BE Safe

BE Safe is a prototype application developed by Buchanan & Edwards in response to RFQ 4QTFHS150004 issued by the General Services Administration (GSA).

BE Safe is an AngularJS web application running on a Node.js/Express API and is deployable across a platforms. It accesses the OpenFDA API and allows users to search for drug recalls and adverse reactions. Users can subscribe to alerts and receive notifications of new recalls and reactions based on their search criteria. Users can share their results on social media and also submit their own individual adverse reactions.

1. Operational Prototype

The BE Safe prototype can be run on a desktop or mobile browser from the following URL:

- https://be-safe.buchanan-edwards.com

This URL is an alias to the actual deployed application running on Amazon Web Services (AWS) Elastic Beanstalk at http://be-safe.elasticbeanstalk.com. The difference between these URLs is that the primary URL has an SSL certificate that is bound to the buchanan-edwards.com domain.

1.1 Prototype Installation

The BE Safe application can be run on any platform supporting Node.js. Here is the installation procedure:

Standalone Node Installation

Prerequisites

- A Git client
- Node.js (v0.12.6 is recommended)
- A command prompt or terminal shell window

Clone our GitHub repository.

```
$ git clone https://github.com/buchanan-edwards/be-safe.git
$ cd be-safe
```

Edit the `config/default.json` file in the repository and update the four all-uppercase environment settings at the top of that file. These settings specify the OpenFDA API key as well as the AWS settings used for SMTP.

Install the dependencies and start the server:

```
$ npm install
$ npm start
```

Point your browser to http://localhost:8081/ and use the BE Safe application. Please note that the port can be changed in the `config/defaults.json` file.

Docker Installation

From a computer running Windows, Mac or Linux, perform the following steps:

1. install boot2docker if on windows or mac
2. install the lone docker package if on linux
3. docker pull buchananedwards/be-safe
4. docker run -pXXXX:3000 -pXXXY:443 buchananedwards/be-safe

2. Solution Approach

Our solution followed these principles:

1. **Team** - Leverage a multi-disciplined team that is technically innovative and has process experience ranging from agile methodologies to user acceptance testing.
2. **Architecture** - Implement a modern UI/API architecture with a Single Page Application (SPA) connected to an independently-testable Application Programming Interface (API).
3. **User-Centric** - Adopt a mobile-first approach by acknowledging the trend that mobile devices are used in ever-increasing numbers for web access.
4. **Devops** - Make continuous build, test, and deployment an integral part of the project workflow and couple the results directly into our product backlog.

We will now discuss each of these principles and reference the required points outlined in the RFQ.

2.1 The Team

Our team was led by Brian Shafer who was responsible for the overall operation of the prototype team as defined in our project charter. The team included the following GSA 18F Labor Categories.

- Product Manager
- Technical Architect
- Interaction Designer / User Researcher / Usability Tester
- Visual Designer
- Frontend Web Developer
- Backend Web Developer
- DevOps Engineer
- Security Engineer

- Delivery Manager
- Business Analyst

We wanted to ensure that we had a team that could provide technical competence and one that delivered *process* to achieve the best *product*.

The labor categories and mapping to our GSA Schedule 70 is fully documented in the evidence portion of this RFQ response.

We incorporated the following 18F guiding principles into our approach.

1. **Put the needs of users first** - multiple team members running the application during our sprints, providing immediate usability feedback and noting any issues on mobile devices.
2. **Release early, iteratively, and often** - We continuously auto-deployed from our Git development branch so any deployment issues would be known immediately. This gave our sample users a frequently-refreshed build that they could evaluate on a continuous basis.
3. **Don't slow down delivery** - Instead of developing the API first and then the user interface, we worked in parallel, taking a wholistic approach so that a working application could be delivered and tested sooner rather than later.
4. **Only do it if it adds value** - We balanced adding new features against the potential for feature bloat and possible user confusion. We focused on the value proposition, delivering core features, their reliability and efficiency over flash.
5. **Work in the open** - We leveraged only open source code and incorporated open source projects produced by Buchanan & Edwards and available on GitHub into the BE Safe application. Ultimately, we released our work in a public repository.

