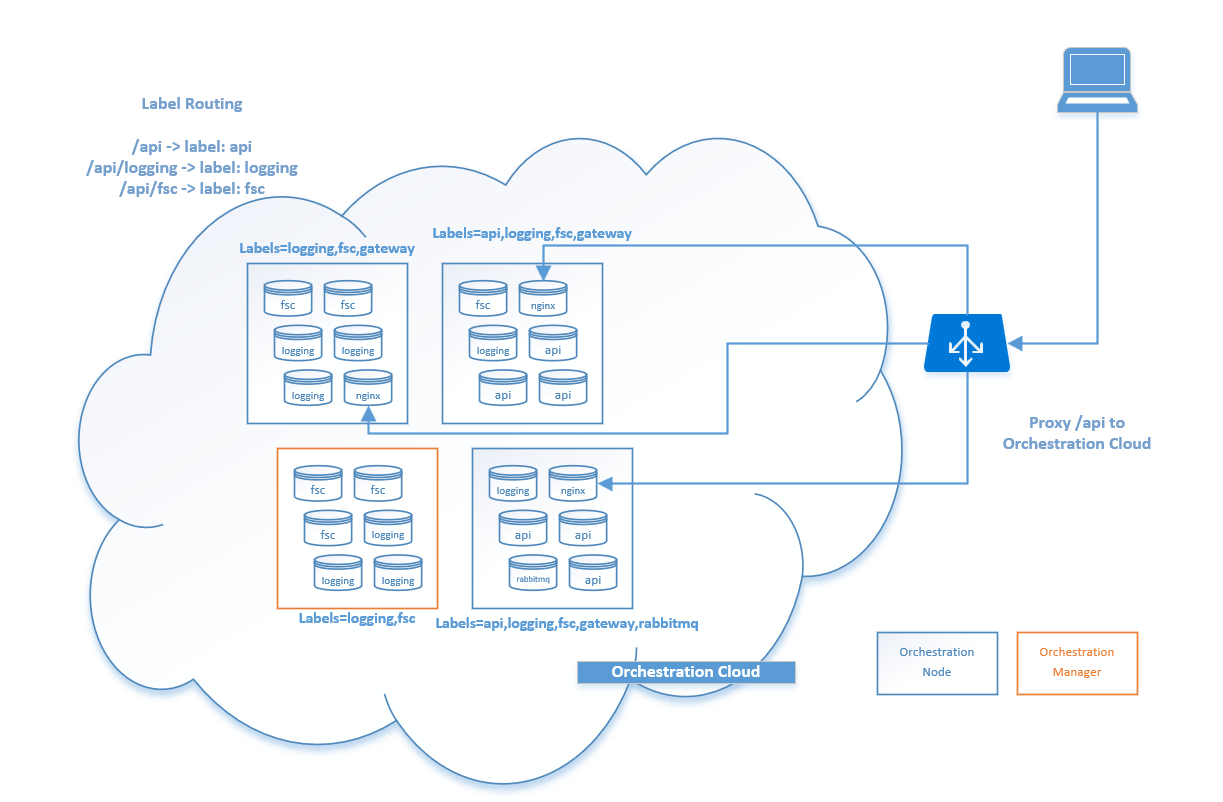


##### Communications Interfaces

MCCF EDI TAS will leverage the underlying communication interfaces for the VDA/VA ESB enterprise service framework. These communication are dependent platform/stack components provisioned by Enterprise Operations. This effort will use standard HTTP/HTTPS over TCP/IP channels available for solutions to communicate over TCP/IP with external applications. The final solution implementation will require coordination with system administrators at data center for configuration setup of endpoints.

##### Orchestration Design

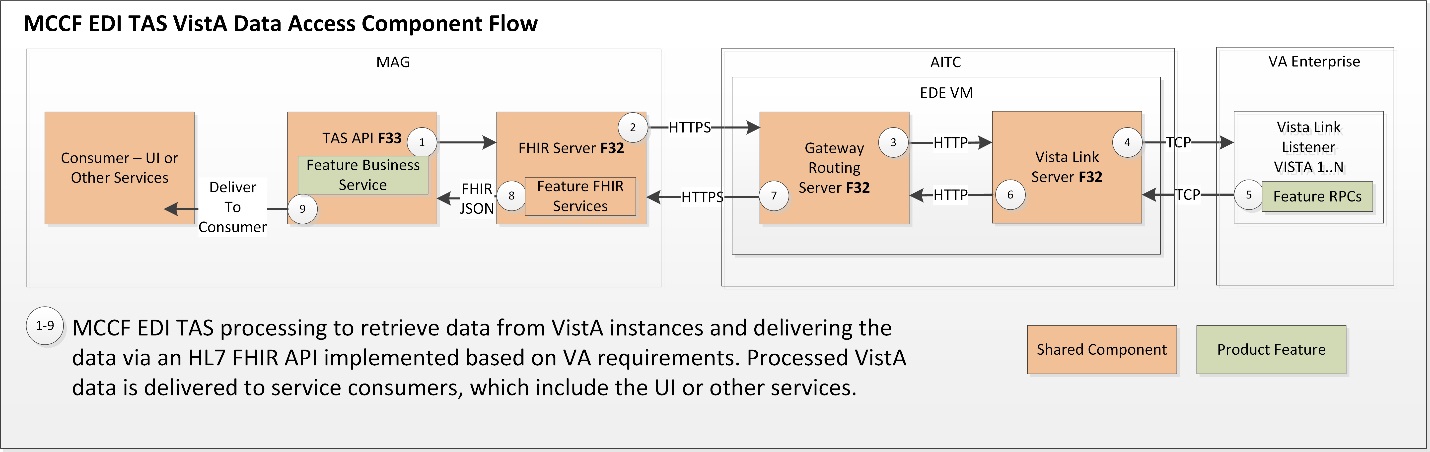
TAS API instances are run with Docker in a private orchestration cloud. Each underlying system is given a set of labels, which the orchestration manager will use to spin-up instances of components. Ingress traffic will go to the gateway, which will route traffic to the proper instance. The orchestration cloud handles failover, transport encryption, and load-balancing automatically.



#### Data Access Services Design

MCCF EDI TAS includes VistA Data Access (VDA) Services that implement access to TAS required data in the VA VistA instances across the enterprise. The diagram and sections below describe the components and design of the TAS VDA services.

Figure 29 - MCCF EDI TAS VistA Data Access Services Design



##### NGINX

NGINX is software that features general purpose web proxy and load balancing capability. It will be implemented on the Gateway Routing Server and will be used for routing, proxying, and load balancing in the VDA design and will provide HTTPS endpoints for CREATE, READ, UPDATE, DELETE, and SEARCH operations for FHIR resources. It will pass through the requests to the FHIR Server.

##### HAPI FHIR Server

The HAPI FHIR Server will provide a REST interface with the same operations. Once the HAPI FHIR Server receives a request from the ESB, it extracts the data from the request that is needed for the request to VistA.

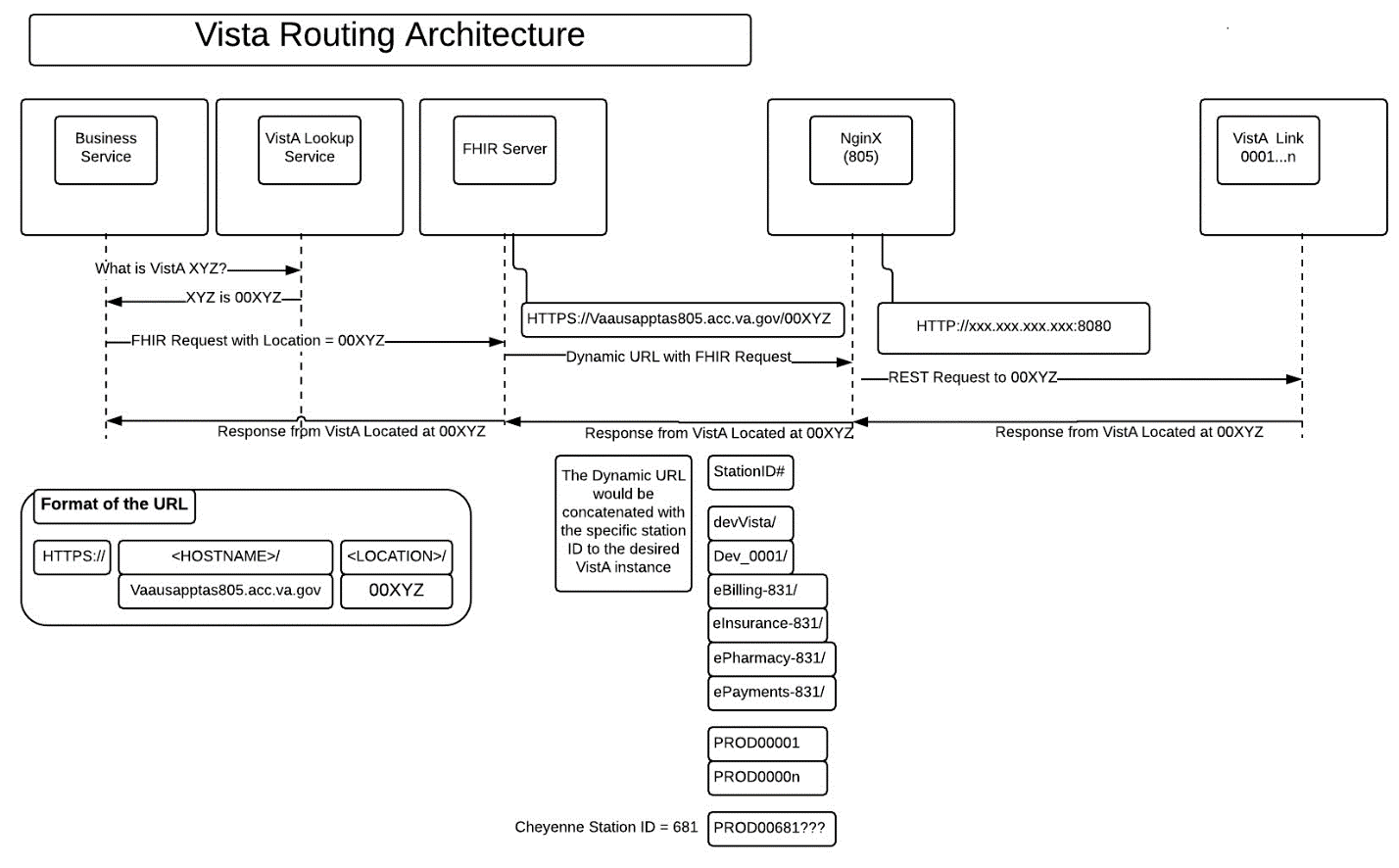
It then determines the files and the records within the files in VistA that are going to be involved in the request. Depending on the request the HAPI FHIR Server creates, reads, updates, deletes, or searches these files and records.

