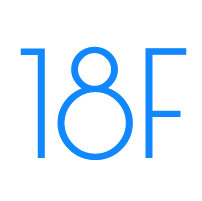
**Deliver To:**

****

Request for Quotation (RFQ)

4QTFHS150004

SIN 132-51

General Services Administration (GSA) Federal Acquisition Service, Integrated Technology Service National IT Commodity Program

401 West Peachtree Street NW, Suite 820 Atlanta, GA 30308

**Created by:**



1900 Campus Commons Drive, Suite 150

Reston, Virginia 20191

703.581.6500

Rich Zimmerman

Contracts Manager

Defender System

Design Document

2015

Table of Contents

1.0 Introduction 1

2.0 Document References 1

3.0 Design Assumptions and Constraints 1

3.1 Design Assumptions 1

3.2 Design Constraints 1

4.0 System Overview 2

5.0 Architecture Alternatives 2

6.0 System Architecture 2

6.1 Data Architecture 2

6.2 Execution Architecture 3

6.3 Development / Software Architecture 4

6.4 Operations Architecture 5

Revision History

**Note:** If this is the first release, type “Original” in the “Summary of Changes” column.

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Name of Author | Summary of Changes |
| Alpha | 6/22/2015 | Tony Petruccelli | Created doc |
| Alpha | 6/23/2015 | Tony Petruccelli | Updated to reflect design details |
| Alpha | 6/26/2015 | Tony Petruccelli | Updated based on peer review |
| Alpha | 06/30/2015 | Tony Petruccelli | Final updates |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

CONTENTS

[1.0 Introduction 1](#_Toc422849472)

[2.0 Document References 1](#_Toc422849473)

[3.0 DESIGN Assumptions and Constraints 1](#_Toc422849474)

[3.1 Design Assumptions 1](#_Toc422849475)

[3.2 Design Constraints 1](#_Toc422849476)

[4.0 System Overview 1](#_Toc422849477)

[5.0 SYSTEM ARCHITECTURE 2](#_Toc422849478)

[5.1 Execution Architecture 2](#_Toc422849479)

[5.2 Development / Software Architecture 3](#_Toc422849480)

[5.3 Operations Architecture 3](#_Toc422849481)

[Attachment A—Acronyms and Abbreviations](#_Toc419885587)

# Introduction

The Defender brings together the FDA formal data with less formal information across the internet, and acts as a big picture utility where users can go get a quick glance at current, known FDA enforcement actions, their location, and affected areas. The site additionally presents users internet trending and social media data to see related information on FDA recalls/enforcements.

# Document References

* OpenFDA: (<https://open.fda.gov/>)
* US Digital Services Playbook (<https://playbook.cio.gov/>)
* The MEAN stack (<http://mean.io>)
* Mapbox (<https://www.mapbox.com/>)
* Bootstrap (<http://getbootstrap.com/>)
* Karma (<http://karma-runner.github.io/0.12/index.html>)
* Grunt (<http://gruntjs.com/>)
* Github (<https://github.com/>)
* Travis CI (<https://travis-ci.org/>)
* Morgan (<https://github.com/expressjs/morgan>)
* AWS (<http://aws.amazon.com/>)

# Design Assumptions and Constraints

## Design Assumptions

Assumptions and related known dependencies that affect the project are as follows:

* Responsive design must be leveraged to support as many platforms and form factors as possible
* The prototype will support a variety of browsers; however versions of old browsers may not be supported

## Design Constraints

The following are design constraints with related known impacts for this system development:

* The prototype must integrate data from OpenFDA (<https://open.fda.gov/>)
* The prototype must use at least five modern and open-source technologies, regardless of architectural layer (frontend, backend, etc.)
* The prototype must make use of an API, by either consuming or providing one RESTfully
* Configuration management must be supported
* The prototype repository must be accessible in a publicly-accessible version control system that supports git
* The prototype must be deployed on an Infrastructure as a Service (IaaS) or Platform as a Service (PaaS) provider
* A continuous integration system will be used to automate the running of tests and continuously deploy code to the IaaS or PaaS provider
* Continuous monitoring must be supported
* The prototype must be deployed to a public URL
* The prototype and underlying platforms used to create and run the prototype must be openly licensed and free of charge

# System Overview

Defender is delivered to the user via a URL of <https://defender.ionep.io>. The site was designed using responsive design principles. The user is presented with search criteria to query the OpenFDA APIs in order to retrieve recall report data. The application, residing within AWS in turn presents this information in a variety of ways to the user. The OpenFDA results can then be augmented through the search and display of related information from internet trend and social media sites.

