Mango Punchbowl Documentation

Intent:

The intent of this application was threefold.

1. Establish a unified posting system to post messages to Facebook and Twitter that can be used for any desired function or integration in the future, keyed on either a supplied email or UUID. Handle initial authorization to both platforms as well as retention of tokens for posting on customers behalf in the background.
2. Build three front end interfaces to the posting flow.
    1. Gift of Mango - A simple form interface that posts a customer created message to Facebook and/or Twitter.
    2. Mango Tango - Collect an email and name from a customer and up to 5 emails and names of friends in a reward program. Once collected, offer the user the ability to post to Facebook and/or Twitter
    3. Mango Client integration - Give the client application a tool that can be loaded into their system that will handle posting to the posting flow for their customers based on UUID
3. Create a automated reward and fulfillment script that can select winners from a list of potential Mango Tango customers. Then email the winners with a link to a form to collect address information. And finally email fulfillment a list of all addresses from this weeks winners

Execution Point 1 - Facebook/Twitter:

Facebook and Twitter integration is a very difficult dance to get right. The Punchbowl application uses a number of tricks to make this as seamless as possible will remaining within the confines of the rules from both platforms for authorizations.

API keys:
Facebook and Twitter currently have temporary API keys assigned to our accounts and registered with the URLs of our test system at punchbowl.heroku.com **New keys will need to be registered under**

Mango accounts for Twitter and Facebook once a permanent location for the application is established

Integration Method:

Facebook - We used Facebook's new OpenGraph API with Oauth2 support when connecting from Punchbowl. Initial authorizations with Facebook are made via an http redirect to a special url. There the user will be presented two paths

1. If not logged into Facebook, a login request is made. Once logged in, move to #2
2. A screen asking to authorize mango's API account to access the users account offline and post stream

Once authorized, a session token is returned to punchbowl. PB will reconnect with FB to exchange that session token for a permanent authorization token that is stored with the customer record in the FacebookAccount model.

A customer record with a FacebookAccount that contains a token is considered Green Lit to post directly to facebook without reauthroization.

Twitter - We use Twitters Oauth1 support when connecting with Punchbowl. The steps are very similar to Facebook's when authorizing an API account to access the twitter user. Twitter uses a token and secret combination to exchange session information for a authorization token, but the idea is the same.

Once authorized, a TwitterAccount record is tied to the Customer record.

Error Handling - Both Facebook and Twitter return JSON for messaging. We have taken the approach of building an error handler into two models, TwitterApi and FacebookApi, which can recognize some of the more common errors returned by these platforms. Much of the time building the posting flow was in trapping and handling error events.

When an error happens, it is logged to the Customer's record, logged to the Rails log, and (optionally) can be emailed to a system account

One error to look out for is 'Duplicate Message' . Neither Facebook nor Twitter allow the same message to be posted back to back, so this error is an indication of someone using the back button or scripting posts to someones account

Modularize Flow:
In order to meet the requirements of the second execution point, we needed to build a modular approach to posting to social media that could be used in many different controllers. A mixin called Postable was constructed to allow both html and json requests and responses to any controller result in a post to social media, along with the handling of redirects and messaging.

Execution Point #2 - Points of Entry

Gift of Mango:
http://punchbowl.heroku.com/gift_of_mango
As a simple form collection, GoM was a good starting integration point to the posting flow. GoM collects a message, and email address, a twitter and a facebook option and posts them via html to the GiftOfMangos#create action. Validations client side require and format the email and message fields

Here a GiftOfMango record is created, associated with a Post and a Customer. The Customer is found or created based on email address provided.

This Create action has the Postable mixin which handles posting to the social media

Mango Tango:
http://punchbowl.heroku.com/mango_tango

A more complex form collection. MT collects an email and name of the customer, and up to 5 emails and names of friends. Upon clicking deliver, two actions happen in sequence through JavaScript.

1. A serialized dataset is POST via json to MangoTangos#create. This action creates or finds the customer based on email, and creates a dance partner for each friend. A callback on the DancePartner model for after_save emails the friend. **This is currently done in serial procession, slowing down the response to the customer. It is HIGHLY recommended that this process be upgraded to use DelayedJob, which is added but not configured. Also, these emails reference a discount CODE that has not been added.**
2. An iframe is opened to Posts#new that mirrors the data collection from GoM. This integration approach allows for more fluid interaction with PB from any web page. If the customer's record, found by email address, is known and greenlit a post can be made in the iframe. If authorization needs to occur, Facebook or Twitter will put a link in the iframe to direct the user directly to the authorization page. This is an unavoidable consequence of using these platforms.

NOTE: There was a mid shift in design that affects the Mango Tango flow. Only Accomplishments (Mango Client) were originally written to support badge images to Facebook. It was mentioned later that Gift of Mango and Mango Tango both would like to have images to facebook (badges). The Gift of Mango uses the GiftOfMangoController#create action to post through postable, so extending this to use badges was easy. However, due to the flow of Mango Tango, two calls are made from the browser on submit, one to the MangoTangoController to create the MT object and a separate async call to pull up the posting form in an iframe and post to Post#create. Therefore, a Mango Tango object is not reliably created before a post is sent to Post#create and so the association cannot happen in time, and a badge can not be associated with the Facebook post. Redesign would be to flow everything through the MangoTangoController to make the proper associations and to use delayed job to kick off the emails, but this was not in scope for this deliverable.

