Bookie Documentation

Release 0.5.0

Rick Harding

May 02, 2015

Contents

Bookie is a young open source project to help replace Delicious.

CHAPTER 1

Latest Release

Stable v0.5

Development v0.6

CHAPTER 2

Bookie Features

- Open source!
- Imports from Delicious.com, Google Bookmarks, Google Chrome, and Firefox.
- Google Chrome extension
- Firefox extension
- Bookmarklet for other browsers (mobile devices)
- Store page content and fulltext searches it
- Support for Sqlite, MySQL, and Postgresql
- Mobile friendly responsive layout
- Android app

See the features page for full length details.

CHAPTER 3

Contributing to Bookie

Learn more about contributing to Bookie.

CHAPTER 4

Important Links

Github http://github.com/bookieio/Bookie

Mailing List https://groups.google.com/forum/?hl=en#!forum/bookie_bookmarks

See it live https://bmark.us

ToDo List https://trello.com/board/bookie/4f18c1ac96c79ec27105f228

CHAPTER 5

Bookie Tools

Bookie App http://github.com/bookieio/Bookie

Readable App https://github.com/bookieio/bookie_parser

Readable Lib https://github.com/bookieio/breadability

Bookie Cli https://github.com/bookieio/bookie_api

Bookie Android https://github.com/bookieio/Bookie-Android

Contents:

6.1 Installing Bookie

6.1.1 The short version

This assumes you're on Ubuntu or can figure out the difference between Ubuntu and your distro for the following:

```
$ git clone git://github.com/bookieio/Bookie.git
$ cd Bookie && make sysdeps && make install
# THIS WILL TAKE A WHILE, GET A COFFEE
$ make run
$ (YOUR BROWSER) http://127.0.0.1:6543/
```

You should have JavaScipt enabled.

Login as admin

```
user: admin
pass: admin
```

You can create new users by writing a script or going through the signup process. You just need to install inbox (bin/pip install inbox) and run make smtp to get a fake email server to catch the activation emails.

Where to go from here

Getting your bookmarks into Bookie

Well, you might want to import a backup of your delicious bookmarks. You can do that by visiting the *Import* link in the footer of your site installation. Make sure you know the API key that you've set in your bookie install's *.ini* configuration file.

You can view your recent bookmarks at: http://127.0.0.1:6543/recent

Installing Extension

You probably also want to install a browser extension to be able to store new bookmarks going forward. Once you install the extension, you'll need to set the options for it to work. See the browser extension docs for those settings.

Hosting Bookie

You can setup Bookie to run in a variety of ways. Make sure to check out some samples in the hosting docs

6.1.2 More details than you can shake a stick at

OS Packages

There are some required packages that need to be installed so you can build bookie. These are:

- build-essential
- libxslt1-dev
- libxml2-dev
- python-dev
- libpq-dev
- git
- python-virtualenv
- redis-server
- unzip

Note: right we we support three databases - mysql, postgres, and sqlite - and the database bindings need to be built into the virtualenv. Out of the box, Bookie will setup a Sqlite version for you to get started with.

Celery backend task processing

Bookie uses the Celery library to handle background processing tasks that might be expensive. Currently, it's setup to use redis as the backend for this. Please check the *bookie.ini* for the connection string to the redis database. Any time a bookmark is saved it will background fulltext indexing for the bookmark. During import, it will attempt to fetch content for the imported urls as well. Emails and stats generation also go through this system. By default, *make run* will start up celery in the background. An exmaple manual command to run celery safely with the sqlite default database is:

```
celery worker --app=bookie.bcelery -B -l info  --purge -c 1
```

Adjust the command to your own needs. You might need to increase or lower the debug level, for instance, to suit your needs.

MySQL & Postgresql Users

If you're using Postgres or MySQL as your database for Bookie you'll also want to grab the dev package for your db so that the Python drivers for them can compile.

- libmysqlclient-dev (Mysql)
- postgresql-server-dev-8.4 (Postgres)

```
$ sudo apt-get install libmysqlclient-dev
- OR -
$ sudo apt-get install postgresql-server-dev-8.4
```

You will also need to install python db drivers for MySql.

- MySQL-python

```
$ bin/pip install MySQL-python
```

Then you'll need to update the database connection string in your *bookie.ini* file. The database and user account need to exist in order for it to bootstrap the database for you. Once you're ready run:

```
$ make db_up
```

Migrate from SQLite to MySQL or Postgresql

First, follow the steps above to set up an empty MySQL/Postgresql database.

