



## **Intellect Solutions Technical Approach**



**Submitted in Response to:**

**Agile Service Delivery**

**4QTFHS150004**

**For:**

**The General Services Administration (GSA)  
Federal Acquisition Service, Integrated Technology Service  
National IT Commodity Program  
401 West Peachtree Street NW, Suite 820  
Atlanta, GA 30308**

**By:**

**Intellect Solutions LLC  
GSA Contract # GS-35F-144AA  
5602 Pickwick Road  
Centreville, VA 20120  
Dr. Sunny Singh, COO, Phone: (703) 898-1498**

## Table of Contents

<b>1.0</b>	<b>PROTOTYPE NAME .....</b>	<b>1</b>
<b>2.0</b>	<b>USE CASES ADDRESSED BY ADERS .....</b>	<b>1</b>
<b>3.0</b>	<b>INTRODUCTION.....</b>	<b>1</b>
<b>4.0</b>	<b>REQUIREMENTS DEVELOPMENT PHASE.....</b>	<b>1</b>
<b>5.0</b>	<b>PROTOTYPE DESIGN PHASE.....</b>	<b>2</b>
<b>6.0</b>	<b>TECHNICAL DESIGN PHASE .....</b>	<b>2</b>
<b>7.0</b>	<b>PROTOTYPE DEVELOPMENT PHASE .....</b>	<b>5</b>
<b>8.0</b>	<b>TESTING.....</b>	<b>8</b>
<b>9.0</b>	<b>CONFIGURATION MANAGEMENT AND SOURCE CODE DEPLOYMENT .....</b>	<b>10</b>
<b>10.0</b>	<b>SYSTEM SECURITY AND ONGOING MAINTENANCE .....</b>	<b>11</b>
<b>11.0</b>	<b>FUTURE ENHANCEMENTS .....</b>	<b>15</b>

## List of Figures

FIGURE 1: GITHUB REPOSITORY SOURCE CONTROL REPOSITORY .....	3
FIGURE 2: TIAGA.IO COLLABORATION SITE .....	4
FIGURE 3: AMAZON WEB SERVICE PROTOTYPE INSTANCE CONFIGURATION SUMMARY .....	4
FIGURE 4: LOGIN SCREEN TO AWS INSTANCE WITH AMI RELEASE INFORMATION REFERENCE.....	5
FIGURE 5: AGILE SPRINTS.....	6
FIGURE 6: USER STORIES DEVELOPED IN TAIGA.IO .....	6
FIGURE 7: DESIGN SESSIONS.....	7
FIGURE 8: DESIGN SESSIONS.....	7
FIGURE 9: TEAM COLLABORATION USING SLACK TOOL .....	8
FIGURE 10: DEMO TO THE INTERNAL TEAM .....	8
FIGURE 11: FEEDBACK FROM INTERNAL TEAM MEMBERS.....	9
FIGURE 12: KALI LINUX PENETRATION TESTING TOOL.....	9
FIGURE 13: USING NOTEPAD++ TO EDIT FILES.....	10
FIGURE 14: SYNCHRONIZING CHANGES TO GITHUB REPOSITORY .....	11
FIGURE 15: HIGH-LEVEL ARCHITECTURE DIAGRAM .....	12
FIGURE 16: AWS PERFORMANCE MONITORING FEATURES .....	13
FIGURE 17: PERFORMANCE BEFORE OPTIMIZATION .....	14
FIGURE 18: PERFORMANCE AFTER OPTIMIZATION .....	15

## 1.0 Prototype Name

Adverse Drug Effects Research System (ADERS)

## 2.0 Use Cases addressed by ADERS

After interviewing few sample users, Intellect Solutions team came up with two use cases that would be very useful to the end users:

- I was told to take certain drug, but I am not sure if I should take that drug. I would like to find out adverse effects of the drugs suggested to me and confirm that I have all the correct label and warning information for the drugs I am using.
- I would like to stress the importance on maintaining privacy during the search.

## 3.0 Introduction

Our site is designed to provide users with ability to research on adverse side effects of the brand name and generic drugs and medical devices. While building the prototype, Adverse Drug Effects Research System (ADERS), Intellect followed all the 13 plays from the US Digital Services Playbook to ensure we deliver successful Agile Service prototype to GSA 18F.

Below are the details of each play and how we followed them and customized to 18F's requirements.

## 4.0 Requirements Development Phase

*This section discusses our approach to Play 1:*

- *Play 1: Understand what people need*

Our approach began with studying prototype requirements and performing research on the APIs provided by open.fda.gov. After reviewing the APIs, we identified some sample users who might use that information and conducted interviews and meetings. GSA was also identified as a stakeholder based on the requirements in the RFQ. After these meetings, our team developed user stories in Taiga.io that addressed following information:

- Functionality
- Usability and 508 compliance
- Information displayed
- Security
- Technical designs and considerations

## 5.0 Prototype Design Phase

*This section discusses our approach to the following Plays:*

- *Play 2: Address the whole experience, from start to finish*
- *Play 3: Make it simple and intuitive*

Based on the requirements gathered from our sample users, Intellect developed high level user stories that describe end-to-end prototype behavior. We engaged an experienced team member to serve in the capacity of Usability expert and front-end developer. We developed mock-ups, created user flows, and addressed 508 compliant elements in our design. We researched different websites that are top-ranked in terms of usability such as

<http://www.rxlist.com/script/main/hp.asp>; <http://www.drugs.com/>; and

<https://www.myprime.com/> to understand latest design elements. Additionally, we also reviewed concepts used by 18F on their consulting projects and developed the user experience.

Our prototype includes other elements such as gather user feedback using Google forms and help for users to easily perform search.

## 6.0 Technical Design Phase

*This section discusses our approach to the following plays:*

- *Play 5: Structure budgets and contracts to support delivery*
- *Play 8: Choose a modern technology stack*
- *Play 9: Deploy in a flexible hosting environment*
- *Play 13: Default to open*