The last two steps in the HAPI FHIR Server are mapping the data from VistA to the FHIR resource(s) and returning them back to the ESB (HAPI FHIR Proxy Message Flow).

##### VistA Access and Routing Message Flows

This flow provides a REST interface that includes CREATE, READ, UPDATE, DELETE, and SEARCH operations, as well. After receiving a request from the HAPI FHIR Server, the message flow determines which VistA instance it needs to route the request to. The criteria and logic for routing will be based on the VistA location. It also provides VistA monitoring and logging of VistA requests/responses by utilizing the logging service.

Figure 30 – VistA Data Access Routing Architecture



##### VistA Instances

Each VistA instance will have a TCP Listener/REST API that will be called by the VistA Access and Routing Message Flow in the Gateway Routing Server. It calls Fileman and Fileman accesses the required file.

More information regarding the TCP Listener/REST API can be found in the following documents:



#### Data Storage Design

##### Azure Storage Mechanics

Azure Storage ("AS") is general storage that can be accessed in four ways: SMB mounts, blobs, tables, and queues. Virtual machines use AS for their disks. Even when VMs use managed disks, VM still uses AS for metadata and systems logging.

AS is redundant and highly available with encryption-at-rest built-it. AS data always replicated to another physical storage rack (LRS) but can also be replicated to a remote site for high availability by using ZRS, GRS, or redundancy RA-GRS modes.

Accessing AS via blobs enables large-scale storage of files. Access via tables enables large-scale storage of flat record data. In either case, these entities (files/records) can be accessed by an API or, if access is granted, by users directly. Access can be open (browsable), limits to direct entity access (with a known URL, e.g. for CSS/JS asset loading), or private. When access is private, access can be granted by key or by shared access security ("SAS") token. The latter is the recommended model. SAS tokens limit access to entities by entity or set of entities by and restricts access to a time window.

Table - Azure Storage Shared Access Signature



Blob storage is for any type of file. File content types are set via programmatic uploading. An account’s blob storage is split via containers. Containers contain files. Though there is no concept of a “folder” is AS, filenames can include a slash (“/”), thus creating the same effect.

Table storage is for flat data with a flexible table schema. All records have a PartitionKey, RowKey, and Timestamp. Each record can include up to another 252 fields. Because there is no schema these fields do not need to be the same in each record. Therefore, table storage is inherently sparse.

Fields have the common data types: Binary, Boolean, DateTime, Double, Int32, Int64, and String. There is also a Guide data type. Each field has a size limit of 64KB. For data that is defined to be larger, either a FatEntity pattern (splitting data across multiple columns) can be used or table storage can reference blob storage.

The PartitionKey field gives scope to the data. This scope is user defined. For example, if log records are stores, the PartitionKey could indicate a year, year/month, or year/month/day combination.

A record can be accessed directly via a PartitionKey/RowKey combination. Entire partitions of records can be loaded at a time with the PartitionKey. Rows can also be queries using basic filtering mechanisms.

##### Summary

Core data storage such as site configuration, logging/auditing data, and content management data can be stored safely and securely in Azure Storage.

This maximizes the core use cases of each system.

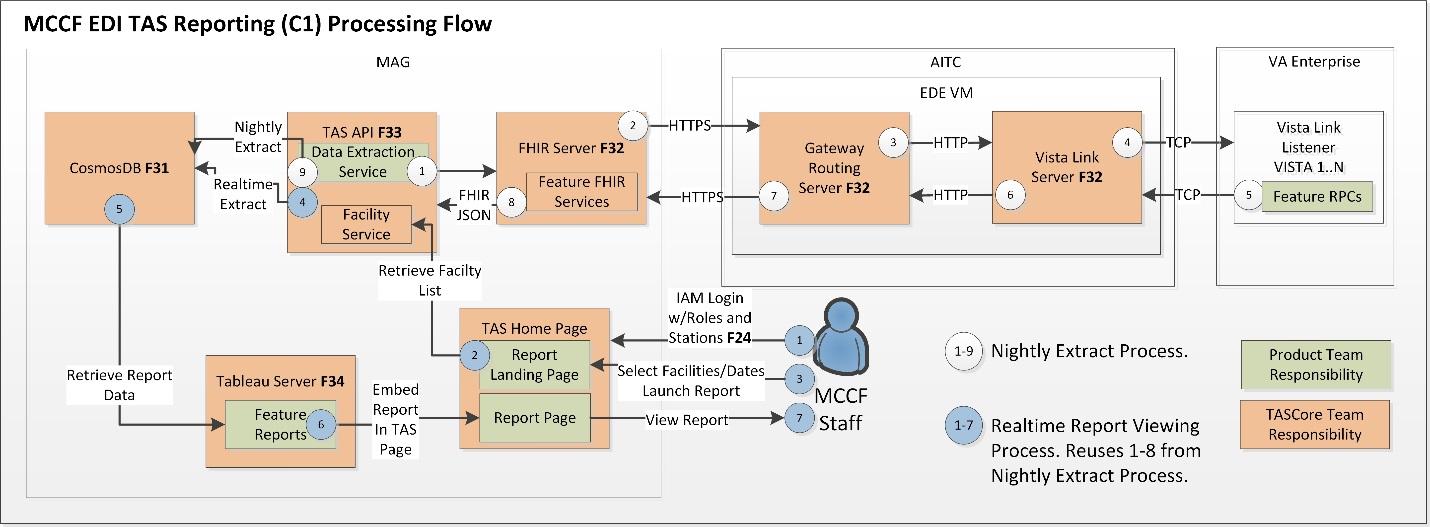
#### TAS Architecture Capabilities

The following general capabilities will be applied across multiple layers in the TAS architecture.

##### TAS Reporting Capability

Reporting will be needed for all product lines. One of the Capabilities in the TAS backlog is for porting reports from VistA to TAS. This section describes the design for reporting within TAS.

Figure 30 - TAS Reporting Design



###### Use Cases

There have been three use cases identified for the reporting capability

1. View Reports in Reporting Frontend
2. Nightly loading of VistA data into MCCF Reporting Database
3. Editing configuration for loading VistA data into MCCF Reporting Database script