To prepare for the migration, we first need to empty the *alembic_version* and *users* tables which are not fully empty. To do this in MySQL:

```
TRUNCATE `alembic_version`;
DELETE FROM `users` WHERE 1;
```

Then, install the migration tools:

```
$ apt-get install ruby-sqlite3 ruby-mysql
- OR -
$ apt-get install ruby-sqlite3 ruby-pg

$ gem install rack -v 1.4.5
$ gem install taps
```

Next, let's publish our existing sqlite database:

```
$ taps server sqlite:///home/user/bookie/bookie/bookie.db tmpuser tmppass &
```

And finally we pull the data into our new MySQL/Postgresql database:

```
$ taps pull -s mysql://bookie:***@localhost/bookie http://tmpuser:tmppass@localhost:5000
- OR -
$ taps pull -s postgres://bookie:***@localhost/bookie http://tmpuser:tmppass@localhost:5000
```

6.2 User Docs

6.2.1 Importing

Bookie currently supports the Delicious export, Google Bookmarks export, Google Chrome bookmarks, and Firefox JSON backups.

- Delicious - You can export from Delicious by going to *Settings* and clicking the *Import / Upload Bookmarks* link under the *Bookmarks* section.
- Google Bookmarks - You can export your Google bookmarks by going to ???
- Google Chrome - You can export by going to *Bookmark manager*, *Organize*, and then *Export bookmarks to HTML file...*.
- Firefox - You can generate a backup by going to *Show all Bookmarks*, then *Import and Backup*, and selecting *Backup...*.

The importer might take a bit on large sets of bookmarks. Let us know if you run into any issues so we can improve our import process.

6.2.2 Readable Parsing of your Bookmarks

One of Bookie's best features is that it will fetch the content of your bookmarks and attempt to parse/fulltext index it. A bookmark import will cause the system to go in and start fetching content for the new bookmarks. There's also a background task that will (by default) attempt to find any bookmarks missing content and fetch it on an hourly basis.

Example cron jobs

```
# run readable parsing on new bookmarks each morning at 1am
0 1 * * * /path/to/bookie/env/bin/python /path/to/Bookie/scripts/readability/existing.py --ini=myconf

# retry error'd parsing at 1am on the 1st of each month
0 1 1 * * /path/to/bookie/env/bin/python /path/to/Bookie/scripts/readability/existing.py --ini=myconf
```

6.2.3 Backup your Bookie bookmarks

There's a quick/dirty sample script you can use to backup your bookmarks. It just calls the */export* url on your installation and creates a *.gz* backup file.

This obviously doesn't store things like the fulltext indexes and such. So if you are using the Readable versions you might want to keep a backup of your database itself instead of just dumping your export file.

A sample of cron'ing this to run at 6am every day would be:

```
0 6 * * * /usr/bin/python /path/to/Bookie/scripts/misc/backup.py
```

6.2.4 Bookmarklet

To use the bookmarklet, log into your account page and drag the link at the bottom to your bookmark bar. In the Android browser, you can long-press on the link and bookmark it.

After that, you can bookmark any page you're currently viewing by clicking on the bookmark in your browser. It will load the current url and page title into an add form on the website.

Once you stored the bookmark with tags and description, you'll be redirected back to the page you were originally viewing.

6.3 Browser Extensions

If you do not use Google Chrome, make sure to check out using the Bookie bookmarklet.

6.3.1 Google Chrome Extension

Provides Bookie bookmarks into Google Chrome

Features

- supports loading existing bookmark data if you're on a page already bookmarked

- Capable of sending current page's html content to Bookie for parsing the readable version of the page so it's immediately available. This slows down the plugin, so uncheck the *Cache Content* if you experience adverse performance
- Assists by providing recently used tags for completion

Installation

- Chrome users can get it from the Gallery: http://goo.gl/NYinc *Hint* good reviews would be appreciated!
- To get the development version of the extension (to use with the develop branch) use the url: http://files.bmark.us/chrome_ext.crx
- Updates currently need to be done manually

Setting up

In order to setup the extension you'll need to set a couple of options. To pull up the options page right-click on the extension in the tool bar and select the *Options* menu.

API Url set this to the installed url for your bookie instance. In dev mode it's *http://127.0.0.1:6543/api/v1*. If you do not set the api url you should get an error about not being able to find a bookie instance at that url.

By default the extension attempts to hook you up to the bmark.us instance.

API Key You can get the API key for your user by logging into the website and going to the *Account* page. There you will find a link for "View API Key" that will show you your key. The default login for a fresh install is a admin:admin.

Username This is the username for your account. This is used to help construct the api urls. The default user account is *admin*.

Cache Content If you check this, then the html of the page is sent to the Bookie installation when you click the button to save a bookmark. This might pass large bits of data over and slow things down a little bit. If you find it too slow, uncheck and run the server side script provided via cron to get the *readable* version of your bookmark content.

6.3.2 Firefox Extension

The Firefox extension is starting over from scratch. You can track it at:

- https://github.com/bookieio/bookie-firefox

6.4 Hosting your Bookie installation

Right now, we're in full developer mode so hosting is up to you. We help you get started by running it with the built-in *paste* webserver locally.