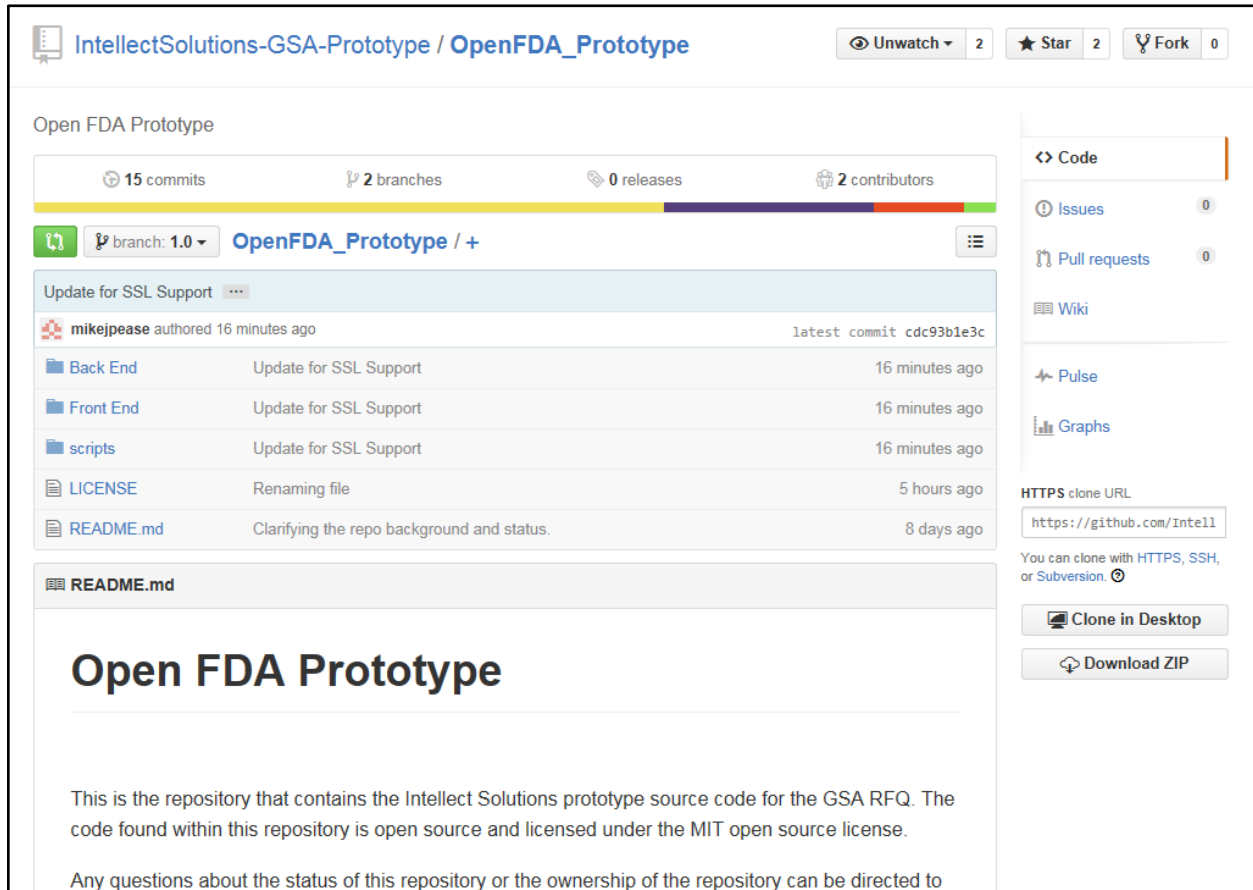
Intellect Solutions team chose all the sub-components that make up our prototype to align with US Digital Services Playbook, industry standard tools, industry leading practices, and 18F's approach to software development, to meet following criteria:

- Use latest and reliable tools used in the industry
- Increase reusability
- Open source and low startup cost
- Fast turn-around time

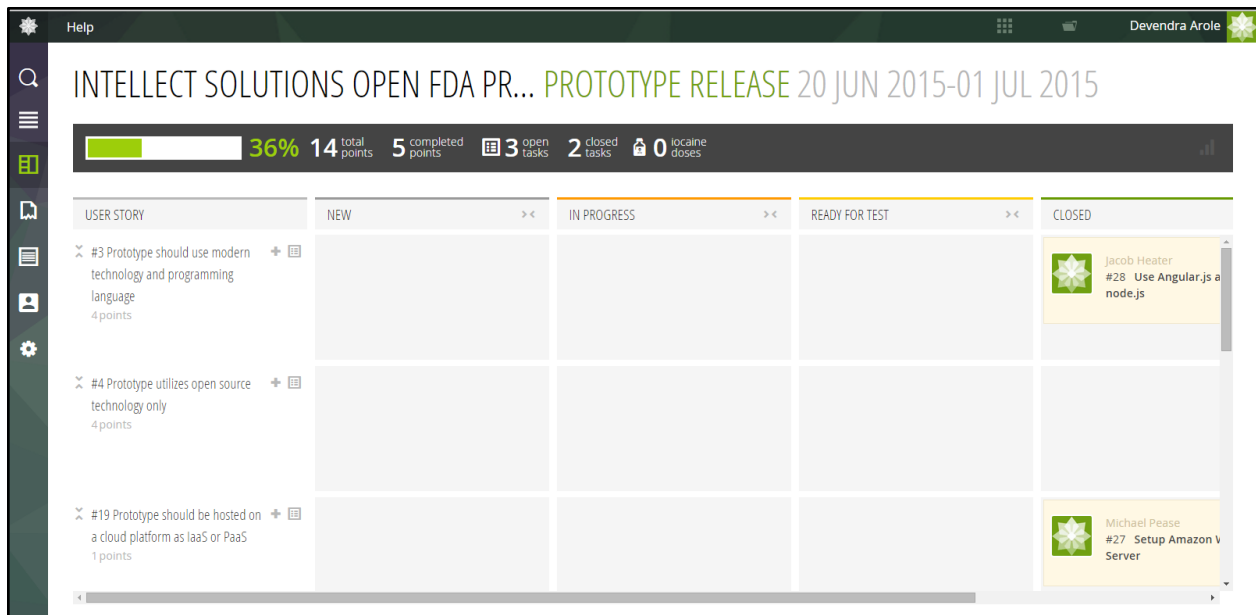
Intellect Solutions team used the following open-source and modern technologies:

- SaaS hosting provider selected to host the prototype is Amazon Web Services
- Source control repository is Github (Figure 1)
- Project management tool is open source and free Taiga.io (Figure 2)
- CentOS virtual server is hosted on Amazon Web Services (Figures 3 and 4)
- Docker Containers are used to package an application into a standardized unit
- Kali Linux platform is used for vulnerability testing

