1. **Technical Overview of the Prototype**

This section provides the technical architecture of the environment and tools used in development of the food recall alert prototype. All tools and components meet the requirements of Note #2 of section A. “NON-PRICE FACTORS” of the RFQ.

Amazon Web Services (AWS) is currently the provider of Infrastructure as a Service (IaaS) and Platform as a Service (PaaS) to IBA under a multi-year contract/agreement. This agreement allows us to consume the services on an as needed basis for all our internal and client needs with defined service levels and continuous security and performance monitoring.

* 1. **Design Considerations**

**Angular.js** was used for three primary reasons:

**Dynamic Templating:**

Rather than rendering an entire page for every URL a user visits, angular loads a template initially and dynamically swaps content in-and-out as the URL changes. The user goes to the server for data, but only redraws the content that is necessary to be changed. This allows us to provide a smoother, quicker, and more interactive experience.

**Data-Binding**

Angular's data-binding is an automatic way of updating the view whenever the model changes, as well as updating the model whenever the view changes. This allows us to dynamically update and manipulate data on the page without having to reload the entire Document Object Model (DOM).

**Clean, Modular, Reusable architecture**

Being a framework, Angular helped us to architect our code to stress modularity. This makes the code easier to maintain, extend, and test. Directives, controllers, and services allow us to build, incorporate, and test reusable components.

While there are other front-end javascript frameworks that provide similar functionality, we decided to use Angular specifically due to its maturity and its large community (Google).

**Node.Js**

Given our application is a search application using an API on top on an object database (DB), we used Node.JS for its non-blocking, event-driven I/O capabilities to remain lightweight and efficient with minimal process overhead.

**Express, npm, Bower**

We used Express for our light-weight web application framework, npm, for our server-side package management, and Bower for our client-side package management. We chose these three as they are the ubiquitous standards for Node.Js and more than suited our requirements.

**HTML5 & CSS3**

We used HTML5 and CSS3 to take advantage of modern functionality.

**jQuery**

We used jQuery for some of the dynamic content and visualization (Map, Loading animations, Usability)

**Design**

Our application's primary use case is for a consumer to find relevant food recalls. We designed the form to narrow down the 8,000 recalls to the ones that affect the user, in as few steps as possible.

**Map**

We decided that location should be the most immediate dimension to search on and devised two ways to retrieve this information from the user:

* We are able to immediately load the user's current location into the search criteria so they never have to manually enter their location.
* If the user is not comfortable sharing their current location, our map allows the user to immediately click on his/her state.

This compares to a traditional form that requires the user to manually enter their state through a dropdown menu requiring multiple clicks every time they visit a site.

**Form**

Traditional web forms are ugly and not intuitive. Users do not think in terms of query parameters, they think in terms of sentences. A typically user story for our application is someone who thinks:

“I am going to buy eggs and yogurt and want to know if there are recalls that affect my purchases.”

Our form is designed to translate this search into English. Users are able to spell out, in English, the recalls they are worried about and our application will map this to a search query.

**Recalls**

The recalls page is a list of the user's relevant food recalls. The results are sorted by date, with the most recent at the top. The search term (reason the record was selected) is highlighted to assist the user in identifying important information and we offer a secondary search for a user to 'drilldown' for even more details.

* 1. **Responsive Design**

Food Safety Alert prototype is responsive and focused for the specific location and items about which a shopper would be concerned. Shoppers are able to use their smartphones, tablet devices, or laptops.