However Bookie is a WSGI application and can be hosted with web servers such as Apache, Nginx, and Cherokee. As Bookie matures we'll try to get cookbook docs for using each. For now, it's up to you to figure out, but feel free to drop by the *#bookie#* irc channel on Freenode for assistance..

6.4.1 Nginx proxy to Paster

To start out, the easiest thing to do is to put a web server in front the paster development server. So you run the web server with the command:

```
paster serve  --daemon bookie.ini
```

Then to serve it behind Nginx you need to setup a new virtual host config.

```
server {

    listen   80;
    server_name  bookie;

    #set your default location
    location / {
        proxy_pass              http://127.0.0.1:6543/;
        proxy_set_header        Host $host;
        proxy_set_header        X-Real-IP $remote_addr;
        proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header        X-Forwarded-Proto $scheme;
    }
}
```

Now you can create an alias in your */etc/hosts*

```
127.0.0.1 bookie
```

now *http://bookie* should be served through Nginx to your bookie instance.

6.4.2 Hosting with Nginx + uWSGI

uWSGI is a great way to run WSGI apps. Nginx is then setup to be the front end and communicate with the uWSGI processes running.

Warning, this isn't the easiest way to set things up, but it's pretty fast and decent to run.

wsgi.py

First, you need a *wsgi.py* file that tells uWSGI where your environemnt and application are to run.

Please this file into your application's bookie app directory. If you've used the normal bootstrap process it should be in:

```
bookie/bookie/bookie/wsgi.py
bookie/bookie/bookie/combo.py
```

With the following:

wsgi.py

```
#!/usr/bin/python env
import os
from os.path import dirname

# Add the virtual Python environment site-packages directory to the path
import site
```

```
ve_dir = dirname(dirname(dirname(dirname(__file__))))
install_dir = dirname(dirname(__file__))

site.addsitedir(os.path.join(ve_dir, 'lib/python2.6/site-packages'))

# Avoid ``[Errno 13] Permission denied: '/var/www/.python-eggs'`` messages
os.environ['PYTHON_EGG_CACHE'] = os.path.join(install_dir, 'egg-cache')

# Load the application
from paste.deploy import loadapp
application = loadapp('config:' + os.path.join(install_dir, 'production.ini'))
```

combo.py

```
"""WSGI file to serve the combo JS out of convoy"""
from convoy.combo import combo_app
JS_FILES = 'bookie/static/js/build'
application = combo_app(JS_FILES)
```

uWSGI Config

Now we need to add the uwsgi daemon settings for this application. We'll create a file */etc/init/bookie.conf* that will give us an upstart enabled service to run the app through.

```
description "uWSGI Bookie Install"
start on runlevel [2345]
stop on runlevel [!2345]
respawn
exec /usr/bin/uwsgi26 --socket /tmp/bookie.sock \
-H /home/$username/bookie/ \
--chmod-socket --module wsgi \
--pythonpath /home/$username/bookie/bookie/bookie \
-p 4
```

combo loader

```
description "uWSGI Convoy"
start on runlevel [2345]
stop on runlevel [!2345]
respawn
exec /usr/bin/uwsgi --socket /tmp/convoy.sock \
-H /home/$username/bookie \
--chmod-socket --module combo \
-p 4 --threads 2
```

We should not be able to start up the server with uWSGI command there.

```
sudo /usr/bin/uwsgi26 --socket /tmp/rick.bmark.sock \
-H /home/$username/bookie/ \
--chmod-socket --module wsgi \
--pythonpath /home/$username/bookie/bookie/Bookie/bookie \
-p 4
```

This will help bring up any potential errors. If all starts up well you can launch the daemon with:

```
$ sudo service bookie start
$ sudo service combo start
```

Nginx Config

Once that's started we just need to tell Nginx where to go access the application.

```
server {
  listen 80;
  server_name bookie;
  charset utf-8;

  root /home/$username/bookie/bookie/bookie/static;
  index index.html index.htm;

  # Remove trailing slash by doing a 301 redirect
  rewrite ^/(.*)/$ /$1 permanent;

  location ~*/(img|js|iepng|css)/ {
    root /home/$username/bookie/bookie/bookie;
    expires max;
    add_header Cache-Control "public";
    break;
  }

  location /combo {
    include     uwsgi_params;
    uwsgi_pass  unix:///tmp/convoy.sock;
    uwsgi_param UWSGI_SCHEME $scheme;
    break;
  }

  location / {
    include     uwsgi_params;
    uwsgi_pass  unix:///tmp/bookie.sock;
    uwsgi_param SCRIPT_NAME /;
    uwsgi_param UWSGI_SCHEME $scheme;
  }

  ## Compression
  # src: http://www.ruby-forum.com/topic/141251
  # src: http://wiki.brightbox.co.uk/docs:nginx

  gzip on;
  gzip_http_version 1.0;
  gzip_comp_level 2;
  gzip_proxied any;
  gzip_min_length  1100;
  gzip_buffers 16 8k;
  gzip_types text/plain text/html text/css application/x-javascript application/xml application/xml+r

  # Some version of IE 6 don't handle compression well on some mime-types, so just disable for them
  gzip_disable "MSIE [1-6].(?!.*SV1)";

  # Set a vary header so downstream proxies don't send cached gzipped content to IE6
  gzip_vary on;
  ## /Compression

}
```