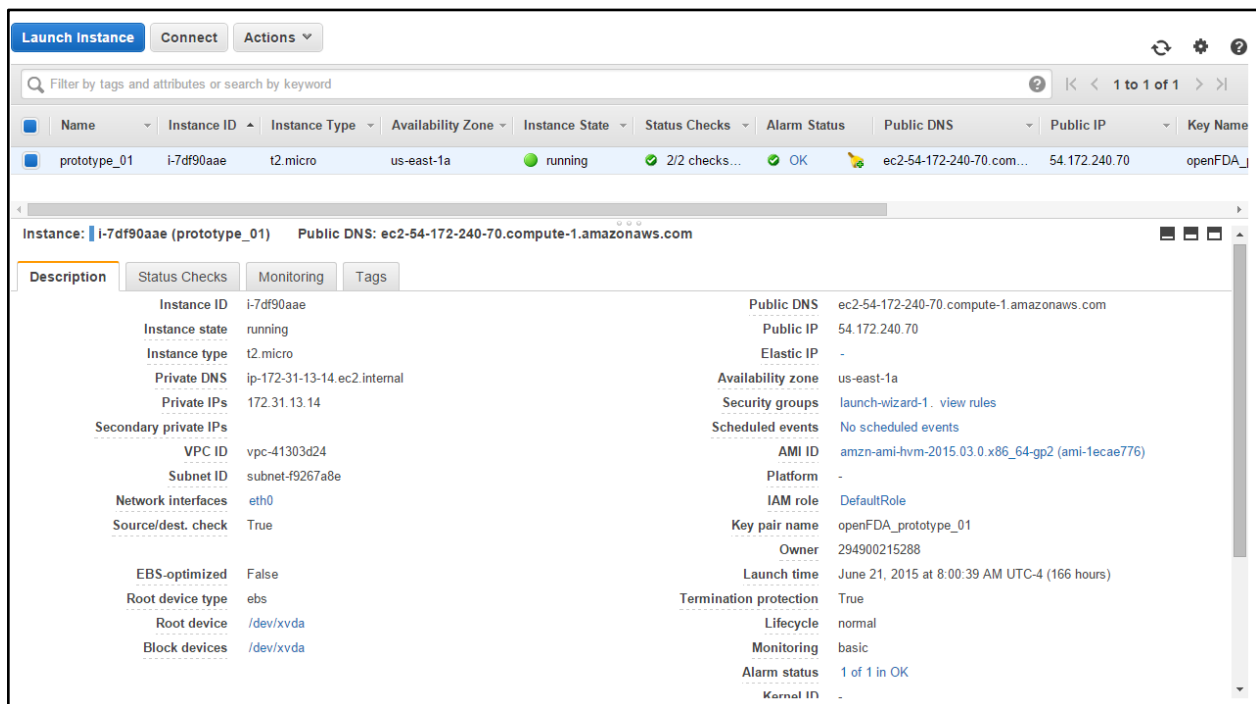
- Open VAS is used for vulnerability testing
- Front-end development framework is angular.js
- Back-end development framework is node.js
- JSON API specifications



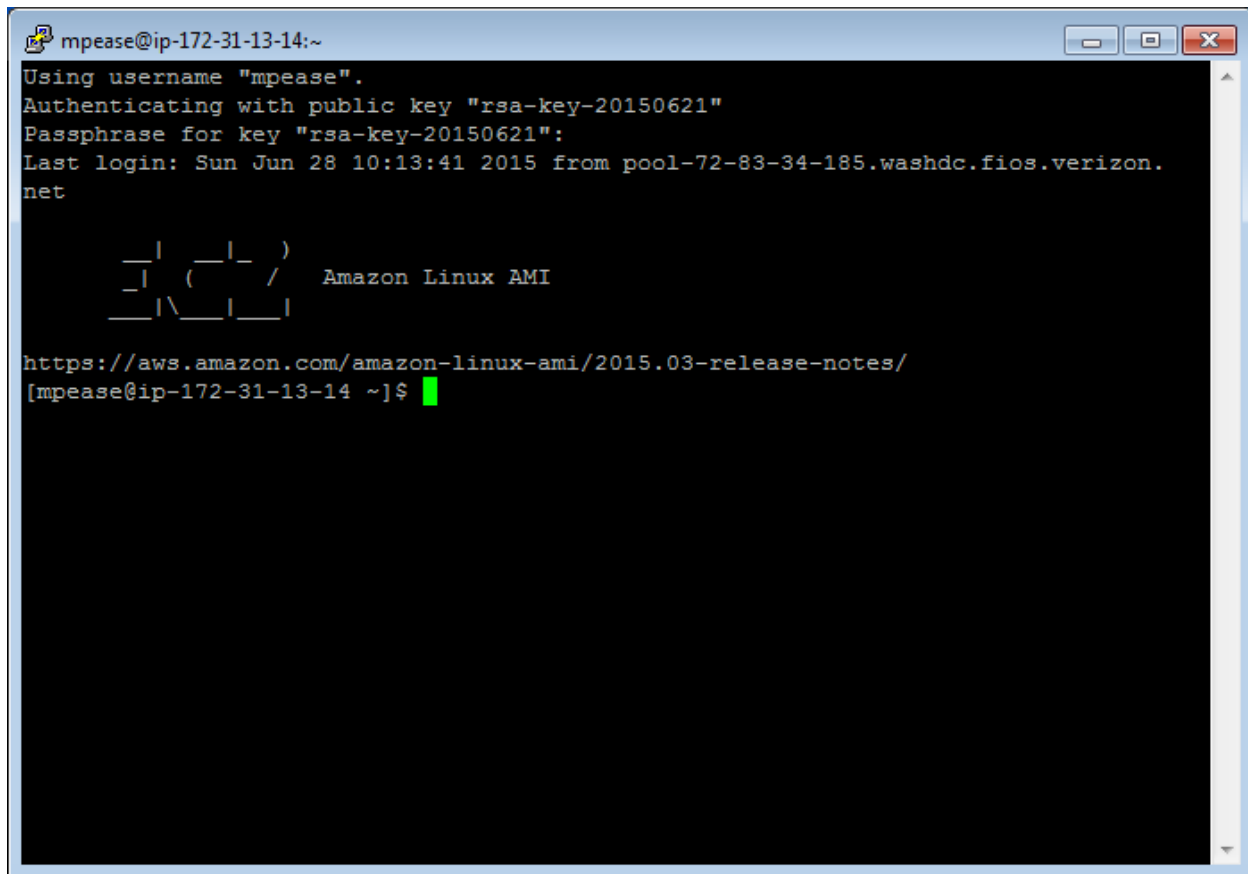
*Figure 1: GitHub Repository Source Control Repository*



*Figure 2: Tiaga.io Collaboration Site*



*Figure 3: Amazon Web Service Prototype Instance Configuration Summary*



*Figure 4: Login Screen to AWS Instance with AMI Release Information Reference*

## 7.0 Prototype Development Phase

*This section discusses our approach to the following plays:*

- *Play 4: Build the service using agile and iterative practices*
- *Play 6: Assign one leader and hold that person accountable*
- *Play 7: Bring in experienced teams*
- *Play 10: Automate testing and deployments*

Intellect Solutions team is adept at Agile development methodology, having done this for other government clients. After reviewing the requirements, gathering user stories by interacting with sample users via interviews and meetings, Intellect Solutions team quickly identified core team members required to develop the prototype, began daily Agile prints, and developed user stories (Figures 5 and 6).

