Voice of Consumer

Standard Operating Procedures

Table of Contents

1 Introduction 1
  1.1 Technical Implementation 1
  1.2 FISMA Compliant 1
  1.3 **OMB approval on Survey** 1

2 Development, Testing and Production Cycle 2
  2.1 Development Environment (Dev): 2
  2.2 Staging Environment (Stage): 2
  2.3 Production Environment (Prod): 2
  2.4 Deployment 2

3 Standard Operating Procedures 3
  3.1 Surveys: 3
  3.2 Flow Control & Infinite Loop 3

4 HHS Deploy Requirements Guide 4
  4.1 Software Requirements: 4
  4.2 Build Requirements: 5
  4.3 Deployment Requirements: 6
  4.4 Further Reading: 7

5 VOC Embed Instructions 7
  5.1 Embedding a survey on a webpage 7
    5.1.1 IFrame 7
    5.1.2 Javascript Widget 8

1 Introduction

Voice of the Consumer (VOC) is a pair of applications collectively capable of first generating and presenting surveys, then collecting and analyzing the results. The Administrative backend application is concerned with the administration interface, including site setup, survey creation, versioning, and results processing. The Public application is responsible for presenting surveys and collecting responses.

1.1 Technical Implementation

1) iFrame
   - HTML and CSS to style the survey
   - JavaScript
2) Web Service (AJAX)
   - JSON-P to get the survey contents
   - JavaScript (jQuery) to insert the survey content into the page
   - CSS to style the survey
   - JSON-P/JavaScript to submit the survey and show results / thank-you page
3) POP UPS
   - Java scripts that can pop up and request a survey during a web session by a user

VOC currently works on Rails and either MRI Ruby (Linux only) or the Win32 version of JRuby. Both versions rely on a database in MySQL and Mongo DB, shared between the Admin and Public applications. Redis is used as a Cache mechanism on the server to perform asynchronous task. The development, stage and production environments are all on Ruby on Rails.

1.2 FISMA Compliant

The application complies with the Federal Information Security Management Act. Compliance responsibility rests with the agency's Chief Information Office who:

- Provides information security protections
- Integrates security management processes with strategic and operational planning processes
- Assesses risk and magnitude of harm
- Implement appropriate levels of security protection for information and systems.
- Performs annual evaluation of security policies
- Any other policy or procedure considered necessary for continuity of operations

1.3 OMB approval on Survey

The framework for these standards and guidelines includes:

- Development of concepts, methods, and design
- Collection of data
- Processing and editing of data
- Production of estimates and projections
- Data analysis
- Review procedures

- Dissemination of Information Products.

2 Development, Testing and Production Cycle

2.1 Development Environment (Dev):

Development environment is a set of servers that use the same architecture as staging and production, but are updated very regularly and may not always function properly. Development is used to test new features as they are being undertaken. The data on this set of servers is erased and recreated to accommodate the current version of code that is being tested. Only CTAC has access to the development environment and will do all their internal testing here, before pushing it to the staging environment for User Acceptance Testing (UAT).

2.2 Staging Environment (Stage):

Staging is used for User Acceptance Testing (UAT). Staging servers host versions of the code that exist as release versions. Each version is tested by a User Acceptance Testing team, which may be comprised of members from HHS as well as CTAC. The UAT team determines if these new features are ready to be moved to production. These servers are generally available, but may intermittently go down as updates are performed. As code on staging is in a different state than production, any data that lives there should be considered short lived, as it may not be possible to migrate it to production servers.

Since the stage environment is where UAT testing is done, and since staging has separate code than production, if someone publishes within stage, this will only exist in stage.

2.3 Production Environment (Prod):

The production environment is where the working application resides. HHS uses this set of servers in their creation and collection of survey results. The production servers are used regularly as they are part of the reviewed and accepted code that may be deployed to conduct surveys. The data on these servers is backed up regularly and is completely recoverable even in the event of hosting problems.

2.4 Deployment

Administrative backend (VOC Admin): This application is used to create new surveys and view responses to surveys. The VOC Admin application allows for creation of survey questions and provides for some level of customization of the overall look and feel of the front-end survey questionnaire. VOC Admin application is the back end application that designs surveys and access is therefore provided to only a few authorized users as determined by HHS.