From there we just need to check Nginx for any issues and reload it.

```
sudo nginx -t
sudo service nginx reload
```

6.4.3 Hosting with Apache and mod_wsgi

Apache and the mod_wsgi Apache module is the tried-and-true standard for WSGI serving. It also happens to be really easy to get your Bookie app working with it.

First you need to install Apache and mod_wsgi:

- On a Debian-based Linux (Ubuntu): apt-get install libapache2-mod-wsgi
- On other Linuxes: ?
- On OSX: ?
- On Windows: ?

Then you need to create a pyramid.wsgi file in the root of your Bookie virtualenv. Something like

```
import os
os.environ['NLTK_DATA'] = '/home/user/bookie/bookie/download-cache/nltk'
from pyramid.paster import get_app
application = get_app('/home/user/bookie/bookie/mybookie.ini', 'bookie')
```

A couple of things to check:

- The get_app path is correct for your system.
- If you're using SQLite, make sure you use the full path to it in your bookie/bookie/mybookie.ini

Next you need to add a virtualhost to your Apache config. You can either put this right in your httpd.conf or create a virtualhost for it.

```
WSGIApplicationGroup %{GLOBAL}
WSGIPassAuthorization On
WSGIDaemonProcess pyramid user=ben group=ben threads=4 \
    python-path=/home/user/bookie/lib/python2.6/site-packages
WSGIScriptAlias / /home/user/bookie/pyramid.wsgi

<Directory /home/user/bookie>
    WSGIProcessGroup pyramid
    Order allow,deny
    Allow from all
</Directory>
```

A couple of things you need to check:

- The python-path line matches the path to your virtualenv's site-packages.
- The WSGIScriptAlias in the example serves your Bookie install at the server's root. You can change that if you wish.
- The WSGIScriptAlias path to pyramid.wsgi is correct for your system.
- The Directory path is correct for your system. It should point to your virtualenv's root.

Finally, all you have to do is restart Apache and off you go!

- On a Debian-based Linux (Ubuntu): /etc/init.d/apache2 restart
- On other Linuxes: ?

- On OSX: ?
- On Windows: ?

For more help running Bookie under mod_wsgi on Apache, check out the modwsgi Pyramid Docs.

6.5 Development

Bookie is a Python web application written using the Pyramid web framework.

Bookie's git repository is managed using a tool/process called git flow. It basically sets standards for how the git repository is set up. You'll find the most up to date *working* code in the develop branch. Individual features that are being worked on are in branches prefixed by *feature/*. As these features get to a workable state they might get merged into the *develop* branch.

The *master* branch is only for releases and we're a long away from that. So when you check out Bookie, make sure to start out using the *develop* branch.

If you want to help out with the hacking of Bookie, here's some info you might want to check out:

6.5.1 Contribute To Bookie

To start contributing to Bookie, here is some info you might want to check out.

Quick Start

If you're on Ubuntu, you should be able to get started with:

```
$ git clone git://github.com/bookieio/Bookie.git
$ cd Bookie && make sysdeps && make install && make run
$ google-chrome (or other browser) http://127.0.0.1:6543
```

If you're on anything else, give our Vagrant image a try. If you don't have it already, you'll need to download and install Vagrant:

http://www.vagrantup.com/downloads.html

After that, you should be able to get started with:

```
$ git clone git://github.com/bookieio/Bookie.git
$ cd Bookie
$ vagrant up
$ vagrant ssh
% cd /vagrant
% make run
$ google-chrome (or other browser) http://127.0.0.1:4567
```

Note: If you run into problems during the *make sysdeps && make install* process, run *make clean_all* to reset the environment prior to re-running *make sysdeps && make install*.

If you're unable to complete the install process and need additional help please feel free to contact us in the #bookie IRC channel on Freenode, or the mailing lists.

Issues

The current issues related to Bookie can be seen at https://github.com/bookieio/Bookie/issues

Community

Our users and developers use Mailing lists and Internet Relay Chat (IRC).

Mailing List https://groups.google.com/forum/?hl=en#!forum/bookie_bookmarks

IRC http://webchat.freenode.net/?channels=bookie

Hacking

Typical Github workflow

Git allows you to work in a lot of different work flows. Here is one that works well for our environment, if you are not already familiar with git.