For the use cases, a sequence diagram is included in the following section

###### Sequence Diagrams

Figure 31 - Viewing Reports in Reporting Frontend

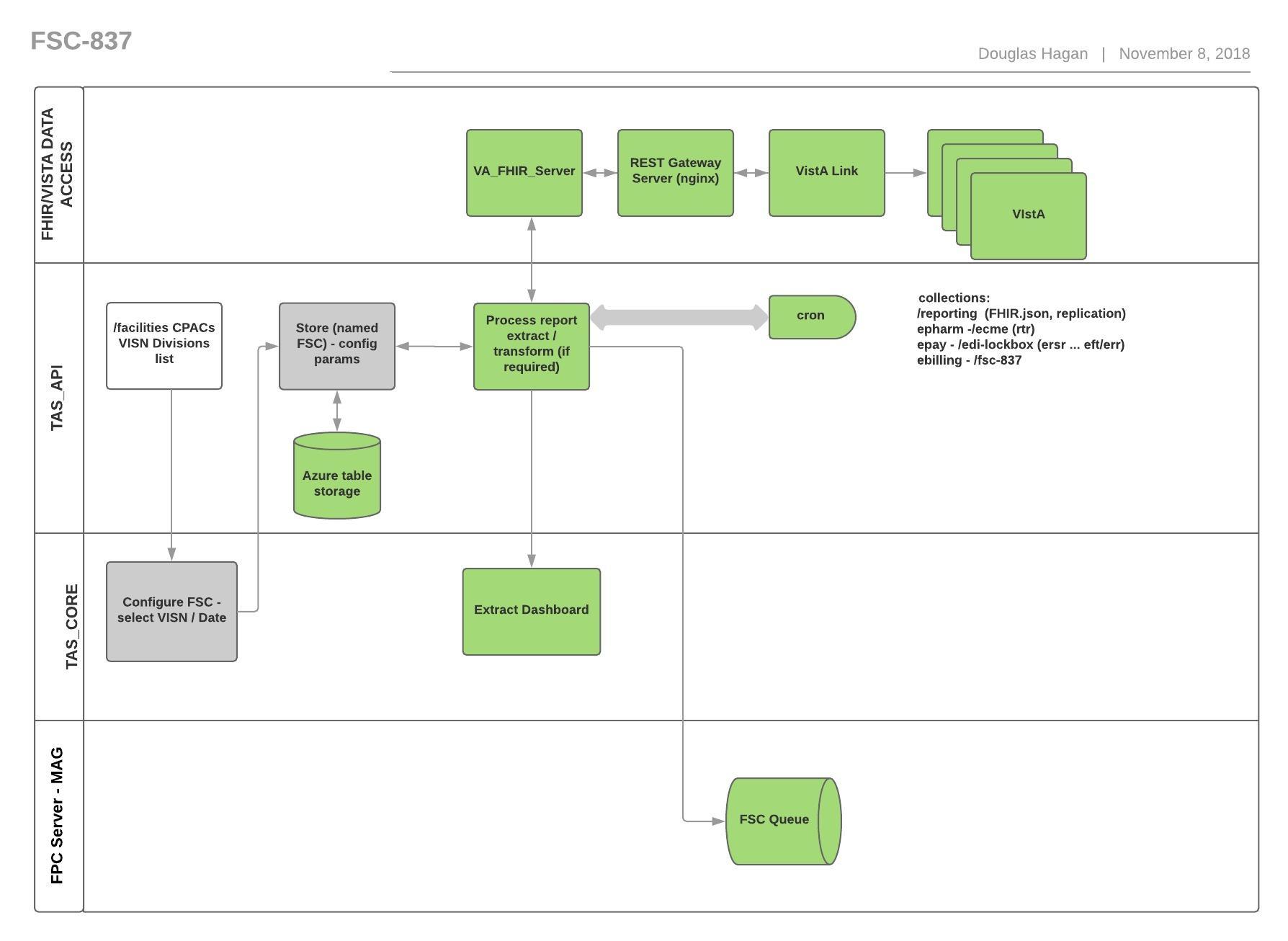
Assuming user has been authenticated and authorized to view report

Assuming the user has been authenticated and authorized to view the report, the Reporting Frontend will provide the user with a link to the report. After selecting the link, the reporting frontend calls a Node.js script in the business layer (URL for report as parameter) that will call a method in the Business Service. The Business Service will call the Tableau Reporting server to retrieve the desired report (using the URL that was sent from the frontend).

###### Nightly loading of VistA data into MCCF Reporting Database

There will be a scheduled job that runs on a nightly basis that will access the HAPI FHIR server via the Business Service to load the data required for the reporting purposes from VistA into the MCCF Reporting database.

Table - MCCF EDI TAS Reporting Nightly Loading Sequence Flow Diagram



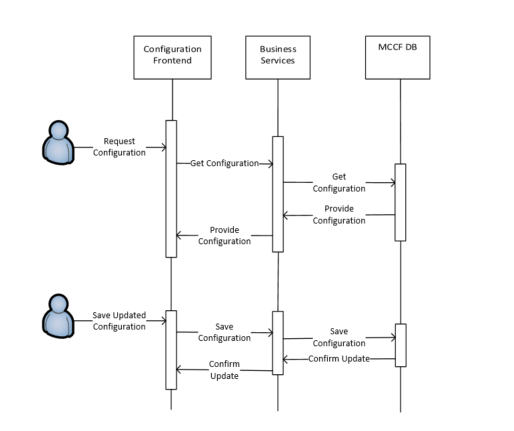
Before the script does its actual job of copying reporting data from VistA to the MCCF Reporting Database, it first checks its configuration (i.e. which data to copy, etc.).

The script then accesses the HAPI FHIR server (using the service in the business layer and NGINX as a proxy). The HAPI FHIR server calls VistA to retrieve the files that are needed to set up the requested FHIR resource(s). VistA returns the required files and HAPI FHIR returns the FHIR resource(s) to the script in the business layer.

The business service then stores the data from the FHIR resource(s) to the MCCF Reporting Database.

Figure 32 - Editing configuration for loading VistA data into MCCF Reporting Database

Assuming user has been authenticated and authorized to edit configuration



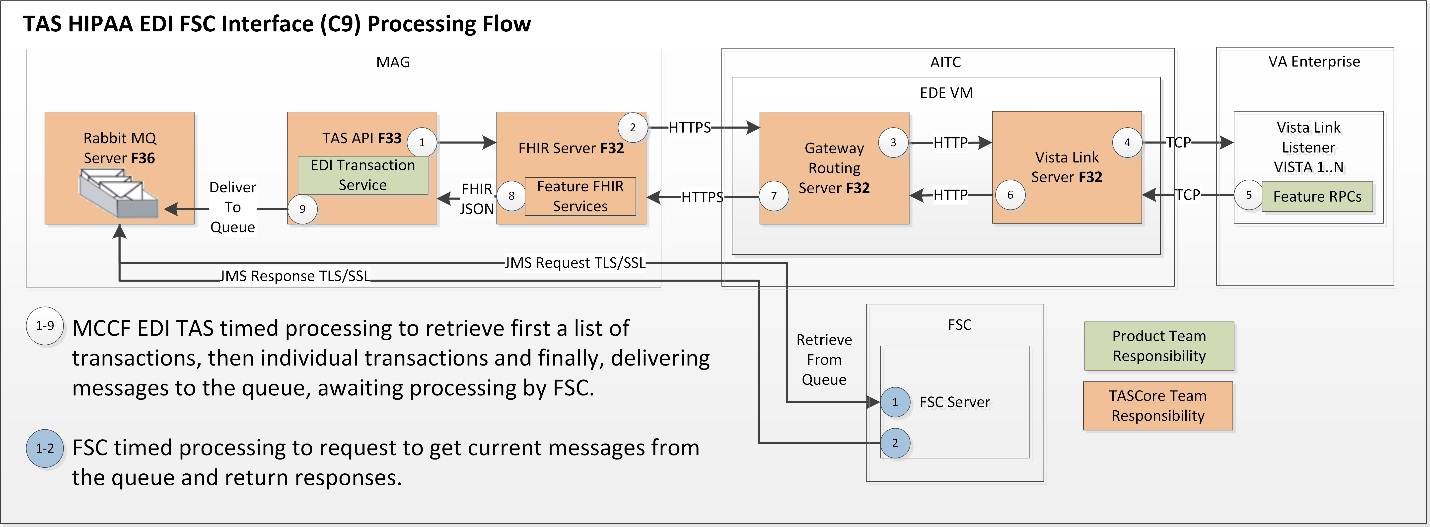
Assuming the user has been authenticated and authorized to edit the configuration, the user loads the configuration file into the frontend. The business layer implements a module that retrieves the configuration from the MCCF Database.

After editing the configuration in the frontend, the user saves the configuration into the MCCF Database via accessing a module in the business layer which then accesses MCCF Database and saves the data edited in the frontend.

##### TAS FSC Interface Capability

The interface to FSC will be needed for all product lines and will be used to transfer HIPAA EDI Transactions. One of the Capabilities in the TAS backlog is for porting EDI Transaction processing from VistA to TAS. This section describes the design for the FSC interface within TAS.

Figure 30 - TAS FSC Interface Design



###### Use Cases

There have been three use cases identified for the FSC interface capability

1. Monitor transaction processing logs
2. Editing configuration for FSC interface service



For the use cases, a sequence diagram is included in the following section

###### Sequence Diagrams

Figure 31 - Processing EDI Transactions through FSC Interface

For each transaction, once the transaction data is ready for processing, the Transaction processing service retrieves the configuration and sends a request to read the transaction data from VistA. Once the data that is needed for the transaction is read from VistA, it is placed on the Queue for processing.