Public (VOC Public): This application shares a database with VOC Admin and serves up the actual surveys and JavaScript or iFrame used to embed / share the surveys on the public Internet. This is the front end of the survey as seen by those who respond to it on the web.

3 Standard Operating Procedures

(Additional questions you may have after reading the User Guide)

3.1 Surveys:

Surveys each have survey versions. Survey versions can be cloned and edited. Only one survey version may be published for a survey at a time. Survey versions should not be edited once published, as this would invalidate the results of the survey.

3.2 Flow Control & Infinite Loop

Survey pages and questions must flow in a logical sequence in order to finish with the intended natural end of the survey.

In the example shown in figure 1 below, Survey Page 1 (SP1) asks a Yes or No survey response to question that leads to Survey Page 2 (SP2) if the response is a yes. A 'No' response will lead the user to Survey Page 3 (SP3). While at SP3, if another 'Yes' or 'No' survey response is requested each response must lead to another page. This is where flow control needs to be used. As a 'Yes' response could lead the user to SP4 as shown in Figure 1 below while a 'No' response needs to be charted carefully so the user does not return to SP1. This could then start the survey again and the user will be in an infinite loop between SP1 and SP3.

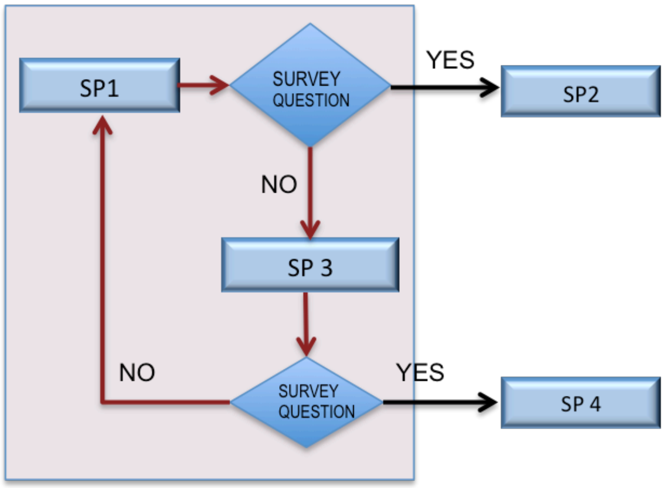


Figure 1: Shaded area demonstrates infinite loop

4 HHS Deploy Requirements Guide

HHS Voice of the Customer (VOC) Deployment Requirements Guide

4.1 Software Requirements:

4.1.1.1 Ruby Version Manager (RVM)

https://rvm.io/

RVM is the industry standard method of supporting multiple ruby environments on the same box or virtual machine while keeping each of them in complete isolation from each other. RVM enables the deployment of multiple projects, each with its own completely self-contained and dedicated ruby environment, from the specific version of ruby, all the way down to the precise set of required gems application. RVM prevents the issue of gem version conflicts between projects. RVM permits easy testing of gems and gemsets for upgrades, by switching to a new clean set of gems to test with, while leaving the original current live set intact. It is flexible enough to even maintain a set of gems per environment, or per development branch, or even per individual developer's taste!

4.1.1.2 Ruby MRI 1.9.3-p484, Bundler v.1.3.5

Ruby version 1.9.3-p484, which includes Bundler v. 1.3.5, which is a standard gem used to manage and build gemsets for Ruby projects. Bundler is installed by RVM when installing most Ruby versions.