To set up the environment, first fork the repository. Once the fork is complete, create a local copy and work on a feature branch.

```
git clone git@github.com:{yourusername}/Bookie.git
cd Bookie
# Add a second remote to the upstream Bookie repository your fork came from.
# This lets you use commands such as `git pull bookieio develop` to update a
# branch from the original trunk, as you'll see below.
git remote add bookie git@github.com:bookieio/Bookie.git
# Create a feature branch to work on.
git checkout -b {featureBranchName}
# Hacky hacky hacky
```

To push code for review, cleanup the commit history.

```
# Optional: rebase your commit history into one or more meaningful commits.
git rebase -i --autosquash
# And push your feature branch up to your fork on Github.
git push origin {featureBranchName}:{featureBranchName}
```

In order to submit your code for review, you need to generate a pull request. Go to your Github repository and generate a pull request to the *bookie:develop* branch.

After review has been signed off on and the test run has updated the pull request, a member of the *Bookieio* organization can submit the branch for landing.

Once the code has been landed you can remove your feature branch from both the remote and your local fork. Github provides a button to do so in the bottom of the pull request, or you can use git to remove the branch. Removing from your local fork is listed below.

```
git push origin :{featureBranchName}
# And to remove your local branch
git branch -D {featureBranchName}
```

Before creating another feature branch, make sure you update your fork's code by pulling from the original Bookieio repository.

```
# Using the alias from the Helpful aliases section, update your fork with
# the latest code in the Bookie develop branch.
git sync-bookie

# And start your second feature branch.
git checkout -b {featureBranch2}
```

Syncing your feature branch with develop (trunk) Time to time you have a feature branch you've been working on for several days while other branches have landed in trunk. To make sure you resolve any conflicts before submitting your branch, it's often wise to sync your feature branch with the latest from develop. You can do this by rebasing your branch with develop.

The recommended pattern would be to

```
# Update your local copy of develop with the latest from the bookie branch.
git sync-bookie

# Then check back out your feature branch and sync it with your new local
# develop.
git checkout {featureBranch}
git rebase develop
```

You should see messages for each landed branch getting rebased into your work.

```
First, rewinding head to replay your work on top of it...
Applying: Created local charm new or upgrade inspector.
Applying: Refactored local charm upload helpers to support multiple service upgrades
```

Helpful Git aliases

Aliases Git provides a mechanism for creating aliases for complex or multi-step commands. These are located in your `.gitconfig` file under the `[alias]` section.

If you would like more details on Git aliases, You can find out more information here: How to add Git aliases

Below are a few helpful aliases we'll refer to in other parts of the documentation to make working with the Bookie easier.

```
###
### QA a pull request branch on a remote e.g. Bookieio
###

# Bring down the pull request number from the remote specified.
# Note, the remote that the pull request is merging into may not be your
# origin (your github fork).
fetch-pr = "!f() { git fetch $1 +refs/pull/$2/head:refs/remotes/pr/$2; }; f"

# Make a branch that merges a pull request into the most recent version of the
# trunk (the "Bookieio" remote's develop branch). To do this, it also updates your
# local develop branch with the newest code from trunk.
# In the example below, "bookie" is the name of your remote, "6" is the pull
# request number, and "qa-sticky-headers" is whatever branch name you want
# for the pull request.
# git qa-pr bookie 6 qa-adding-some-tests
qa-pr = "!sh -c 'git checkout develop; git pull $0 develop; git checkout -b $2; git fetch-pr $0 $1; g

# Update your local develop branch with the latest from the bookie remote.
# Then make sure to push that back up to your fork on github to keep
# everything in sync.
sync-bookie = "!f() { git checkout develop && git pull bookie develop && git push origin develop; };

# Rebase develop (trunk) into the current feature branch.
sync-trunk = rebase develop
```

6.5.2 Change Log

Ok, so I'm going to try to summarize changes here, but for full changes see the commit log. Commit often and all that.

What's new in 0.4

0.4 was all about the port from jQuery to YUI for the Javascript side of things. This meant that the UI got a big facelift, we've moved to the YUI MVC framework, and the extensions needed to be updated to use the new library of code.

We also spent time getting most of the fabric tasks moved over to the Makefile to help aid in running and managing the installation.

Celery has been introduced as a background task processor and we track stats for the system in there. Long running tasks, such as imports, and handled through the Celery background system.

The database migrations have been collapsed and ported over to run under alembic vs sqlalchemy-migrate. This really cleans up a lot of the old migrations where we used per-database fulltext support instead of Whoosh.

What's new in 0.3

The main goals of 0.3 were to add a full JSON API and to add authentication so that a single Bookie install could support multiple users. This involved a ton of changed code and so a lot more has changed in the process.