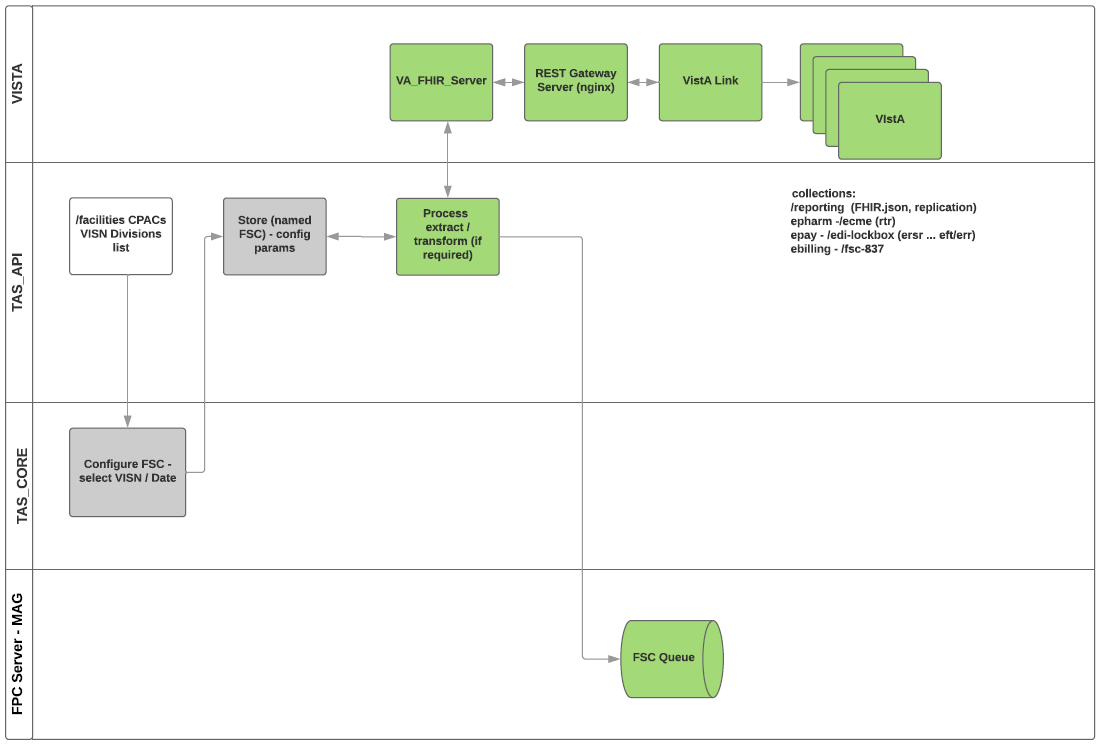
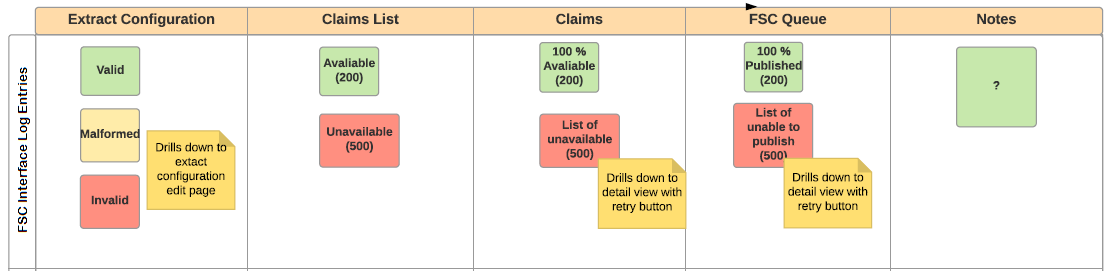
Assuming the user has been authenticated and authorized to view the transaction log entries, the TAS UI will provide the user with a link to the report. After selecting the link, the logging dashboard frontend calls a Node.js script in the business layer that will call a method in the Business Service to retrieve and display the log entries shown in the diagram below.

Figure 31 – FSC Interface EDI Transaction Log Entries



##### TAS Proxying

Since TAS will implement web services that expose HTTPS endpoints, Service Proxying will follow the same design patterns as proxying for web sites. Proxying will be implemented for all the service implementations at the business layer as well as for the TAS FHIR API endpoint. The content below describes how service proxying will occur at each layer.

###### Business Service Proxying

Table 9 - MCCF EDI TAS Business Service Proxying



Business services in TAS will be the primary means of executing business rules, processes and logic as well as being the primary way that data is accessed from the data access layer and then delivered to the presentation layer and the UI. Proxying will be implemented above and below the Business layer. Proxying will occur through proxy servers and/or load balancers. The TAS architecture includes NGINX, which is a proxy server and load balancer. It will be deployed to implement proxying and load balancing for requests coming into the business layer and to the TAS FHIR API servers.

###### Client to Presentation Layer Proxying

Proxying of a client browser request will occur from the VA locations where the TAS web application is being used to the TAS application hosted on the MCCF MAG servers. This HTTPS request will be made within the internal VA network and after being routed using the VA network infrastructure, it will be received by the NGINX server. The NGINX server will provide URL proxying for the TAS web application, mapping the published URL on the VA network to the URLs for specific servers in the MCCF MAG environments hosting the TAS application. The NGINX server will also load balance between servers and will cache requests if needed.

###### Presentation Layer to Business Layer Proxying

Proxying of a service request for a business service from the presentation layer will occur from the VA locations where the TAS web application is being used to the business service HTTPS endpoints hosted on the MCCF MAG servers. This HTTPS request will be made within the internal VA network and after being routed using the VA network infrastructure, it will be received by the NGINX server. The NGINX server will provide URL proxying for the service endpoints, mapping the published URL on the VA network to the URLs for specific servers in the MCCF MAG environments. The NGINX server will also load balance between servers and will cache requests if needed.

###### Business Layer to FHIR API Proxying

The service requests made from the business layer services to the TAS FHIR API endpoint will occur entirely within the TAS system boundary. These requests may get routed to any one of the servers running the FHIR services within the MCCF MAG environment. The decision of which server to route the request to will be made by the load balancer implemented within the TAS infrastructure. The FHIR API endpoint will also be proxied to allow a single endpoint to represent all the servers hosting the TAS FHIR services. Proxying and load balancing will be implemented using NGINX initially. Microsoft Azure proxying, and load balancing services may also be used in the future.

##### TAS CORE Caching Strategy

###### Introduction

Caching is the reuse of previously executed computations and/or enablement of faster hardware alternatives. In the MCCF TAS CORE application, caching will be used improve application performance and application reliability. The MCCF TAS CORE application will implement industry standard best practices at all layers of the To Be architecture to achieve the best performance and user experience possible. This section describes the caching in the following components: web browser, web application, business logic REST API implementation, data store, Corporate Data Warehouse (CDW), HAPI FHIR and the interface to VistA – InterSystems Cache product. These components include caching at all layers – Presentation, Business, Messaging – Enterprise Service Bus – ESB, Services, Data Access and Data Storage.

Caching will not address delivery of content when the origin server is down, this will need to be addressed by failover or other high availability architectural design strategy. Caching will not support the web application in an offline mode. Hardware load balancing will be addressed in a different section.

The caching strategy shall adopt the Just in Time (JIT) or lean approach. A common pitfall is to over-cache early and thereby misappropriate both memory resources and engineering implementation resources where they are not required.

###### Use cases at a glance

Use case 0) TAS Web application Data access - Data is in VistA – data is accessed via a FHIR API for use in business services or to display in Angular UI.

Use case 1) Logging – Internet of Things (IOT) pattern – Primarily write only.

Use case 2) Content Management Service (CMS) – Essentially Create, Read, Update and Delete (CRUD).

Use case 3) Web application state - User settings and menu management – Essentially Create, Read, Update and Delete (CRUD).

Use Case 4) Data warehousing – Data is in VistA – reports are done in TIBCO Spotfire. – Essentially Create, Read, Update and Delete (CRUD).

###### Caching Basics

Read cache

In read only cache the client component uses higher performance in memory cache in lieu of using the backing data store. It turns out that higher performance in memory is a limited resource and that caching strategies implemented use these resources.

Write cache

Where possible the component layers will use non-blocking writes. One option is to write to cache and write to persistent store when the persistent store is available or less loaded. More research is required to understand the load cases and then address this issue. For now, the recommendation is not to support write cache.

Cache warming

Cache warming is a technique for preloading the cache to improve performance after a server had been restarted.

###### Caching at Each Layer

###### Presentation

Angular 4.0.x

Angular offers ahead of time (aot) compilation of html templates, a form of caching. As well the TAS CORE web application utilizes lazy loading of modules to improve performance and user experience.

Caching with RxJs Observables

Utilize Angular caching strategy via react architectural pattern and Observables. In this technique the Observable implementation provides for .publishReplay(1).refCount() pair of operators which re-use previously retrieved $http.get() results from the react store. No additional setter or getter is needed.

Browser Local Storage

(Need to check browser support)

Store session, user and other commonly used application data to local storage where possible. For example – use localStorage.getItem('token') to store and retrieve application json web tokens (jwt)

Cache Control Headers

Angular will not set cache control headers for REST API calls.

###### Business

express.js

Best practices for express performance

Per the express.js documentation, the best option to cache the results of requests is using a caching server like nginx.

<https://expressjs.com/en/advanced/best-practice-performance.html>

Memcached

In memory cache utilizing memory-cache or memcached is another option.

Approved: <https://www.va.gov/TRM/ToolPage.aspx?tid=9511>

Potential technique: memcached storage express module is a potentially useful option but would require additional One-VA Technical Reference Module (TRM) approval.

###### Messaging – Enterprise Service Bus (ESB)

MuleSoft

Mulesoft Enterprise, Cloudhub supports the “Cache Scope”. This is caching built into specific message flows. Business analysts, strakeholders and subject matter experts would need to be consulted prior to implementing a per workflow based caching strategy. A performance caching optimization at the ESB layer is not recommended at this time.