4.1.1.3 Gem Dependencies

Rails 3.0.20 – Ruby on Rails web framework
jQuery-rails 2.1.4 – jQuery javascript library bridge for Ruby on Rails
Kaminari 0.14.1 – Pagination plugin
Authlogic 3.2.0 – Authentication plugin
Memcache-client 1.8.5 – Memcached interface for Ruby on Rails
Paperclip 3.4.0 – Document attachment management
Resque 1.24.1 – Asynchronous jobs processor
Resque_mailer 2.2.3 – Asynchronous Mailer processor
Resque-status 0.4.1 – Makes redis/resque jobs trackable
Bson_ext 1.9.0 – Provides BSON support for mongo database
Escape_utils 0.3.2 – Fast string escaping for rails
Mongoid 2.2.6– Provides ORM support for mongo database
Open_uri_redirections 0.1.1 – Handles redirection between http / https fluidly
Ranked-model 0.2.1 – Database row sorting library
Redis-objects 0.7.0 – Redis Queue Integration

Pdfkit 0.5.4 – PDF generation utility
Best_in_place 0.2.4– Javascript / support for in place editing of tables

Database Requirements

MySQL 5.5 or later should be installed with a custom application user with read/write and create table permissions.

MongoDB 2.2.0 or later should be installed with a custom application user with read/write/create permissions.

Redis 2.8 or later should be installed.

Redis is an open source, BSD licensed, advanced key-value store. It is often referred to as a data structure server since keys can contain strings, hashes, lists, sets and sorted sets. Redis runs as a deamon like MySQL or MongoDB. Full documentation can be found at http://redis.io/

Redis installation directions (http://redis.io/topics/quickstart):

```
1. wget http://download.redis.io/releases/redis-
   2.8.5.tar.gz
   other versions available here, (http://redis.io/download)
2. tar xzf redis-2.8.5.tar.gz
3. cd redis-2.8.5.tar.gz
4. make
5. run with src/redis-server
For configuration details, see http://redis.io/topics/config
```

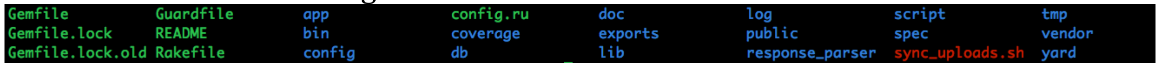
4.2 Build Requirements:

4.2.1.1 Build Server

A build server, independent of existing host infrastructure, should be used to pull updates of the deployed code from the source controlled code repository. The code repository will be the HHS Github locations established for this project. Since there is no compilation process for Ruby, the resulting code pulled from the Github is ready for deployment. This code may then be compressed and transferred to the development/staging/production environment so that those environments may be kept secure from outside connections.

Private Githubs:
https://github.com/HHS/voc-public.git
https://github.com/HHS/voc-admin.git

The file structure on the Application Server in each environment should look similar to the structure in the image below.



```
Gemfile          Guardfile  app     config.ru  doc      log              script           tmp
Gemfile.lock     README     bin     coverage   exports  public           spec             vendor
Gemfile.lock.old Rakefile   config  db         lib      response_parser  sync_uploads.sh  yard
```

The public folder contains all of the html and html support static files, which should be pushed to the front facing web server/CDN.

The build server can most easily constructed on top of any of the modern flavors of Linux.

4.3 Deployment Requirements:

4.3.1.1 Configuration

RVM should have a configuration file for each project directory generated by the command line:

```
rvm use ruby-1.9.3-p484 --rvmrc
```

Validation can be done by switching to the project directory in the terminal and reading the ruby version report string output upon the directory switch.

Once the Github code tree is deployed to Application Server, run the bundle install from the root of the project directory to install the required gems via the command line:

```
bundle install –local
```

In the VOC/config directory the files database.yml, mongoid.yml, secret_settings.yml, and app_config.yml should be modified in the following ways. (The Github repository will have *.example files for reference and syntax requirements.)

a. database.yml – Change the User/Database connection information
b. mongoid.yml – Change the user/database connection information
c. secret_settings.yml – Uncomment (delete the hash symbol) and generate a new secret on the command line. This is used to hash cookies for users.
d. app_config.yml – Set the host url to target of the ajax calls for each distinct environment dev/staging/production. Set the redis server location and port; along with expire value (86400 recommended). Branding information can be applied here, defining a custom header/footer file as well as a custom CSS file.

Configure the Web servers. VOC consists of two distinct applications, each of which should have its own virtual host section. These virtual host sections will segment these applications away from existing and future Ruby applications running on the same server. As VOC will be hosted using Phusion Passenger, indicate the rvm and ruby info for Phusion Passenger in the same way as other Ruby on Rails applications within the virtual host section.