\* target device and viewport options

--Mainly phone and tablet/desktop, utilizing the viewport metatag

<meta name="viewport" content="width=device-width, initial-scale=1.0"

\* grid system and responsive web design specifications

--Site is based on Bootstrap¹s Starter Template

--Utilizing Bootstrap¹s 12 column grid system

--Used custom CSS overrides and CSS Media Queries to assure user experience worked well on any device

--Natural Language Form for user input, which gives the user a more natural way to enter information into a cohesive sentence

\* the design and behavior of the foodsafetyalert site

\* Logo

--Logo was created with food safety in mind, user sees a familiar table setting with an alert symbol in the middle. If user on mobile device so chooses to do so, the bookmark can be saved to their mobile device desktop complete with app icon

\* Menu

--Menu allows the user to see what the site is all about and how to contact the owners

\* Navigation

--Easy workflow from choosing location to customizing search criteria to seeing results

\* Recall results

--Based on user input, the results are shown from newest to oldest with an infinite scroll, so that the user is not left waiting for a long list to load

\* zoom

--No limitation on zoom from the site. User can zoom in from the default level if need be. Used standard fonts for familiarity and font sizes to be easily read on mobile and desktop

* 1. **Creation and installation of Container**

We have developed a container that includes all the elements and components of the prototype for migration and installation of the application on a different platform in a different environment. This container was developed and used in our process successfully and is a major contributor to our successful automation of an iterative process of DevOps.

* 1. **Instances**

For this prototype we created EC2 instances on the AWS platform with the following specifications for our Test, Development, and Production environments:

* AWS EC2 t2.small instance in US-east-1 region
* 20 GB SSD
* Running Ubuntu 14.04 Long Term Support “LTS” Server
* Access Control Lists (ACL) allowing port 9922 inbound everywhere for SSH, and port 80 for site access
* Used “Agile1.pem” key for Ubuntu root user, and created accounts for developers
* Developers/Admins need to create their own local SSH key and upload to server to log-in as users

IBA used GitHub to store Development (Development branch) and Production (Master branch) code. IBA’s GitHub repository is publicly available at https://github.com/ibagit/ADS1.git.

Deployment of Production environment running “Docker” Engine:

IBA utilized https://hub.docker.com to build Docker images by pulling production code from the master branch of our repository at https://github.com/ibagit/ADS1.git. Our Docker Hub account is linked to this GitHub account.

1. Create AWS EC2 environment per the specifications above
2. Start a build on https://hub.docker.com from Master at GitHub/ads1
   1. Under Settings...Linked Accounts. Add GitHub account read/write access
   2. In ADS1 repository, start a Build. Typically takes 7-8 minutes to Finish
3. ssh into instance on port 9922 as Ubuntu user and run items 4 through 11
4. sudo apt-get update
5. sudo apt-get upgrade
6. sudo apt-get install docker.io
7. sudo ln -sf /usr/bin/docker.io /usr/local/bin/docker
8. sudo docker login (enter Docker Hub credentials and email)
9. sudo docker pull ibahub/ads1
10. sudo docker run -p <eth0 IP address of EC2 instance>:80:80 ibahub/ads1 &
11. go to public IP of instance on port 80 (e.g. http://52.5.243.57) to view Food Safety Alert application - For persistent use of the EC2 instance, we recommend allocating an Elastic IP address from AWS and assigning it to the EC2 instance
12. **Tools**

The following table provides a list of tools and components that were used in the development, testing, and production of the Food Safety Alert prototype.