<https://docs.mulesoft.com/mule-user-guide/v/3.7/cache-scope>

###### Services

HAPI FIHR

HAPI FHIR 2.5 supports default 1 minute caching of search results. It will need to be determined by stakeholders and business analysts if this caching is desired or a hindrance to the source of truth.

<http://hapifhir.io/>

It is the recommendation at this time that caching not be implemented at this layer.

Data Access

Inter Systems Cache/Mumps and REST API

Since VistA is the source of truth, VistA policy, stakeholders and business analysts will need to determine the need for caching. Performance concerns at the Cache level will be addressed by load balancing and other scaling techniques.

It is the recommendation at this time that caching not be implemented at this layer.

###### Data Storage

Azure storage – block blobs

<https://docs.microsoft.com/en-us/rest/api/storageservices/understanding-block-blobs--append-blobs--and-page-blobs#about-block-blobs>

Elasticsearch

There are three types of Elasticsearch caches

1. Node query cache - queries being used in a filter context <https://www.elastic.co/guide/en/elasticsearch/guide/current/filter-caching.html>
2. Shard request caches – caches query results independently for each shard

<https://www.elastic.co/guide/en/elasticsearch/reference/current/shard-request-cache.html>

1. Field data cache – aggregation field values are loaded in to memory <https://www.elastic.co/guide/en/elasticsearch/reference/current/caching-heavy-aggregations.html>

More research is required to understand what the performance gains and other tradeoffs are for this Elasticsearch caching capabilities.

###### MCCF Database

REDIS

Redis is a key value store which is often mentioned in caching discussions. Microsoft Azure support Redis cache https://azure.microsoft.com/en-us/services/cache/

<https://www.va.gov/TRM/ToolPage.aspx?tid=7113>

###### Additional Layers

CDN

Content Delivery Network (CDN) for fast access across geographies to large and commonly accessed static media such as images, video and in some cases static html.

Web cache

A web cache is a software service layer between the bowser client and the server. Web cache caches content. Any content saved in the cache can then be served without regenerating the pages. Nginx is commonly used to reverse proxy an application state. Nginx has full set of caching features. Nginx utilizes Cache-Control headers from the origin server so. Since the web application uses the single page application (SPA) design pattern opportunities for utilizing the web cache will be relegated to static html. The TAS\_CORE web application is highly dynamic and Cache-Control headers are unlikely to be set – see details in the Presentation layer above.

nginx caching

nginx is part of the MCCF TAS CORE load balancing strategy and is TRM approved

https://www.va.gov/TRM/ToolPage.aspx?tid=6605

<https://www.nginx.com/blog/nginx-caching-guide/>

###### Summary

The MCCF TAS CORE caching strategy can be summarized as follows –

* Cache media and static web content
* Use memory where possible
* Utilize data storage component’s built in query cache capability

The table below summarizes the caching strategy at the per layer level.

Table - MCCF EDI TAS Caching by Layer

| Layer | Technology | Capable | Recommended | Discussion |
| --- | --- | --- | --- | --- |
| Presentation | Web browser; javascipt: Angular | Yes | Yes | Use localstorage and best practice javascript techniques for non-blocking and non-repeating service calls within a single page application (SPA) |
| Business | express.js | Yes | Yes | Use memcached if possible. |
| Messaging (ESB) | MuleSoft | Yes | No | Not recommended at this time. |
| Services | HAPI FHIR | Yes | No | Not recommended at this time. |
| Data Access | InterSystems Cache | Yes | No | Not recommended at this time. |
| Data Storage | Azure Storage, Elastic Search | Yes | Yes | Use the component equivalent of query cache where possible. |
| Additional | Content Delivery Network (CDN):Azure block blobs | Yes | Yes | Use a CDN for media and other static content. |
|  | Web Cache: nginx | Yes | Yes | Use Cache-Control headers for static content. |

#### Dependencies and Constraints

IAM

The authoritative authentication service for the VA is IAM. Several IAM services are available for use in the authentication process, including a Single Sign-On service. IAM also provides the Master Veterans Index (MVI) service for obtaining authoritative identifiers for Veterans. MCCF EDI TAS must use IAM services for authentication and identification when the request for access is made and when the request is received. This requires integration with IAM at the presentation layer for making access requests, and will be implemented using the IAM Single Sign-On Internal (SSOi) and Security Token Service (STS) services

**VA Enterprise Architecture**

Adherence to the VA approved architecture and design patterns is enforced through the Compliance Epics defined in the MCCF EDI TAS Product Backlog, Compliance Epics. These Compliance Epics have been mapped to specific MCCF EDI TAS Architecture Epics and Architecture layers to ensure that the MCCF Architecture adheres to VA EA. MCCF EDI TAS needs input from ASD regarding the architecture and compliance to the VA EA.

**VA Standard Data Models**

The MCCF EDI TAS needs to find out what data extensions and profiles will be required for VA implementations, so these can be incorporated into the design of the MCCF services. All currently planned data models are included in section 5.3 Data View. The MCCF team will continue to monitor any new adopted profiles and extensions to ensure that the MCCF services adhere to VA standards.

### Specific Requirements

#### Database Repository

MCCF EDI TAS will follow the FHIR specification as the logical data model for the TAS. The MCCF EDI TAS team is identifying needed extensions and profiles that will be used to modify or constrain the base FHIR resources used by the TAS application. MCCF EDI TAS will conform to any FHIR extensions or profiles.

#### System Features

Major TAS system features that are identified in product team functional user stories will require shared services and components as well as specific functionality. The mapping of shared services and features to specific system features is detailed in a Capability to Build Matrix. This matrix defines which technical components and services are needed for each feature and when they will be needed. The Capability to Build Matrix is available at http://vaww.oed.portal.va.gov/pm/hape/ipt\_5010/EDI\_Portfolio/TASCore/TASCore%20Capability%20to%20Build%20Matrix%2020171006.xlsx