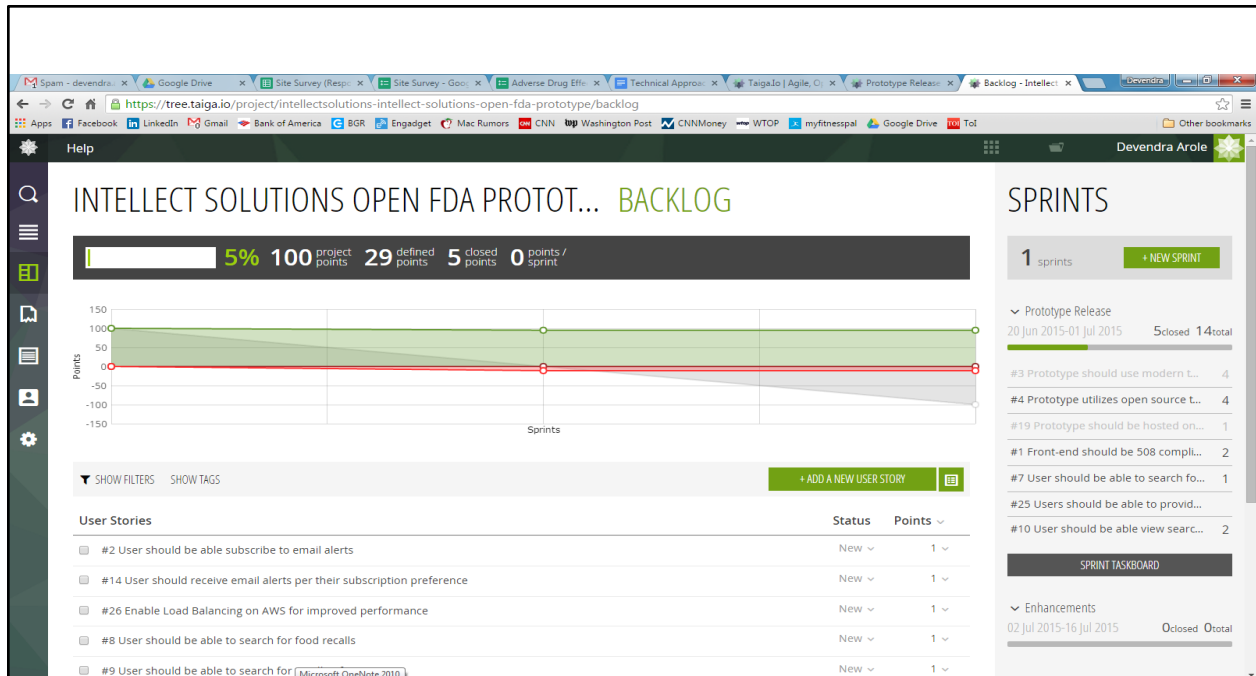


Figure 5: Agile Sprints

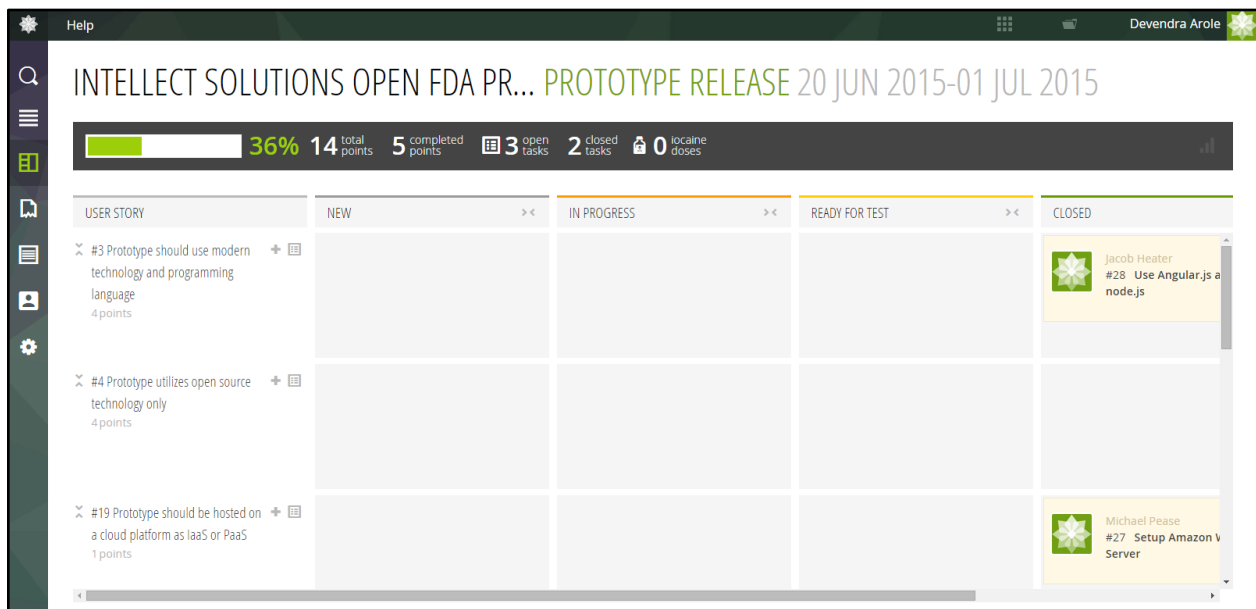


Figure 6: User Stories Developed in Taiga.io

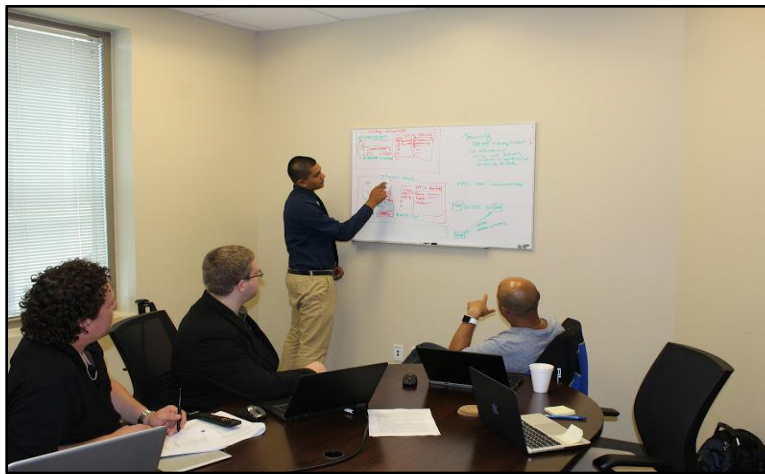
The work was based around software releases at the end of each sprint. Initial release focused on developing an alpha prototype in the short time frame requested by the government. We also identified subsequent releases to address feedback from unit testing and end user testing. We also identified some nice to have features that included functionality enhancements and UI improvements.