# Architecture Alternatives

We considered a number of alternatives for this prototype including Python, Ruby, and Java to name but a few. In the end, we decided to use the MEAN stack to facilitate rapid development of prototype iterations and to leverage the commonality of JavaScript knowledge across our development team. The selection of the MEAN stack in turn made clarified some additional choices for logging and continuous integration. Similarly, by choosing AWS for the deployment environment, other choices for things such as continuous monitoring were constrained to those offered by AWS.

# 

# System Architecture

This section describes the logical system and subsystem architecture selected for Defender. The architecture reflects modern technologies that allow for the addition of incremental functionality and external integration in rapid iteration.

## Data Architecture

For Release 1 of this prototype, data is consumed, parsed, aggregated and displayed within various portions of the architecture. No data is persisted by the application as of yet; MongoDB is available for flexibility in future sprints but as of Release 1, its functionality is not used.

For Release 1, a significant amount of data is made available via API calls to OpenFDA, which provides data from FDA Enforcement Reports. These APIs are:

1. OpenFDA Food Recalls
2. OpenFDA Drug Recalls
3. OpenFDA Device Recalls

The remainder of data is provided in the form of map and map data, and made available via the combination of Leaflet and Mapbox.

## Execution Architecture

This application is being deployed inside Amazon Web Services EC2 environment. The site is rendered through HTML with Javascript. Leveraging Bootstrap for responsive design, the site is viewable via desktop/laptop and mobile platforms and is currently functional on Chrome v43, IE v.11, Firefox v43, and Safari v8.

At its highest level, the AWS environment consists of the node.js container and the MongoDB NoSQL database. Note that for Release 1, MongoDB was made available but no functionality requiring a database was required. It is however included in the architecture and available for anticipated functionality in future sprints.



**Exhibit 5-1: Execution Architecture**

## Development / Software Architecture

The primary technologies used are represented by the MEAN stack (MongoDB, Express, AngularJS, node.js). For additional geospatial display capabilities, the application leverages Mapbox, an open source mapping API built on top of Leaflet. To facilitate test-driven development, a combination of Karma and Grunt are used. Functional testing is performed through a combination of manual testing and automated testing using Selenium. Morgan and AWS CloudWatch supply logging and continuous monitoring capabilities. Github provides collaboration and code management to ensure standard builds and deployment of the code. Continuous Integration is accomplished through integration of the github repository and Travis CI with automated deployments to AWS. Developers are notified of build issues through auto-generated emails. Please see Exhibit 5-2 for a depiction of the development/software architecture.



**Exhibit 5-2: Development / Software Architecture**

## Operations Architecture

The Operations Architecture supports system logging, application logging, and continuous monitoring to ensure availability and provide a window into potential functional and performance improvements. Morgan provides HTTP request logging at the application container level. Captured application log data is parsed and aggregated to collect the metrics that AWS CloudWatch uses for display of system and application performance as well as to send alerts when system or application performance exceeds acceptable standards. Server-level metrics are also captured by AWS CloudWatch.



**Exhibit 5-3: Operations Architecture**

Attachment A—Acronyms and Abbreviations

|  |  |
| --- | --- |
| **Acronym** | **Definition** |
| API | Application Program Interface |
| AWS | Amazon Web Services |
| CI | Continuous Integration |
| EC2 | Elastic Compute Cloud |
| IaaS | Infrastructure as a Service |
| MEAN | MongoDB, Express, AngularJS, node.js |
| PaaS | Platform as a Service |
| REST | Representational State Transfer |
|  |  |