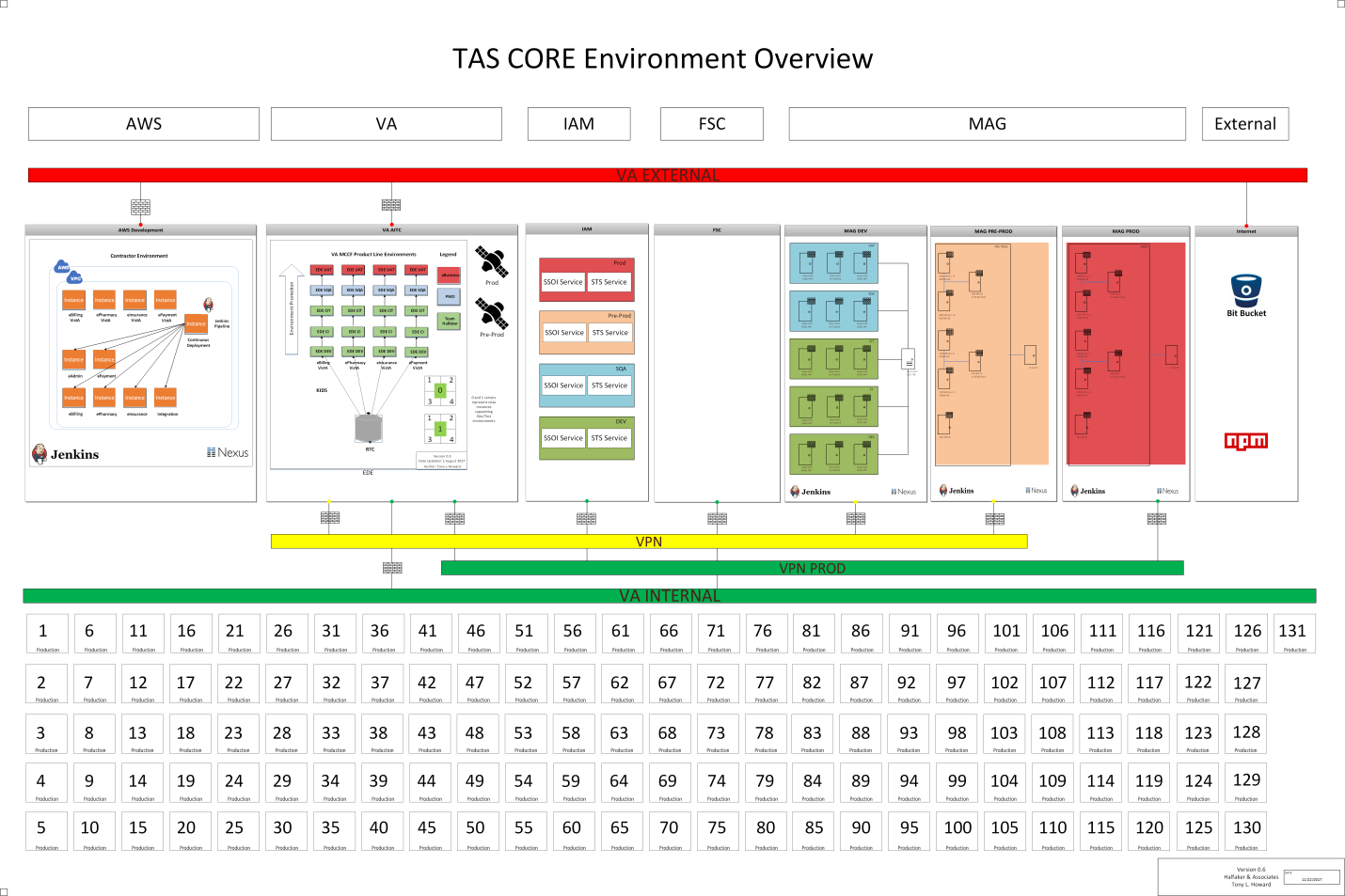
## Network Detailed Design

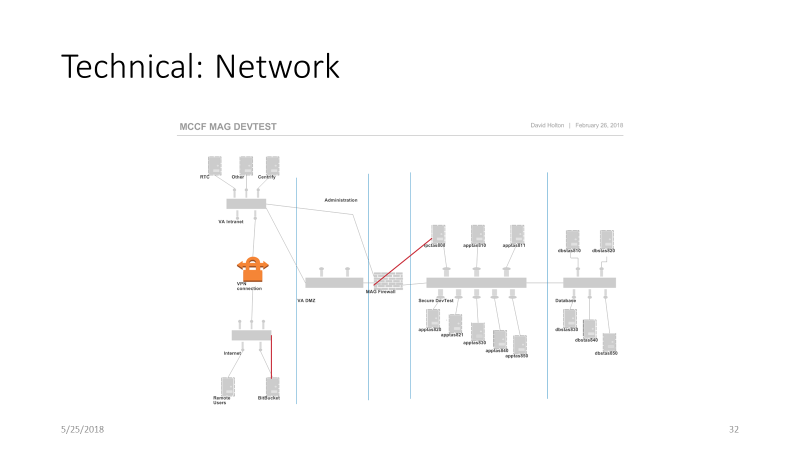
The figure below depicts an overview of the physical architecture of the MCCF EDI TAS application. it includes the following:

* The Development Team AWS development environment.
* The VA MAG environments.
* The VA VistA environments.
* Connections external to the TAS application, including IAM and the production VistA instances.
* Network connectivity between the environments.

More detailed diagrams for each of the environments are also included later in this section.

Figure - MCCF EDI TAS Physical Architecture Overview





## Security and Privacy

### Security

MCCF EDI TAS will leverage internal authentication and authorization mechanisms, but the expectation is that prior to national deployment, access services will be provided by the IAM program, including SSO. The components that run on VistA servers will leverage VistA access management controls.

A majority of the controls for MCCF EDI TAS will be inherited either nationally or by the Enterprise Operations and Field Operations common controls from the service lines.

These controls also apply to MCCF EDI TAS components running within the EO cloud within the Regional Data Centers.

Most of the audit and logging controls will be provided at the OS level with the use of the Enterprise Operations Security Information and Event Management (SIEM). The application shall be designed to leverage the OS-logging abilities for security-relevant events.

### Privacy

Connections to and from VistA via REST Resources will all be bound by Hypertext Transfer Protocol Secure (HTTPS) connections. Access to PHI and PII is only allowed in production environments. VistA REST Resources can only be deployed to those environments after Authorization To Operate (ATO) compliant testing has been completed. No PHI or PII is allowed in development environments.

Since MCCF EDI TAS will use VistA as the data store for PHI and PII, privacy considerations are very dependent on VistA privacy controls. Data will be encrypted in transmission to protect confidentiality of data, using TLS over HTTPS.

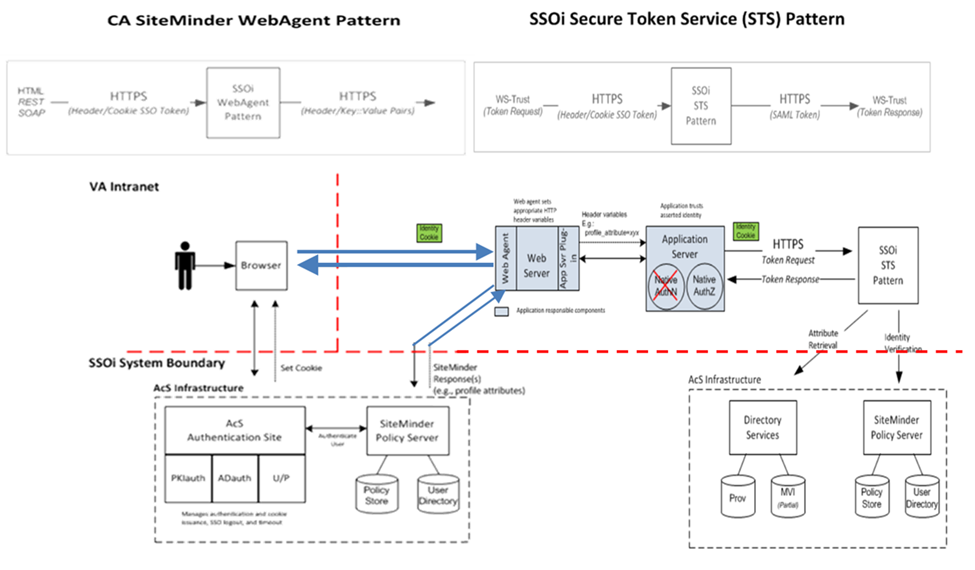
More detailed security and privacy control and design considerations will be included in this section as they are developed.

### Security

MCCF EDI TAS will leverage enterprise authentication and authorization mechanisms for national deployment, where access services will be provided by the IAM program and services, including SSOi (Single Sign-on Internal), STS (Secure Token Service, and Provisioning. The components that run on VistA servers will leverage IAM-VistA integration for access management controls.

A majority of other security controls for MCCF EDI TAS will be inherited either nationally or by the VA Enterprise Cloud common controls.

#### Authentication and Authorization with IAM



The TAS application will integrate with the IAM service as follows:

1. The User has already successfully logged in (User Authentication) and has been using his SSO token to access the application ((App/API Proxy Protection). The User's actions in the application requires access to other backend services or applications.
2. The Application, using the User's SSO cookie, makes a WS-Trust request to the SSOi STS service.
3. The SSOi STS service authenticates and authorizes the Application to act on behalf of user, validates the SSO token and retrieves the required data and finally generates, signs and returns the SAML token.

To restate: When a user accesses the entry point for the application (e.g. that which loads Angular), an Apache module called “Web Agent” which is part of the SSOi service, will intercept the call and check for an access token. If there is no access token, there’s a redirect to a PIV login screen. Upon login, there’s a redirect back and this time Apache will see the access token and let the TAS app load. Upon receipt of the proprietary token from the Web Agent, there’s a call to a STS server which swaps this out for a SAML token. This is the token to be passed throughout the entire system.

Because Angular is a purely static application run entirely in the web browser, there is no concept of cookies or sessions without an additional server-side component. Therefore, an additional layer must handle the storage of session state. Angular will call this intermediate TAS service layer with cookie information. The intermediate layer will handle API calls.

When the intermediate layer calls an API, it will pass security headers (e.g. ACCESSROLES) in addition to passing the SAML token. These security headers, plus the SAML header information will be carried between API calls. Because the API calls are JSON, not SOAP, this SAML header information will not be used in an officially SAML manner; rather, SAML token information will be read on-demand. The SAML information will be passed from API to API so each API has access to appropriate SAML information.

##### Authorization

IAM handles authentication, but authorization is custom based on the roles assigned to each user during IAM Provisioning. Requests contain the HTTP\_ACCESSROLES header with the format "ROLE1^ROLE2^ROLE3". The TAS API handles the internal mechanics of parsing these roles. Once the roles are parsed, they are available through a shared service in the TAS API and can be used to control access to any component or element at any layer of the TAS architecture.

Operations can be locked down to group by passing the required group to req.authorize.

This lock down happens in the TAS endpoint versions.

Example:

module.exports.delete = async (req, res) => {

//+ authorize accepts array or value

req.authorize('SBTEST\_ADMIN')

const user\_id = req.user.\_id

const id = req.params.id

await sbtest.delete(user\_id, id)

return response.sendJson(res, 204, {})

}

To allow more than one group, use the array form:

req.authorize(['SBTEST\_WRITER', 'SBTEST\_ADMIN'])

Authorization is based on the ACCESSROLES header. IAM will supply the HTTP\_ACCESSROLES header.

API calls check this header against requires roles in a manner similar to the following pseudocode:

IF required\_group in users\_iam\_group\_list:

ALLOW;

ELSE

DENY;

This creates a role-based access control model.

This is checked per API call. Each TAS API endpoint operation (e.g. GET, POST) will check the roles with a check following a model similar to aspect-oriented programming as seen in the following pseudocode:

FUNCTION abc:

authorize('SBTEST\_ADMIN')

...

...

RETURN result

This example locks function "abc" down to the SBTEST\_ADMIN role. Unauthorized access will result in an HTTP 401 response.

There is no permission model. Instead, granularity is at the role-level.

The UI will also use data structures based on the HTTP\_ACCESSROLES data. The UI will load the roles information from the API via an API call. This call will abstract some of the details of the HTTP\_ACCESSROLES header to ease development for UI developers.

## Service Oriented Architecture / ESS Detailed Design

Enterprise Design Patterns (EDPs), which include a Microservices architecture. The details regarding which EDPs are included in section 4.5 Enterprise Architecture, and the details of services that will be implemented are included in section 6.2.1.2 Services Layer Design and section 6.2.1.3 Data Access Services Design.