Intellect Solutions team identified a diverse team with varied experience to develop the prototype. Team members that supported the development of prototype represented the following labor categories under the *development pool 2*:

- Technical Architect
- Front End Developer
- Back End Developer
- DevOps Engineer

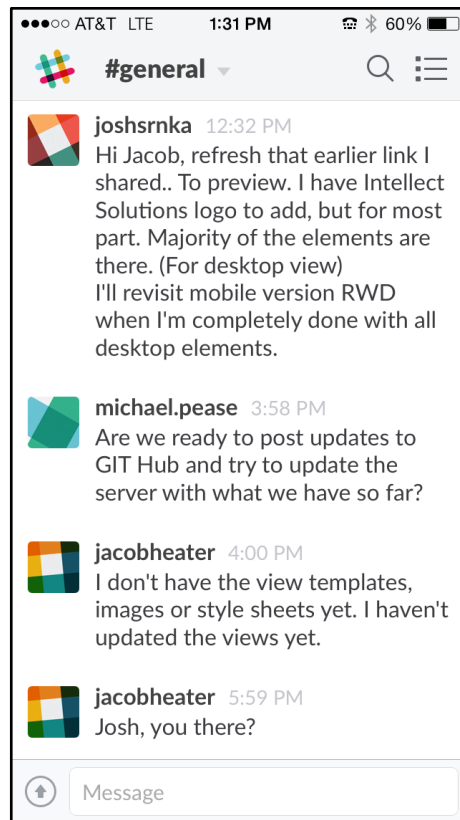
The Technical Architect played the role of the leader who was responsible for setting the tone of the engagement, execution, and overall quality. This team came together, shared their experience, reviewed the RFQ requirements, and quickly developed a prototype (Figures 7 through 9).



*Figure 7: Design Sessions*



*Figure 8: Design Sessions*



*Figure 9: Team Collaboration Using Slack Tool*

## 8.0 Testing

We performed the following type of testing as part of our methodology:

1. Functionality – We used JSFiddler.Net to create test AJAX scripts for submitting and validating Node.JS functionality to query and return data from OpenFDA. Additionally, we demonstrated the working prototype to the Intellect team members and invited immediate critique (Figures 10 and 11).



*Figure 10: Demo to the Internal Team*



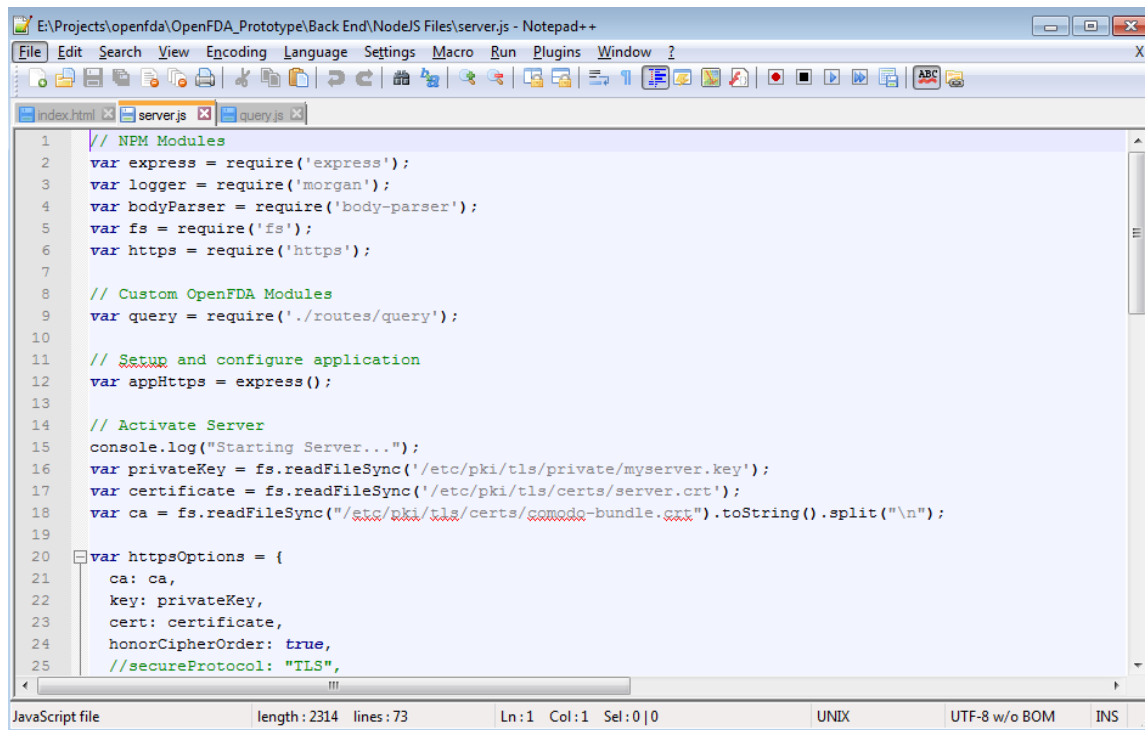
*Figure 11: Feedback from Internal Team Members*

2. Performance – We used review logs to monitor query and page response times then adjusted code and configurations based on log analysis results.
3. Security (through vulnerability management and penetration testing) – We used Kali Linux (Figure 12) and Open VAS platform tools to test exposed services for vulnerabilities; SSL Labs and Comodo for validating SSL configurations.



*Figure 12: Kali Linux Penetration Testing Tool*

4. Unit testing – We used text editors (Figure 13) to edit code files; preliminary functional testing was performed on local web services running on development workstations; resulting updates were submitted to the GitHub Repository.