Mango Client:
http://punchbowl.heroku.com/test_badges
The most complex integration task for this project was building a tool that could be used by the Mango Client. To this end we have constructed a jQuery plugin called Punchbowl that can be included as a javascript file. The plugin has all the needed code to query PB directly for information and send customers in different directions depending on the results of the query.

The example page uses a test and debug mode to show the various states of integration with PB from the client. Currently the initialization settings are

1. url - currently defaulted to 'http://punchbowl.heroku.com'
2. debug - defaulted to false but turned on on heroku
3. testMode - defaulted to false but turned on on heroku

Once initialized, the client will pass the following information to the plugin on call

1. UUID - '1'
2. Message - 'I just completed lesson 2 of english'
3. badge_name - 'English'
4. language - 'English'
5. lesson_number - '2'

Naming is important with badge_names and language, as they are case sensitive and are capital in the demo

If a UUID is null, the customer is considered Anonymous and is shown the overlay without remember_me

If the UUID is not null, given the settings and the data, a call is made to PB using GET json to lookup the customer based on UUID. Response is evaluated

1. Customer is not known - show overlay
2. Customer is known but does not want to share - do nothing
3. Customer is known, wants to share. but wants to be asked - show overlay without remember_me
4. Customer is known, wants to share, doesn't want to be asked but is not green lit - show overlay

5. Customer is known, wants to share, doesnt want to be asked and is green lit - POST json to Accomplishments#create - **NOTE: Since this post is constructed by the jQuery plugin it lacks the authorization token created by Rails for forgery protection. As such forgery protection has been turned off at this time.**

Overlay:
The plugin has the HTML code needed to construct an overlay located within the div acted on by the plugin. This overlay is based on the html given to us by Mango, and requires the jquery.punchbowl.css file for styling.

When the overlay is filled in and the post button is hit, a jQuery event serializes the data and sends an does an html submit to a target = _blank to Posts#new to begin the authorization process. **NOTE: This window is not being closed by the application at this time on success and needs to be closed manually. A close event on success should be added if desired.**

When the customer click on the 'No Thanks' button, and if the customer is not anonymous, a json PUT to /customers/uuid/:uuid is made to update the customer with wants_to_share = false

NOTE: Currently the testing of this on the demo site is using a uuid of 1, associated with one of our customer accounts.

Execution Point #3 - Mango Tango Script

A rewards engine was requested as the third execution point for Punchbowl. Every Mango Tango created within the previous week would be eligible to be rewarded by Mango with swag.

The first four weeks of the reward program should reward every Mango Tango that has at least one Dance Partner, a customer email, and has not already won.

Following on the 5th week, the Top 5 most referring Mango Tango customers (having the largest number of Dance Partners) who have a customer email. were created this week and have not won before are winners. Additionally 20 randomly selected Mango Tango customers who have not won before, are not part of the top 5, were created this week, have a customer email and at least one Dance Partner also win.

Winners are emailed a congratulations email with a link back to PB **NOTE: The copy for the congratulations email is currently TEMP copy** The link will contain a token that is used to lookup the reward on loading the page. Filling out the address information will 'redeem' the reward.

Every week a list of all the redeemed winners will be sent to an email address for fulfillment.

Rake Task:

Handling this process is a Rake Task called pb:rewards. The following tasks are used

1. pb:rewards:tango - All tasks for the week
2. pb:rewards:send_tango - select winners and send emails for rewards
3. pb:rewards:fulfill_tango - pull redeemed winners and send to fulfillment
4. pb:rewards:increment_week - up the week number

Tango Week Number:
The process of selecting possible winners depends on what week of the process we are in. Date calculation is too dependent on executions being on time. So instead we implemented a single row in a model called AppStat that holds this number. It starts with week 0 on launch. This week is offset from a launch date, set within the config.yml file. **NOTE: this needs to be updated on launch**

Calculations:
The many permutations of the tango rewards system resulted in very complex calculations that are embeded in the MangoTango controller. **As such, it is highly advised that these calculations be run manually prior**

to running the tasks and the results be verified, at least until comfort with the calculations accuracy is established

Addendum

Parts of the application that were not specifically scoped for this iteration but were added

Admin interfaces for Badges and Languages:
Maintenance of the Badges and Languages demanded that at least a simple admin interface be constructed. /admin/badges and /admin/languages are the locations. Each is authorization protected by Devise though a proper login/login action has not been constructed, it is advisable that this function be built out more or disabled with an established admin user already in place on the system.

Final location of system:
Currently this application is being demoed on a Heroku instance. Installation at the final location is not part of this iteration, but should follow standard Rails 3 system requirements. Delayed Job will need to be setup if desired. The config.yml file should have all the AppConfig settings. New API keys for Twitter and Facebook will need to be created to a mango account. Spec tests are early TDD development files and are there as a starting point for full test coverage, but do not pass currently.

System Information:
Rails 3
Formtastic
Haml
Sass
Twitter
Oauth2 (Facebook)
json
Delayed Job
Devise

Typhoeus (JSON parser)
curb (curl wrapper)