### Service Integration Flow

**Error! Reference source not found.** shows a data flow diagram that follows the layers in the Conceptual Architecture and adds details of the data flowing between the components in each layer. For example, it identifies the FHIR resources that will flow from the FHIR API in the Services layer to the ESB in the Messaging layer.

Figure 39: Service Integration Flow Diagram

Presentation

Business Services

Rules Engine

Service Interface

User Interface

* Coverage, Condition, DiagnosticReport, EligibilityResponse, Encounter, Location, MessageHeader, OperationOutcome, Patient, Practitioner, Procedure, RelatedPerson, and Organization

HTTPS REST API

Business

HTTPS FHIR REST API

HTTPS REST API / MFT / FTP?

HTTPS FHIR REST API

Messaging

FSC \*

Enterprise Service Bus

* Coverage, Condition, DiagnosticReport, EligibilityResponse, Encounter, Location, MessageHeader, OperationOutcome, Patient, Practitioner, Procedure, RelatedPerson, and Organization

Services

\* will FSC also use FHIR?

FHIR API

## Interface Design Rules

U.S. Web Design Standards will be followed for the design, layout and styling of the MCCF EDI TAS user interface. More information is available at <https://standards.usa.gov/>.

# External System Interface Design

## Interface Architecture

The diagram below shows details regarding how the MCCF EDI TAS Architecture data flows between the components in each layer for MCCF EDI TAS as well as the interfaces with external systems. For example, the diagram identifies the FHIR resources that will flow from the FHIR API in the Services layer to the ESB in the Messaging layer. HealthShare provides Cache Classes for custom FHIR resources that could implement a VA FHIR Profile. Federation could be accomplished using the MVI identifiers as well as the ability to connect to all the VistA instances.

Figure 40: High-level Application Design

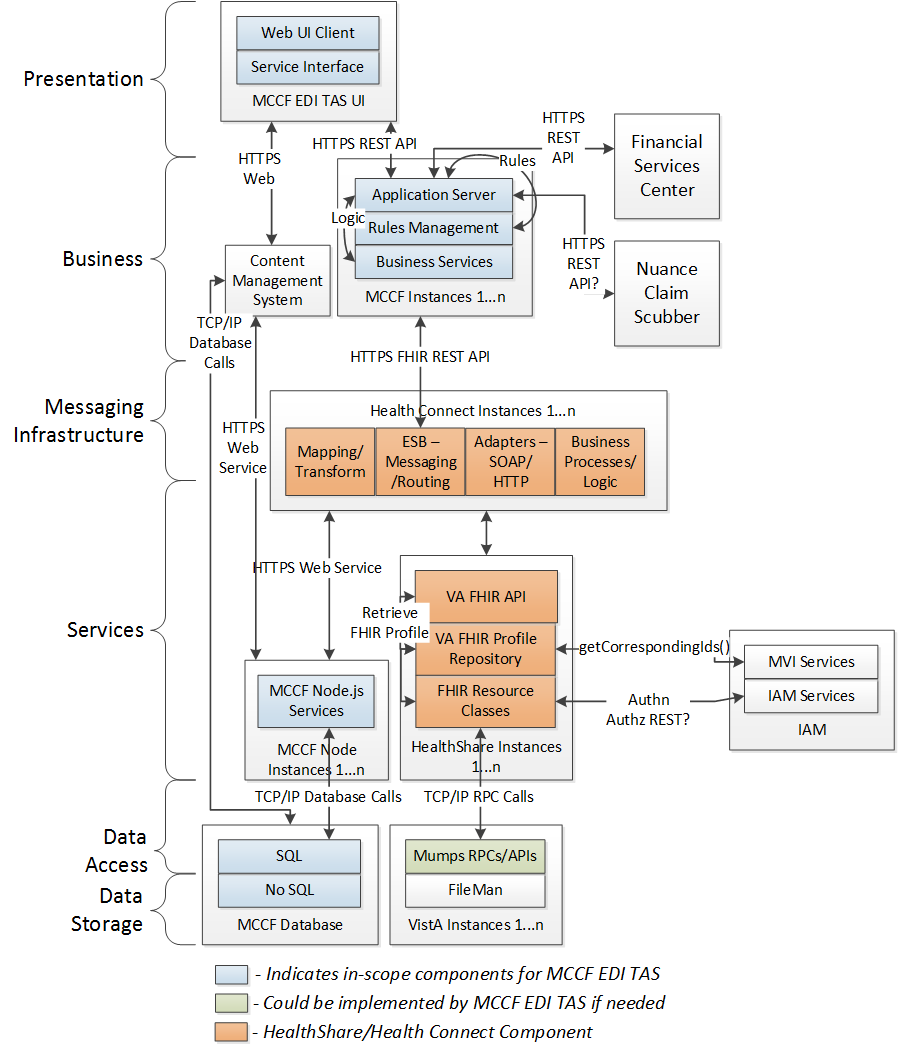


Figure : MCCF EDI TAS Interface Architecture

### TAS Web Development Ports

Table 11: DEV MAG Servers

| Source Domain Name | Source IP Address | Source Description | Source Port | Target Domain Name | Target IP Address | Target Description | Target Port | Functional Description | Directionality |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | \* | 80 | vaausapptas805 | xxx.xxx.xxx.186 | web server | 80 | Allow access to the web server port from the jump server for testing | bi-directional |
|  |  | \* | 443 | vaausapptas805 | xxx.xxx.xxx.186 | webserver | 443 | Allow access to the web server port from the jump server for testing | bi-directional |
| vaausapptas805 | xxx.xxx.xxx.186 | web server | x | vaausdbstas800 | xxx.xxx.xxx.157 | database server | 27017, 27018 | Allow access to the database cluster from the webserver | bi-directional |
| vaausdbstas800 | xxx.xxx.xxx.157 | database server | 22 | vaausapptas805 | xxx.xxx.xxx.186 | web server | 22 | Allow remote ssh capability | bi-directional |
| vaausfpctas801 | xxx.xxx.xxx.158 | dns server | 22 | vaausapptas805 | xxx.xxx.xxx.186 | web server | 22 | Allow remote ssh capability | bi-directional |
| vaausfpctas801 | xxx.xxx.xxx.158 | dns server | 22 | vaausdbstas800 | xxx.xxx.xxx.157 | database server | 22 | Allow remote ssh capability | bi-directional |

The following format is all that is required for a firewall request

Table 12: DEV MAG Connections

| ****SOURCE IP**** | ****DESTINATION IP**** | | | ****SERVICE/PORT**** | ****Business Case/Justification, Comments**** |
| --- | --- | --- | --- | --- | --- |
| **TAS Host - xxx.xxx.xxx.186 (WEB)** | **TAS Host - xxx.xxx.xxx.157 (DB)** | | | **TAS Port - 27017** | **TAS Application Writes to the TAS Database** |
| **TAS Host - xxx.xxx.xxx.157 (DB)** | | **TAS Host - xxx.xxx.xxx.186 (WEB)** | | **TAS Port - 27017** | **TAS Application Reads from the TAS Database** |
| **TAS Host - xxx.xxx.xxx.186 (WEB)** | | **TAS Host - xxx.xxx.xxx.157 (DB)** | | **TAS Port - 27018** | **TAS Application Writes to the TAS Database** |
| **TAS Host - xxx.xxx.xxx.157 (DB)** | | **TAS Host - xxx.xxx.xxx.186 (WEB)** | | **TAS Port - 27018** | **TAS Application Reads from the TAS Database** |
| **VPN/GFE/CAG** | | **TAS Host - xxx.xxx.xxx.186 (WEB)** | | **TAS Port - 80** | **Access TAS Application** |
| **VPN/GFE/CAG** | | **TAS Host - xxx.xxx.xxx.158 (DNS)** | | **TAS Port - 8080** | **Leverage Jenkins CI Service to Create Application Build** |
| **TAS Host - xxx.xxx.xxx.158 (DNS)** | | **https:// xxx.xxx.xxx.2** | | **TAS Port - 80** | **Populate TAS Bit Repository from Hosting at** [**Bitbucket.org**](http://Bitbucket.org) |
| **VPN/GFE/CAG** | **TAS Host - xxx.xxx.xxx.158 (DNS)** | | **TAS Port - 80** | | **Retrieve source code from TAS repository** |

Table 13: CI MAG Servers

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Source Domain Name | Source IP Address | Source Description | Source Port | Target Domain Name | Target IP Address | Target Description | Target Port | Functional Description | Directionality |
|  |  | webserver | 80 |  |  | jump Server | 80 | Allow access to the web server port from the jump server for testing | bi-directional |
|  |  | webserver | 443 |  |  | jump Server | 443 | Allow access to the web server port from the jump server for testing | bi-directional |
|  |  | webserver | x |  |  | database server | 27017, 27018 | Allow access to the database cluster from the webserver | bi-directional |
|  |  | CI Server | 80 |  |  | jump server | 80 | allow access to the CI Server (Jenkins) | bi-directional |
|  |  | CI Server | 22 |  |  | web server | 22 | allow remote installation from CI Server | bi-directional |
|  |  | CI Server | 22 |  |  | database server | 22 | allow remote installation from CI Server | bi-directional |