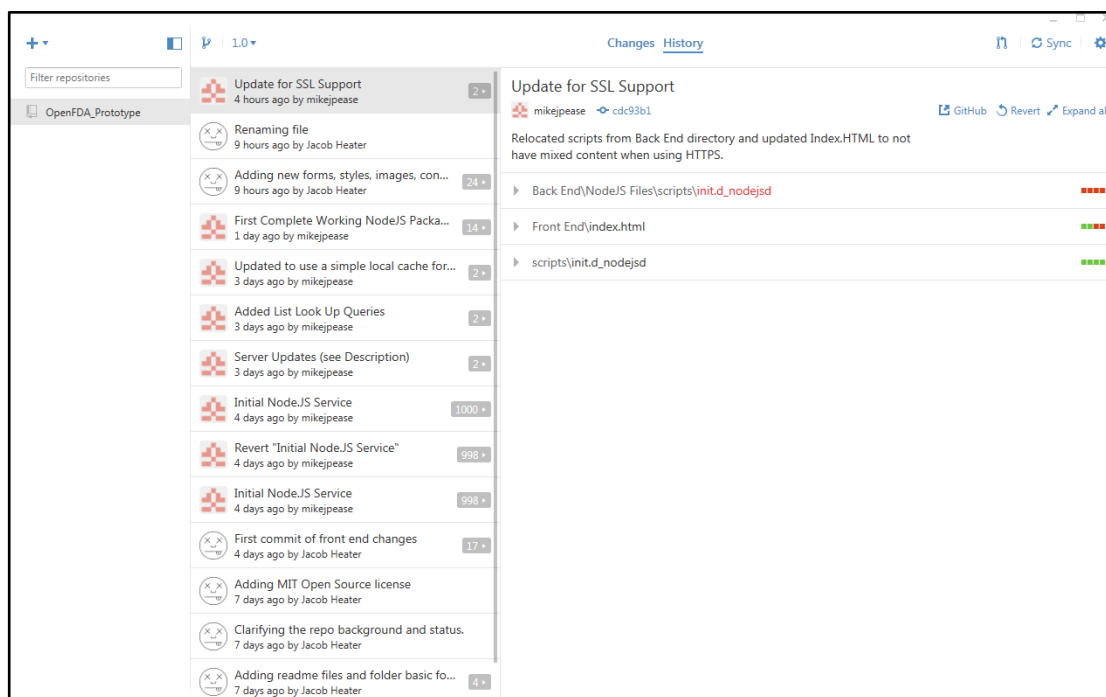
```
1 // NPM Modules
2 var express = require('express');
3 var logger = require('morgan');
4 var bodyParser = require('body-parser');
5 var fs = require('fs');
6 var https = require('https');
7
8 // Custom OpenFDA Modules
9 var query = require('../routes/query');
10
11 // Setup and configure application
12 var appHttps = express();
13
14 // Activate Server
15 console.log("Starting Server...");
16 var privateKey = fs.readFileSync('/etc/pki/tls/private/myserver.key');
17 var certificate = fs.readFileSync('/etc/pki/tls/certs/server.crt');
18 var ca = fs.readFileSync("/etc/pki/tls/certs/comodo-bundle.crt").toString().split("\n");
19
20 var httpsOptions = {
21   ca: ca,
22   key: privateKey,
23   cert: certificate,
24   honorCipherOrder: true,
25   //secureProtocol: "TLS",

```

*Figure 13: Using Notepad++ to Edit Files*

## 9.0 Configuration Management and Source Code Deployment

- Taiga.Io integrates with Github to check-in, check-out source code files, and for collaboration (Figure 14).
- Automated scripts are manually executed to update the server with the latest versions of the front and back end files.



*Figure 14: Synchronizing Changes to GitHub Repository*

## 10.0 System Security and Ongoing Maintenance

*This section discusses approach to the following plays:*

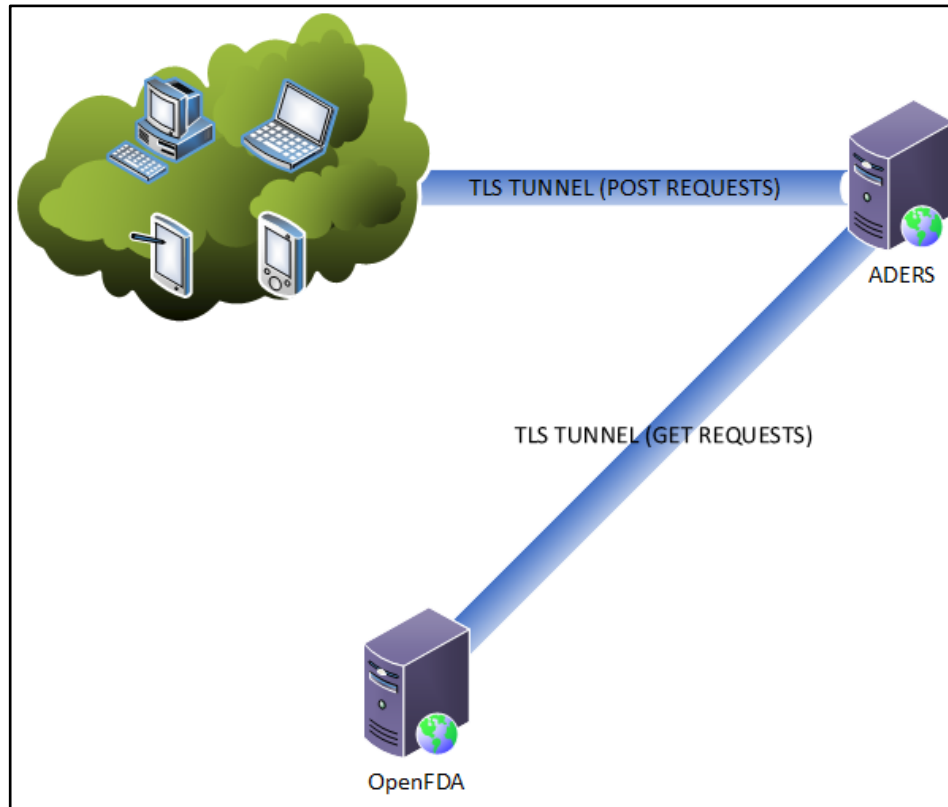
- *Play 11: Manage security and privacy through reusable processes*
- *Play 12: Use data to drive decisions*

Security and privacy played an important role in our approach. Intellect designed its prototype by placing huge emphasis on security up front unlike traditional software development projects where security is an afterthought or nice to have.

Our prototype is deployed on Amazon Web Services hosting provider which is FedRAMP certified - <http://aws.amazon.com/compliance/fedramp-faqs/> (Figure 15). All the connections made to our website used HTTPS TLS v1.0, v1.1, and v1.2 with Cipher modes that place preferences on TLSv1.2 and only used the TLSv1.0 and 1.1 for backward compatibility. Additionally, we used OpenVAS to perform weekly vulnerability scans and addressed the critical and high vulnerabilities exploitable from the internet within 15 days.