Example VirtualHost Config Assuming Passenger 4.0.0 or greater:

```
<VirtualHost *:80>
```

```
 ServerName voc.example.gov
 DocumentRoot "/webapps/voc-app/public"
 RailsEnv production
 PassengerRuby /Users/rubyapp/.rvm/wrappers/ruby-1.9.3-p448@comment/ruby
</VirtualHost>
```

If multiple app servers are needed, they can be configured independently and placed behind a load balancing solution.

Run the initialization rake tasks in a single command line:

```
rake db:create db:migrate db:seed
```

Assuming the settings in database.yml are correct rake will create the database, load the schema, and populate with the required seed data.

Run the initialization rake tasks in a single command line with

RAILS_ENV=production, rake db:create db:migrate db:seed

Assuming the settings in database.yml are correct rake will create the database, load the schema, and populate with the required seed data.

Touch the /tmp/restart.txt file to signal to Phusion Passenger to restart.

4.4 Further Reading:

The VOC User Guide will tell them how to create a survey. All surveys will have a survey ID associated with them.

5 VOC Embed Instructions

5.1 Embedding a survey on a webpage

There are two mechanisms by which a survey can be embedded into a web page: via an IFrame or via a javascript widget.

5.1.1 IFrame

To embed a survey onto a web page using an IFrame, add the following HTML markup:

```
<iframe src=http://[Public App URL]/surveys/[Survey ID]></iframe>
```

- Replace [Public App URL] with the URL of the public application.
- Replace [Survey ID] with the numeric ID of the survey to display.

A working HTML snippet with the [Public App URL] and the [Survey ID] correctly specified can be found in the survey tool admin interface at the bottom of the survey versions page.

5.1.2 Javascript Widget

There are two JavaScript widgets available to add a survey to a content page. The embeddable widget will load the survey directly into the HTML. This will load the survey into the page as a visible element of the page content.

The invitation widget will load the survey through an invitation style pop-up window. This will show an invitation pop-up to the user on page load asking them whether they wish to take the survey. If the user accepts the invitation a new window is opened with the survey content.

5.1.2.1 Embeddable Widget

To add an embeddable survey widget onto a web page, add the following HTML markup:

1. Add an empty DIV element to the page to act as a placeholder for the survey content.

   ```
   <div id="survey_target"></div>
   ```

2. Immediately after the placeholder DIV created in the previous step, add the following HTML markup in order to fallback to the IFrame version for users without JavaScript.

   ```
   <noscript>
     <iframe src=http://[Public App URL]/surveys/[SurveyID]>
     </iframe>
   </noscript>
   ```

3. Add the following HTML markup to the bottom of your content page.

   ```
   <script type="text/javascript" src="http://[Public App
   URL]/widget/widget.js?survey_id=[Survey
   ID]&target_id=[Placeholder DIV ID]"></script>
   ```

   * Replace [Public App URL] with the URL of the public application.
   * Replace [Survey ID] with the numeric ID of the survey to display.
   * Replace [Placeholder DIV ID] with the ID of the placeholder DIV created in step 1.

5.1.2.2 Invitation Pop-up Widget

To add an invitation survey widget onto a web page using the JavaScript widget, add the following HTML markup:

1. Add an empty DIV element to the page to act as a placeholder for the survey content.

   ```
   <div id="survey_target"></div>
   ```

2. Immediately after the placeholder DIV created in the previous step, add the following HTML markup in order to fallback to the IFrame version for users without JavaScript.

   ```
   <noscript>
     <iframe src="http://[Public App URL]/surveys/[SurveyID]">
     </iframe>
   </noscript>
   ```

3. Add the following HTML markup to the bottom of your content page.

   ```
   <script type="text/javascript" src="http://[Public App
   URL]/widget/invitation.js?survey_id=[Survey
   ID]&target_id=[Placeholder DIV ID]"></script>
   ```

   * Replace [Public App URL] with the URL of the public application.
   * Replace [Survey ID] with the numeric ID of the survey to display.
   * Replace [Placeholder DIV ID] with the ID of the placeholder DIV created in step 1.