- new json api, see the api docs for details on usage. See *bookie.api.js* for an implementation of most of it.
- auth works. There's a whole forgotten password process and a fabric function for adding new users. We'll work on a real signup/register front end at some point before things get all public.
- Updated the CSS to be generated using Sass
- update to the chrome extension, supporting recent tags used, better completion, better error handling, and sync of bookmarks you've bookmarked before
- Lots of documentation updates. Updated install docs, full api docs, cleaned up some of the other docs in the process.
- update to pyramid 1.1
- added the start of the tag commands framework so that we can have *!sometag* perform a specific command on the server side when storing a bookmark

6.5.3 Makefile

Nearly everything about Bookie is managed via the Makefile. If you're not familiar with Makefiles, it's worth a little time to get your head around.

Commonly Used Make Targets

run

This command will start up the Bookie application along with the combo loader needed to serve the Javascript for Bookie.

stop

This will kill the running servers started up from the *make run*.

js

This command will check for updated Javascript library files and, if required, copy changed files to the build directory and minimize them.

run_dev

When doing development you might want some help keeping things "built" while you work. This command will also start up the sass watch process and a python script that will auto build changed Javascript files for you. This is how I tend to work and debug. For production purposes though, *make run* does everything you need.

stop_dev

This will kill things started via *make run_dev*.

test

Run the Python tests.

jstest

Open up all of the Javascript tests in the browser, one per tab.

db_up

Run any database migrations.

db_new

Start out a new migration file. Make sure to pass *desc="What is this migration"*.

clean

This will wipe the majority of the built files and resources. Think of it as a little bit of a hard reset.

all

Should recover froma a *make clean* and perform steps just as checking all deps are installed, the database is up to date, and the Javascript and CSS are up to date.

6.5.4 Setting Up Email Server

To work with code in the signup process it's often best to setup email communication for the site. It does not do this automatically and you need an smtp server for it to send email out to test accounts.

One way to setup an smtp server is to use a small tool, msmtp.

This assumes that you're on Ubuntu. You'll need to adjust these instructions for your own platform. If you get it working please feel free to submit a pull request with your platform as we'll happily add it to the docs.

```
$ sudo apt-get install msmtp
$ touch ~/.msmtprc
$ chmod 0600 ~/.msmtprc
```

Next, edit the *~/.msmtprc* file with your favourite text editor. The configuration file should contain the following lines.

```
defaults
account examplemail
host smtp.examplemail.com
tls on
tls_certcheck off
port 587
auth login
from somebody@example.com
user somebody@example.com
password somesecret

account default: examplemail
```

In the above example, the *host* has to be replaced with your email service smtp host. The *from*, *user*, and *password*, needs to be valid for your own email account.

Once that's complete, you can perform simple test to ensure your configuration is correct. Copy and paste these lines to your command prompt modifying the email address to your own address:

```
cat <<EOF | msmtp someone@example.com
Subject: test

This is a test!
EOF
```

If all the instructions are followed correctly, you can now receive the activation mail to the specified email address.

6.5.5 Bookie Tests

Running Tests

Running the test suite for Bookie is very simple:

Basic app tests

```
# Init the db first to prepare for running tests
$ INI=test.ini make test_bookie.db
$ make test
```

Javascript tests

```
$ make jstestserver

# open a new tab
$ make jstest
```

Other ways to run tests

- *make testcoverage*: will run the tests and generate the html coverage data in the cover directory for viewing.
- *make mysql_test*: Runs the tests against mysql using the test_mysql.ini file for processing.
- *make pgsql_test*: Runs the tests against postgres using the test_pgsql.ini file for processing.

Test Types (OUT OF DATE)

Unit Tests

Unit tests are small tests that should test small bits of code. These should be setup in the same directory that the file you're testing is setup. So if you're working on a file in *lib/feature.py* you'd have a matching file *test_feature.py*. This file should be runnable via the test runner by itself.

Functional Tests

Functional tests are larger scope tests that make sure the application is responding correctly as a whole. These are run through the fabric command *fab test*. It will run all tests defined in the tests directory.

Note: All unit tests should be added to the *tests/__init__.py* so that they get run during the large test run. This way the ci server will just need to run the one test pass and all tests will run during each build.

Testing Docs

A bit confusing. There's lots of docs, but none of them seem to agree on how to bootstrap the environment properly.

- http://docs.pylonsproject.org/projects/pyramid/dev/narr/testing.html
- https://bitbucket.org/sluggo/pyramid_sqla/src/d826ad458869/demos/SimpleDemo/simpledemo/tests.py
- http://docs.pylonsproject.org/projects/pyramid/1.0/tutorials/wiki2/definingviews.html#adding-tests

6.5.6 About JSON API

For best performance, and so that we can implement an api that meets the features we hold important we have our own api you can implement. It's JSON based and will return a standard JSON response for any call

All api calls should be against *https://$yoursite.com/api/v1*.

Remember, the only authentication method is the api key. If your site is not hosted behind secure http server then it's likely to get stolen. Please think about this before setting up a server exposed to the internet.