During the initial architecture evaluation and design security review, we noted that while the OpenFDA site allowed HTTPS for secure data transmission, the service only supported GET requests. As a result, a risk was identified that internet service providers (ISP's) and other man-in-the-middle situations could obtain information regarding users of the service by allowing them to capture the query parameters and identify drugs, devices, or other medical information that users researched. Our architecture attempts to mitigate this vulnerability used Node.JS to provide a server side query capability. Encrypted connections to the ADERS server platform were

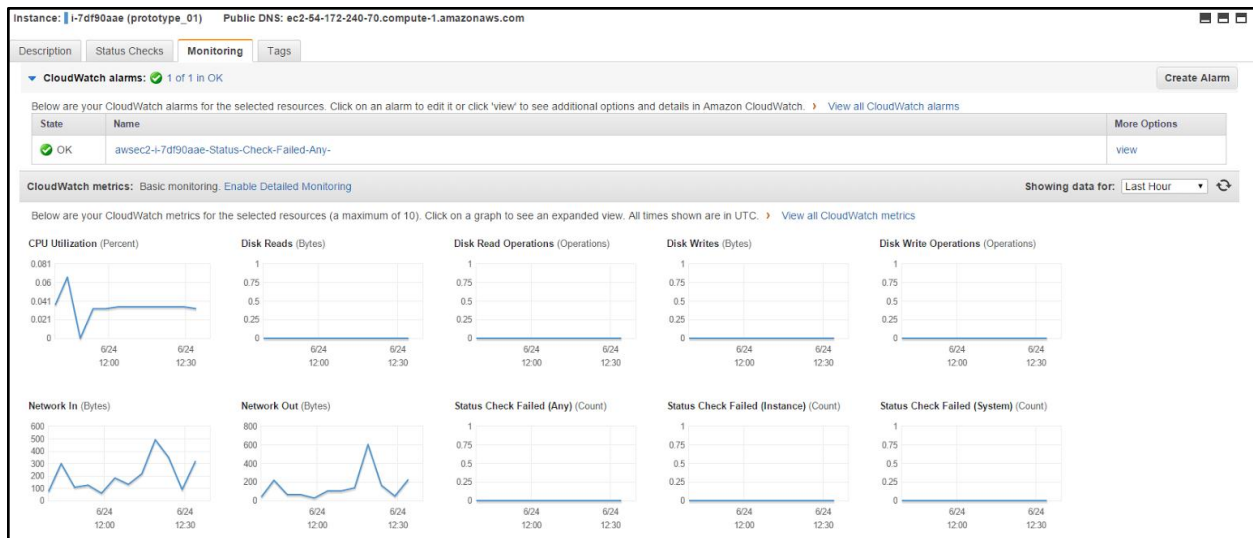
created for the requestor to send the query request to OpenFDA and then return the results to the requestor. While this did not eliminate the vulnerability, it anonymized the connection by preventing the attribution of the query parameters to the actual individual. To further protect the user, the web services were not configured to capture query parameters with requesting IP addresses to ensure the anonymity of the user query requests to OpenFDA.



*Figure 15: High-Level Architecture Diagram*

As part of our commitment to maintaining privacy of user information, we only captured information necessary to meet functionality of the system. Apache captured IP addresses as part of normal website logging for statistical, performance, and security monitoring.

AWS provides system level continuous monitoring to assist in data driven decisions (Figure 16). This helps us setup alerts, monitor system performance, and make necessary adjustments to the prototype behavior in a timely manner.



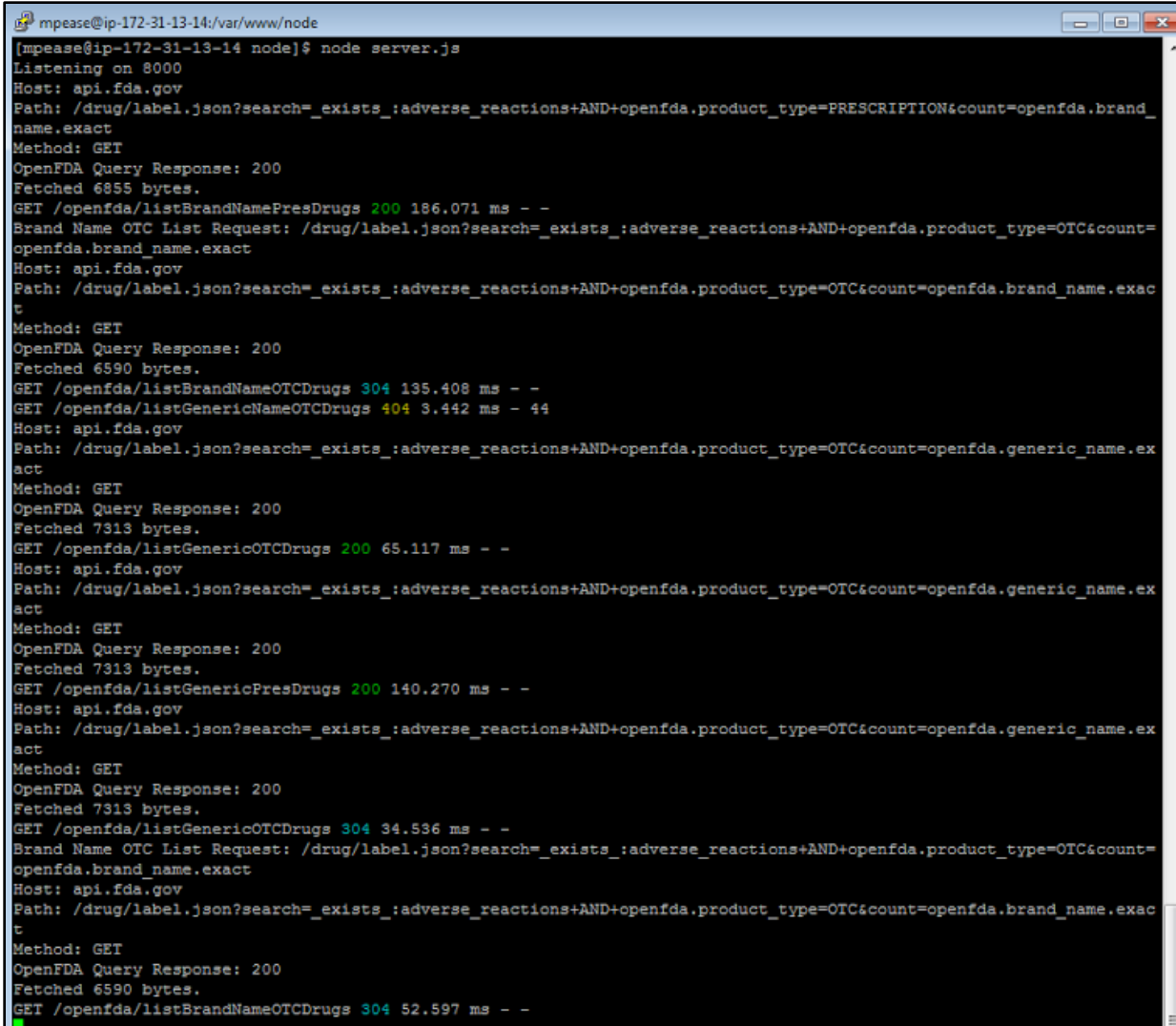
**Figure 16: AWS Performance Monitoring Features**