### CIT MAG Servers

Table 14: CI MAG Servers

| Source Domain Name | Source IP Address | Source Description | Source Port | Target Domain Name | Target IP Address | Target Description | Target Port | Functional Description | Directionality |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | Webserver | 80 |  |  | jump Server | 80 | Allow access to the web server port from the jump server for testing | bi-directional |
|  |  | Webserver | 443 |  |  | jump Server | 443 | Allow access to the web server port from the jump server for testing | bi-directional |
|  |  | Webserver | x |  |  | database server | 27017, 27018 | Allow access to the database cluster from the webserver | bi-directional |
|  |  | CIT Server | 80 |  |  | jump server | 80 | allow access to the CI Server (Jenkins) | bi-directional |
|  |  | CIT Server | 22 |  |  | web server | 22 | allow remote installation from CI Server | bi-directional |
|  |  | CIT Server | 22 |  |  | database server | 22 | allow remote installation from CI Server | bi-directional |

### SQA MAG Servers

Table 15: CI MAG Servers

| Source Domain Name | Source IP Address | Source Description | Source Port | Target Domain Name | Target IP Address | Target Description | Target Port | Functional Description | Directionality |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | webserver | 80 |  |  | jump Server | 80 | Allow access to the web server port from the jump server for testing | bi-directional |
|  |  | webserver | 443 |  |  | jump Server | 443 | Allow access to the web server port from the jump server for testing | bi-directional |
|  |  | webserver | x |  |  | database server | 27017, 27018 | Allow access to the database cluster from the webserver | bi-directional |
|  |  | SQA Server | 80 |  |  | jump server | 80 | allow access to the CI Server (Jenkins) | bi-directional |
|  |  | SQA Server | 22 |  |  | web server | 22 | allow remote installation from CI Server | bi-directional |
|  |  | SQA Server | 22 |  |  | database server | 22 | allow remote installation from CI Server | bi-directional |

### UAT MAG Servers

Table 16: CI MAG Servers

| Source Domain Name | Source IP Address | Source Description | Source Port | Target Domain Name | Target IP Address | Target Description | Target Port | Functional Description | Directionality |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | webserver | 80 |  |  | jump Server | 80 | Allow access to the web server port from the jump server for testing | bi-directional |
|  |  | webserver | 443 |  |  | jump Server | 443 | Allow access to the web server port from the jump server for testing | bi-directional |
|  |  | webserver | x |  |  | database server | 27017, 27018 | Allow access to the database cluster from the webserver | bi-directional |
|  |  | UAT Server | 80 |  |  | jump server | 80 | allow access to the CI Server (Jenkins) | bi-directional |
|  |  | UAT Server | 22 |  |  | web server | 22 | allow remote installation from CI Server | bi-directional |
|  |  | UAT Server | 22 |  |  | database server | 22 | allow remote installation from CI Server | bi-directional |

## Interface Detailed Design

Below are the interfaces currently identified for MCCF EDI TAS. Many of the details are not known at this time. These details will be provided as they become available.

### eBilling ICD

The eBilling ICD will reside at the VA Sharepoint Design Document section on the eBillingpagelocatedat:.https://vaww.oed.portal.va.gov/pm/hape/ipt\_5010/EDI\_Portfolio/eBilling/Forms/AllItems.aspx

Table 17: FSC Interface Design

| Interface Description | Financial Service Center |
| --- | --- |
| **Port(s)** | Unsure |
| **Source** |  |
| **Destination** | 2-way traffic |

Table 18: Planned Nuance Claim Scrubber Interface Design

| Interface Description | Nuance Claim Scrubber |
| --- | --- |
| **Port(s)** | Unsure |
| **Source** |  |
| **Destination** | 2-way traffic |

Table 19: IAM Interface Design

| Interface Description | IAM |
| --- | --- |
| **Port(s)** | IAM SSOi the ports are 44441 (Accounting) 44442 (Authentication) and 44443 (Authorization)  IAM STS uses port 443 for SSL/TLS |
| **Source** | 2-way traffic |
| **Destination** | 2-way traffic |

# Human-Machine Interface

## Inputs

MCCF EDI TAS will employ a Web-based user interface for the application. It will be developed to allow access through mobile platforms as well.

## Outputs

Initially the MCCF EDI TAS will provide the same reports that are available currently in the existing applications used for EDI transaction processing. The MCCF reports that will be ported over to TAS are defined in the Product Team features, specifially F7 ECME OPECC reports. The details describing the data elements included in these reports is contained in sections 5.3 Data View.

## Navigation Hierarchy

The navigation structure for MCCF EDI TAS has not been designed at this time. As the designs of the navigation structure is finalized, the details in this section will be updated.

### Screens

As the MCCF Product teams roll out the screens for specific functionality, the details for those screens will be included in separate design documentation or technical manuals.

# Appendix A: STAT Team Diagrams

## STAT Team Diagram Reference to Equivalent SDD Detailed Diagram

|  |  |  |
| --- | --- | --- |
| **New STAT Team Diagram** | **Equivalent Existing SDD Diagram** | |
| CI\_EDE\_ENV\_MCCFEDITAS\_04182018 | Test Environment Conceptual Infrastructure Diagram | |
| CIT\_EDE\_ENV\_MCCFEDITAS\_04182018 | Test Environment Conceptual Infrastructure Diagram | |
| CONTEXT\_MCCFEDITAS\_04182018 | MCCF EDI TAS Conceptual Architecture | |
| CPA\_MCCFEDITAS\_04182018 | MCCF EDI TAS Package Management Process | |
| CSD\_MCCFEDITAS\_04192108 | High-level Application Design | |
| DEV\_EDE\_ENV\_MCCFEDITAS\_04182018 | Test Environment Conceptual Infrastructure Diagram | |
| ENV\_MCCFEDITAS\_04182018 | MCCF EDI TAS Package Management Process | |
| SQA\_EDE\_ENV\_MCCFEDITAS\_04182018 | Test Environment Conceptual Infrastructure Diagram | |
| UAT\_EDE\_ENV\_MCCFEDITAS\_04182018 | Yes | Test Environment Conceptual Infrastructure Diagram |

# Attachment A – Approval Signatures

This section is used to document the approval of the System Design Document. The review should be conducted face to face where signatures can be obtained ‘live’ during the review. If unable to conduct a face-to-face meeting, then it should be held via LiveMeeting and concurrence captured during the meeting. The Scribe should add /es/name by each position cited. Example provided below.

The Business Sponsor and Project Manager are required to sign.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Signed: Date:

Frank Annecchini

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Signed: Date:

Enrique Gomez

1. Additional Information
   1. Identification of Technology and Standards

Health Level 7 (HL7) Fast Health Interoperability Resources (FHIR)

HIPAA EDI

ASC X12

* 1. Constraining Policies, Directives and Procedures

VA6500 Security Handbook

VA Compliance Epics – DEA, SEC and 508

* 1. Requirements Traceability Matrix

The MCCF EDI TAS RTM is available in the Rational Team Concert RM project for the system

* 1. Packaging and Installation

Outline any special considerations for software packaging and installation.

* 1. Design Metrics

Describe all metrics to be used during the design activity.

Template Revision History

| Date | Version | Description | Author |
| --- | --- | --- | --- |
| June 2015 | 2.10 | Changed Heading 1 default setting to eliminate page break before | Process Management |
| May 2015 | 2.9 | Edited for Section 508 conformance and remediated with Common Look Office tool | Process Management |
| February 2015 | 2.8 | Incorporates revisions from PMAS Reform Lockdown; namely removing requirements for information that can be obtained from other PMAS authoritative sources. | Andrew Slawter, Office of Technology Strategies |
| September 2014 | 2.7 | Adds Enterprise Shared Services terms and requires AERB Compliance Certificate attachment. | Process Management |
| August 2014 | 2.6 | Signature block update authorized by AERB CR\_018934 | Process Management |
| March 2014 | 2.5 | Section 508 repairs to new version approved by AERB Chair approved | Process Management |
| August 2013 | 2.3 | Replaced the Service Architecture sub-section with new sub-sections for consumed and provided services. Also applied miscellaneous feedback from VA team. | ASD Enterprise Shared Services (ESS) Work Group |
| June 2013 | 1.3 | Upgraded to MS Office 2007-2010 format | Process Management |
| June 2013 | 1.2 | Address inconsistencies in Section 3, Conceptual Design, Correct headings | Process Management |
| March 2013 | 1.1 | Formatted to documentation standards and edited for Section 508 conformance | Process Management |
| January 2013 | 1.0 | Initial Document | PMAS Business Office |

The Template Revision History pertains only to the format of the template. It does not apply to the content of the document or any changes or updates to the content of the document after distribution.

The Template Revision History can be removed at the discretion of the author of the document.

See TOGAF® 9.1, Part III: ADM Guidelines & Techniques, Gap Analysis on TOGAF Website at <http://pubs.opengroup.org/architecture/togaf9-doc/arch/chap27.html>

1. Design Pattern Library https://vaww.portal2.va.gov/sites/asd/TechStrat/edp/Design%20Pattern%20Library/Forms/AllItems.aspx [↑](#footnote-ref-2)