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **Description of how we use it** | **Version** | **Tier** |
| Jenkins-CI | Used for continuous integration and supports our building and testing of functionality. It is also used to deploy our code across platforms. |  | Build |
| AWS  CodeDeploy | Used in our Continuous Deployment/Continuous Integration process to deploy our application to an AWS EC2 instance |  | Build |
| HTML5 | Used for structuring and presenting content |  | Front |
| CSS3 | Language that helps modify the templates coming from Bootstrap and AngularJS |  | Front |
| AngularJS | The toolset with which we consume, repackage, and display FDA’s data. Due to its extensible characteristic, it works well with the libraries in our development environment. It allows us to pass returned and repackaged data (received from the API) to the user's browser. | 1.3.16 | Front |
| Bower | One of two package managers that we utilized to pull down code packages to help fortify the application. (npm is the other). This step updates the configuration file(s) to include the new dependencies. | 1.4.1 | Front |
| JSON | JavaScript Object Notation has been used primarily to exchange data between our server, application and openFDA API |  | Data |
| Mocha | Has been used to automate and document unit testing for the site/application | 0.16.1 | Test |
| npm | One of two package managers that we utilized to pull down code packages to help fortify the application. (Bower is the other) | 1.3.10 | Build tool |
| Node.js | Build and serve up our code for the event-driven, non-blocking I/O functionality to make it lightweight and efficient well suited for our data-intensive real-time app that runs across distributed devices. | 0.10.25 | Mid |
| Bootstrap | Our responsive design, mobile first, front end framework | 3.3.5 | Front |
| Ubuntu | Our operating system of choice for hosting applications and containers, due to its robust user and support community. | 14.04 | OS |
| Docker | Used to build, deploy, and run our application in our AWS environment Virtual Machines (VM). Docker is being deployed to auto build containers based on the GIT master branch. | 1.0.1 | Container |
| Express | (Library) Works with Node.js to bundle the query request to pass to FDA’s API (Express Passes queries from browser to API and brings results to the server to be served to the user browser | 4.12.4 | Mid |
| jQuery | Used to simplify our client-side scripting | 1.11.3 | Mid |
| Selenium | Is used to automate application testing via the browser | 2.90 | UI Test |
| WAT | Used to conduct 508 testing on the application | 2012-09-13 | 508 Test |

1. **High-level Architecture**

The following diagrams provide the high-level architectural view of our prototype design. Diagram 1 was originally developed to provide a visual presentation of how we would proceed to implement our vision. Diagram2 and 3 provide views of flow of development and operations (DevOps) as we implemented our vision.Diagrams 1 provides a simplified view of the Food Safety Alert architecture

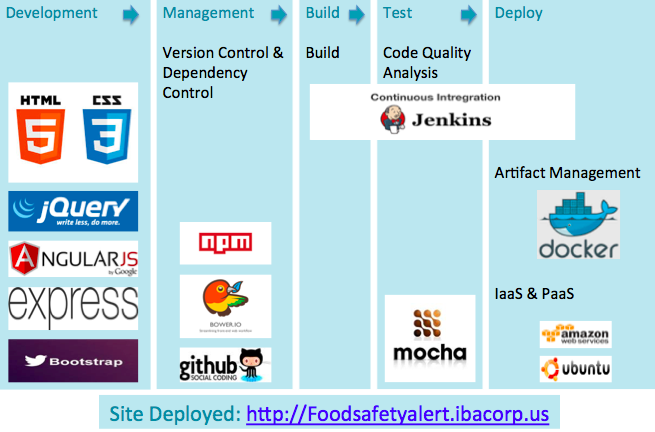


Diagram 1

Development

Food Safety Alert CI Pipeline

Jenkins/CodeDeploy

**Build/Deploy**

**Automated Test**

GitHub

Repository

Test Server

Jenkins/CodeDeploy

**Build/Deploy**

**Automated Test**

Staging Server

Docker

**Build/Deploy**

Production Server

Fail

Fail

Fail

Fail

Fail

**FQT/UAT**

Diagram 2

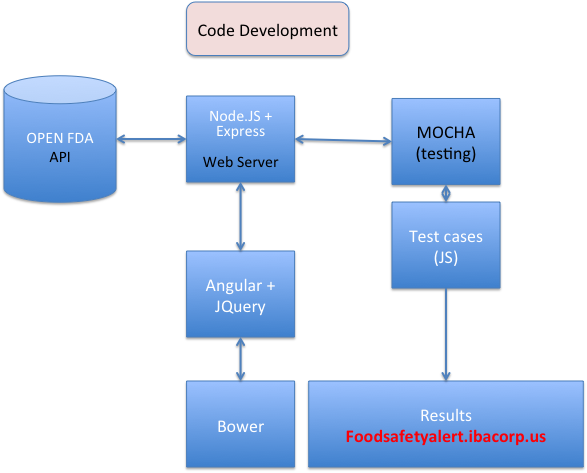


Diagram 3