During development, performance monitoring for query response and page displays were analyzed as per normal functional testing and evaluation. During the test result review process and expected user interaction on the pages, the team identified that there are four queries required to display the drop down lists for users to select a drug. Those were:

- Generic Drug List of Over the Counter Medication
- Generic Drug List for Prescription Medication
- Brand Name Drug List of Over the Counter Medication
- Brand Name Drug List for Prescription Medication

During initial testing, the OpenFDA query response times ranged from 65 ms to 186 ms. Based on this information, the team concluded that these queries for multiple users and multiple times per user could be cached on the server to reduce the number of queries submitted to OpenFDA and improve the response time for page displays (Figure 17).





```
mpease@ip-172-31-13-14:/var/www/node
[mpease@ip-172-31-13-14 node]$ node server.js
Listening on 8000
Host: api.fda.gov
Path: /drug/label.json?search=_exists_:adverse_reactions+AND+openfda.product_type=PRESCRIPTION&count=openfda.brand_name.exact
Method: GET
OpenFDA Query Response: 200
Fetched 6855 bytes.
GET /openfda/listBrandNamePresDrugs 200 186.071 ms - -
Brand Name OTC List Request: /drug/label.json?search=_exists_:adverse_reactions+AND+openfda.product_type=OTC&count=openfda.brand_name.exact
Host: api.fda.gov
Path: /drug/label.json?search=_exists_:adverse_reactions+AND+openfda.product_type=OTC&count=openfda.brand_name.exact
Method: GET
OpenFDA Query Response: 200
Fetched 6590 bytes.
GET /openfda/listBrandNameOTCDrugs 304 135.408 ms - -
GET /openfda/listGenericNameOTCDrugs 404 3.442 ms - 44
Host: api.fda.gov
Path: /drug/label.json?search=_exists_:adverse_reactions+AND+openfda.product_type=OTC&count=openfda.generic_name.exact
Method: GET
OpenFDA Query Response: 200
Fetched 7313 bytes.
GET /openfda/listGenericOTCDrugs 200 65.117 ms - -
Host: api.fda.gov
Path: /drug/label.json?search=_exists_:adverse_reactions+AND+openfda.product_type=OTC&count=openfda.generic_name.exact
Method: GET
OpenFDA Query Response: 200
Fetched 7313 bytes.
GET /openfda/listGenericPresDrugs 200 140.270 ms - -
Host: api.fda.gov
Path: /drug/label.json?search=_exists_:adverse_reactions+AND+openfda.product_type=OTC&count=openfda.generic_name.exact
Method: GET
OpenFDA Query Response: 200
Fetched 7313 bytes.
GET /openfda/listGenericOTCDrugs 304 34.536 ms - -
Brand Name OTC List Request: /drug/label.json?search=_exists_:adverse_reactions+AND+openfda.product_type=OTC&count=openfda.brand_name.exact
Host: api.fda.gov
Path: /drug/label.json?search=_exists_:adverse_reactions+AND+openfda.product_type=OTC&count=openfda.brand_name.exact
Method: GET
OpenFDA Query Response: 200
Fetched 6590 bytes.
GET /openfda/listBrandNameOTCDrugs 304 52.597 ms - -
```

*Figure 17: Performance before Optimization*



Once the caching capability was added to the service, the response times dropped to 2-9ms (Figure 18).

```

Listening on 8000
OpenFDA Query Response: 200
Caching BrandName OTC List
OpenFDA Query Response: 200
Caching Generic OTC List
OpenFDA Query Response: 200
Caching BrandName Prescription List
OpenFDA Query Response: 200
Caching Generic Prescription List
Using Cached BrandName OTC Data
GET /openfda/listBrandNameOTCDrugs 200 9.360 ms - -
Using Cached BrandName Prescription Data
GET /openfda/listBrandNamePresDrugs 304 3.533 ms - -
Using Cached Generic Prescription Data
GET /openfda/listGenericPresDrugs 304 2.696 ms - -
Using Cached Generic OTC Data
GET /openfda/listGenericOTCDrugs 304 9.900 ms - -
GET /openfda/clearcache 304 1.187 ms - -
Brand Name OTC List Request: /drug/label.json?search=_exists_:adverse_reactions+AND+openfda.product_type=OTC&count=openfda.brand_name.exact
Host: api.fda.gov
Path: /drug/label.json?search=_exists_:adverse_reactions+AND+openfda.product_type=OTC&count=openfda.brand_name.exact
Method: GET
OpenFDA Query Response: 200
Fetched undefined bytes.
GET /openfda/listBrandNameOTCDrugs 304 26.032 ms - -
Host: api.fda.gov
Path: /drug/label.json?search=_exists_:adverse_reactions+AND+openfda.product_type=PRESCRIPTION&count=openfda.brand_name.exact
Method: GET
OpenFDA Query Response: 200
Fetched undefined bytes.
GET /openfda/listBrandNamePresDrugs 304 100.049 ms - -
Host: api.fda.gov
Path: /drug/label.json?search=_exists_:adverse_reactions+AND+openfda.product_type=OTC&count=openfda.generic_name.exact
Method: GET
OpenFDA Query Response: 200
Fetched undefined bytes.
GET /openfda/listGenericPresDrugs 200 52.571 ms - -
Host: api.fda.gov
Path: /drug/label.json?search=_exists_:adverse_reactions+AND+openfda.product_type=OTC&count=openfda.generic_name.exact
Method: GET
OpenFDA Query Response: 200
Fetched undefined bytes.
GET /openfda/listGenericOTCDrugs 304 47.172 ms - -
Using Cached Generic Prescription Data
GET /openfda/listGenericPresDrugs 304 2.757 ms - -
Using Cached Generic OTC Data
GET /openfda/listGenericOTCDrugs 304 0.819 ms - -

```

*Figure 18: Performance after optimization*

## 11.0 Future Enhancements

One of the future enhancements planned is to analyze the location of the traffic to determine load balancing requirements. AWS provides load balancing services at its servers across the Continental US and internal locations in Europe, Asia, etc. Once set up appropriately, the load balancer automatically re-distributes connections across multiple locations and instances. We plan to leverage the IP addresses geocoding feature to identify locations where additional AWS EC2 instances and load balancing would provide the most significant performance increase based on actual user activity.