User specific calls

/:username/bmark

Usage *POST /api/v1/admin/bmark*

Submit a new bookmark for storing

query param api_key *required* - the api key for your account to make the call with

query param callback - wrap JSON response in an optional callback

post param url *required*

post param description

post param extended

post param tags - space separated tag string

post param content - html content of the page to be parsed as the readable version. if not provided, will be rendered by the celery job at some point in the future (or never if celery is not running).

post param is_private - specifies whether the bookmark is private or not. By default the bookmarks are stored as private

Status Codes

success 200 If successful a "200 OK" will be returned

error 403 if the api key is not valid or missing then this is an unauthorized request

All error responses will have a json body with an error message string and possibly other helpful information.

Example

```
requests.post('http://127.0.0.1:6543/api/v1/admin/bmark?api_key=12345...')
>>> {
        "bmark": {
            "username": "admin",
            "updated": "",
            "extended": "Extended notes",
            "description": "Bookie",
            "tags": [
                {
                    "tid": 2,
                    "name": "bookmarks"
                }
            ],
            "bid": 1,
            "stored": "2011-08-06 20:35:54",
            "inserted_by": "unknown_api",
            "is_private": true,
            "tag_str": "bookmarks",
            "clicks": 0,
            "hash_id": "c5c21717c99797"
        },
        "location": "http://localhost/bmark/readable/c5c21717c99797"
    }
```

/:username/bmark/:hash_id

Usage *GET /api/v1/admin/bmark/c605a21cf19560*

Get the information about this bookmark.

query param api_key *optional* - the api key for your account to make the call with

query param with_content - do you wish the readable content of the urls if available

query param url - This is the url of the page that you are trying to bookmark.This is used to supply tags in the Chrome extension.

query param description - This is the title of the page.This is used to supply tags in the Chrome extension.

query param callback - wrap JSON response in an optional callback

Status Codes

success 200 If successful a "200 OK" will be returned

error 404 if the hash id can not be found you'll get a 404

error 403 if the api key is not valid or missing then this is an unauthorized request

All error responses will have a json body with an error message string and possibly other helpful information.

Example

```
requests.get('http://127.0.0.1:6543/api/v1/admin/bmark/c605a21cf19560?api_key=12345...')
>>> {
      "bmark": {
        "bid": 2,
        "clicks": 1,
        "description": "Bookie: Recent Bookmarks",
        "extended": "",
        "hash_id": "c605a21cf19560",
        "inserted_by": null,
        "is_private": true,
        "stored": "2011-06-21 13:20:26",
        "tag_str": "test bookmarks",
        "tags": [
          {
            "name": "test",
            "tid": 3
          },
          {
            "name": "bookmarks",
            "tid": 2
          }
        ],
        "updated": "2011-07-29 22:23:42",
        "username": "admin"
      }
    }

requests.get('http://127.0.0.1:6543/api/v1/admin/bmark/c605a21cf19560?api_key=000')
>>> {"error": "Not authorized for request."}
```

Usage *POST /api/v1/bmark/admin/c605a21cf19560*

Update the stored bookmark with new information.

query param api_key *required* - the api key for your account to make the call with

query param callback - wrap JSON response in an optional callback

post param description

post param extended

post param tags - space separated tag string

post param content - html content of the page to readable parse

Status Codes

success 200 If successful a "200 OK" will be returned

error 404 if the hash id can not be found you'll get a 404

error 403 if the api key is not valid or missing then this is an unauthorized request

All error responses will have a json body with an error message string and possibly other helpful information.

Example

```
requests.post('http://127.0.0.1:6543/api/v1/bmark/admin/c605a21cf19560?api_key=12345...')
>>> {
        "bmark": {
            "username": "admin",
            "updated": "",
            "extended": "Extended notes",
            "description": "Bookie",
            "tags": [
                {
                    "tid": 2,
                    "name": "bookmarks"
                }
            ],
            "bid": 1,
            "stored": "2011-08-06 20:35:54",
            "inserted_by": "unknown_api",
            "is_private": true,
            "tag_str": "bookmarks",
            "clicks": 0,
            "hash_id": "c5c21717c99797"
        },
        "location": "http://localhost/bmark/readable/c5c21717c99797"
    }
```

Usage *DELETE /api/v1/bmark/admin/c605a21cf19560*

Remove the bookmark from the user's list

query param api_key *required* - the api key for your account to make the call with

query param callback - wrap JSON response in an optional callback

Status Codes

success 200 If successful a "200 OK" will be returned, with json body of message: done

error 404 if the hash id can not be found you'll get a 404

error 403 if the api key is not valid or missing then this is an unauthorized request

All error responses will have a json body with an error message string and possibly other helpful information.