We interpret that last point, *work in the open*, on a more personal level. During the course of developing this prototype, none of our team members worked in cubicles. Just as deploying early and often was critical, so too was an open atmosphere of team members collaborating in a single room, sharing ideas and working towards a common goal. We employed pair-programming so that no one person was stuck and was always collaborating with another developer at their side.

2.2 The Architecture

We took a modern, best practice architecture view for this prototype. BE Safe is an AngularJS web app that is served by a Node.js server application. This server application provides the API consumed by the AngularJS web app. This approach allowed our API to be tested in isolation from the user interface and is a clean way of separating the user experience from the business logic and database access.

2.2.1 Open Source Technologies

An examination of both our `package.json` and `bower.json` files will reveal all of the technologies used in the BE Safe application. Here are the top technologies employed:

1. **Node.js** - A cross-platform server-side technology that allows us to develop, test, and deploy on Windows, Mac, and Linux systems.
2. **Express** - A web framework for Node that provides both our web application as well as our backend API.
3. **AngularJS** - A model-view-model framework for creating routable and extensible web applications.

4. **Bootstrap** - A CSS framework that provides cross-browser compatible styling and components for a robust user interface.
5. **SQLite** - An embedded database that we use for storing subscriptions as well as adverse reaction reports.
6. **Bunyan** - A logging facility for Node that is customizable for both development and production.
7. **Morgan** - An Express logging facility for tracking all HTTP requests, responses, and errors.
8. **Circle CI** - A continuous integration tool that pulls from GitHub, runs tests, and auto-deploys to AWS.
9. **Docker** - A container-as-a-service platform for packaging our application in addition to AWS deployment.

Our architecture also fulfills user's expectations and is deployable and testable in a variety of configurations. The next two sections discuss these important points.

2.3 Our Users

We took a mobile-first approach to our application. Using the Bootstrap CSS framework to build a cross-browser and cross-device application.

Our business analyst ensured that we used user-centric approaches to our development. We included people in every aspect of our efforts including envisioning, design and development. We conducted usability testing with a group of employees that were not part of this proposal effort. All responses were captured and put into our Jira backlog. Most of the feedback concerned issues where application use was clear to the developers but proved less intuitive to an uninitiated user. This was an important part of our software development life cycle.

2.4 Deployment and Operations

2.4.1 Continuous Integration

We set up automatic deployment to both Microsoft Azure and AWS Elastic Beanstalk at the very beginning of the project. This deployment approach ensured that we *could* deploy to multiple environments and that our software was platform-independent.

For Azure, we pull directly from GitHub. Azure proved to be a solid test asset as it was continuously updated on commits. For AWS, we used the Circle CI tool to build, run automated tests, build the docker container and then deploy to AWS Elastic Beanstalk on successful completion of those tests.

2.4.1 Continuous Monitoring

We used AWS CloudWatch for continuous monitoring to ensure a baseline knowledge of our application's performance and to properly manage our security risks.

2.4.2 Docker Containers

We leveraged Docker to containerize our application as it integrated nicely with our source control, CI and deployment approaches. Docker is well recognized and supported and met our needs for configuration management, continuous integration and ease of deployment and distribution.

3. Development Approach

Our Agile development approach was tailored to the size, scope and timelines of this effort. Specifically we condensed Google's Ventures Design Sprint (http://www.gv.com/sprint/) into less than 2 days and conducted daily sprints vs. 2 week sprints.

The BE team was assembled in February and conducted practice sessions to ensure we were ready to work together as an integrated team. The team came back together upon the solicitations release. We used less than 2 days to unpack, sketch and design. On the 2nd day we began daily sprints that included development, releases, demos, retrospectives and user story and backlog updates. At key points we incorporated user feedback including usability and acceptance testing.

4. Evidence Index