failsafe Technical Guide
Version 1.0
December 10, 2014

Contents

Databases

Failsafe uses the MySQL database management system. The individual database files can be found in the *sql_files* directory of the code. Data currently are stored in two main databases with a total of three tables.

The first database is *directory.* In *directory*, there exists the *tblUser* table, which contains contact data on each user. The second database is *calendar*, which contains *schedule* and *substitutions* tables. The *schedule* table contains the monthly assignments of users to days whereas the *substitutions* table contains the hourly shifts that are traded from the original users.

For descriptions of each table's schema in the two databases, see doc/DatabaseSchemas/[TABLENAME].txt

Common Layout

Content that is common across all pages can be found under `templates/layout.html`. This html file contains information for both the actual navigation buttons (dashboard, directory, calendar, substitutions) and the activation buttons, which are responsible for activating & deactivating teams.

Navigation Bar

The navigation bar is essentially boilerplate code from Twitter Bootstrap. Twitter Bootstrap is well-documented, and so changes to our navigation bar can be easily made by following their documentation relative to the HTML at `templates/layout.html`.

Activation Buttons

The Activation buttons are within the navigation bar, but have their own complex functionality and are crucial to the use of FailSafe. These buttons are controlled by the JavaScript at static/navbar.js, and are as follows:

Activate

The activate button sends an alert to all members of the active Call Team every 30 seconds until they respond to the Twilio server. The JavaScript code for initiating this process is located in the `alertOnCall()` method in `navbar.js`. This code initiates the call using a JavaScript interval, and thus re-pings the endpoint backend/contact every 30 seconds, which actually makes the calls through Twilio. To change this interval, simple edit the `alertFrequency` field at the top of the file to the desired number of milliseconds.

Deactivate

The deactivate button sends a single simple message to all members of a previously activated call team to inform them that the alert is *no longer* in effect, and that they should go home. This behavior is controlled by the `deactivate()` method in

navbar.js, which calls the *backend/form_team* and *backend/deactivate* endpoints. The first consolidates members of the currently active team from the substitutions and schedule tables, and the second actually sends the messages. This button, similar to the Silence button, also breaks any alerting loops started by clicking the activate button.

Silence

The silence button simply breaks any alerting loops that have been started by clicking the activate button. All this has to do is clear every interval in the *alertingIDs* map in navbar.js. This behavior is done through the *cancelAll()* method (which is also called by the deactivate button).

Code Base Analysis

Backend Python/Flask Overview

The Views are organized under into various blueprint.py files. Each blueprint.py file maps different URLs to different HTML Jjina2 templates, which are located in the *templates* folder of each view's folder located in the *src* folder. In order to learn more about blueprints, you should go to: http://flask.pocoo.org/docs/0.10/blueprints/.

Resources Overview

All JavaScript, CSS and other static resources for views exist in the *static* folders of each view folder. Javascripts contain jQuery. All dialogs are created using the *createDialog()* function.

Moment.js

All of the storage of dates and times relies on the moment.js library, which allows one to easily parse and manipulate dates using Moment objects in JavaScript.

Month View vs. Day View

The month view is represented by the HTML file under *calendar_view/templates/month_view.html*. This includes the dialogs that pop up when one tries to create or edit call events. Similarly, the day view is represented by the HTML file at *calendar_view/templates/day_view.html*. The calendar itself (within the month view) is updated using the *makeCalendar()* function in *calendar_view/static/eventScript.js*. The substitutions table (within the day view) is updated using the *makeDayView()* function in the same file.

AJAX

Data is communicated back and forth to the Python endpoints using AJAX. The primary functions that we use in the calendar view to communicate this information are as follows:

Simple get methods: *AJAXGetWrapper(endpoint)*

Get methods with data: *AJAXGetWithData(endpoint, requestParams)*

All other HTTP methods: *AJAXJSONWrapper(method, url, data)*

Twilio/Paging Code

Messaging, calling and paging code are located in the *backend/fs_twilio* folder. Texting and calling functionality are made possible through the Twilio API. In *fs_twilio*, the file also contains paging functionality which explicitly places URL calls to the paging website. In order to get more information about how Twilio API works with Flask, you should refer to the following link: http://www.twilio.com/docs/quickstart/python/sms/hello-monkey. This will allow you to get up to speed on how basic functionality is done in Twilio. Additionally, you should

also examine your settings for your phone if you want to change the way responses occur by editing the URL that is requested whenever a message or call is made to the server through Twilio.

Blueprint Hierarchy for Backend

The backend follows the same structure as the front end in terms of the “blueprint” formatting related to backend. That is, we have python code in flask to directly map functions to URLs under the framework. We have functions for the following things:

- Processing text message responses from members of the on-call team after they have received notifications to come to the hospital for an emergency.
- Logging the user into the system
- Looping through the contact information of all members of the on-call team
- Identifying which users have responded and ensuring that they do not remain included as part of the activation looping process
- Deactivating the activation loop for the on-call team: this will prevent users from

In addition to these main functions, we also have a few helper functions that allow us to retrieve information from the front end and also keep our code a bit more flexible.