Example

```
requests.delete('http://127.0.0.1:6543/api/v1/bmark/admin/c605a21cf19560?api_key=12345...')
>>> {
      "message": "done",
    }
```

/:username/bmarks

Usage *GET /api/v1/admin/bmarks*

Return a list of the most recent bookmarks

query param api_key *optional* - the api key for your account to make the call with

query param count - the number in the result you wish to return

query param page - the page number to get results for based off of the count specified

query param with_content - do you wish the readable content of the urls if available

query param callback - wrap JSON response in an optional callback

Status Codes

success 200 If successful a "200 OK" will be returned, with json body of message: done

error 403 if the api key is not valid or missing then this is an unauthorized request

Example

```
requests.get('http://127.0.0.1:6543/api/v1/admin/bmarks?count=2&api_key=12345...')
>>>{
    "count": 2,
    "bmarks": [
        {
            "username": "admin",
            "updated": "2011-07-29 22:23:42",
            "extended": "",
            "description": "Bookie: Recent Bookmarks",
            "tags": [
                {
                    "tid": 3,
                    "name": "test"
                },
                {
                    "tid": 2,
                    "name": "bookmarks"
                }
            ],
```

```
            "bid": 2,
            "stored": "2011-06-21 13:20:26",
            "inserted_by": null,
            "is_private": true,
            "tag_str": "test bookmarks",
            "clicks": 1,
            "hash_id": "c605a21cf19560",
            "url": "https://bmark.us/recent",
            "total_clicks": 5
        },
        {
            "username": "admin",
            "updated": "2011-07-15 14:25:16",
            "extended": "Bookie Documentation Home",
            "description": "Bookie Website",
            "tags": [
                {
                    "tid": 2,
                    "name": "bookmarks"
                }
            ],
            "bid": 1,
            "stored": "2011-06-20 11:42:47",
            "inserted_by": null,
            "is_private": true,
            "tag_str": "bookmarks",
            "clicks": 1,
            "hash_id": "c5c21717c99797",
            "http://docs.bmark.us",
            "total_clicks": 4
        }
    ],
    "tag_filter": null,
    "page": 0,
    "max_count": 10
}
```


/:username/bmarks/export

Usage *GET /api/v1/admin/bmarks/export*

Get a json dump of all of the bookmarks for a user's account. This will include all content that we have available. It will take a while to build and we will be limited this call to only a few times a day at some point.

query param api_key *required* - the api key for your account to make the call with

query param callback - wrap JSON response in an optional callback

Status Codes

success 200 If successful a "200 OK" will be returned, with json body of message: done

error 403 if the api key is not valid or missing then this is an unauthorized request

Example

```
requests.get('http://127.0.0.1:6543/api/v1/admin/bmarks/export?api_key=12345...')
>>> {
      "bmarks": [
        {
          "bid": 1,
          "clicks": 1,
          "description": "Bookie Website",
          "extended": "Bookie Documentation Home",
          "hash_id": "c5c21717c99797",
          "hashed": {
            "clicks": 4,
            "hash_id": "c5c21717c99797",
            "url": "http://bmark.us"
          },
          "inserted_by": null,
          "is_private": true,
          "stored": "2011-06-20 11:42:47",
          "tag_str": "bookmarks",
          "updated": "2011-07-15 14:25:16",
          "username": "admin"
        },
        {
          "bid": 2,
          "clicks": 1,
          "description": "Bookie: Recent Bookmarks",
          "extended": "",
          "hash_id": "c605a21cf19560",
          "hashed": {
            "clicks": 1,
            "hash_id": "c605a21cf19560",
            "url": "https://bmark.us/recent"
          },
          "inserted_by": null,
          "is_private": true,
          "stored": "2011-06-21 13:20:26",
          "tag_str": "test bookmarks",
          "updated": "2011-07-29 22:23:42",
          "username": "admin"
        },
        ...
      ],
      "count": 137,
      "date": "2011-08-08 20:11:43.648699"
    }
```

/:username/bmarks/popular

Usage *GET /api/v1/admin/bmarks/popular*

Return a list of the most clicked on bookmarks for the user.

query param api_key *optional* - the api key for your account to make the call with

query param count - the number in the result you wish to return

query param page - the page number to get results for based off of the count specified

query param with_content - do you wish the readable content of the urls if available

query param callback - wrap JSON response in an optional callback

Status Codes

success 200 If successful a "200 OK" will be returned, with json body of message: done

error 403 if the api key is not valid or missing then this is an unauthorized request

Example

```
requests.get('http://127.0.0.1:6543/api/v1/admin/bmarks/popular?count=2&api_key=12345...')
>>>{
    "count": 2,
    "bmarks": [
        {
            "username": "admin",
            "updated": "2011-07-29 22:23:42",
            "extended": "",
            "description": "Bookie: Recent Bookmarks",
            "tags": [
                {
                    "tid": 3,
                    "name": "test"
                },
                {
                    "tid": 2,
                    "name": "bookmarks"
                }
            ],
            "bid": 2,
            "stored": "2011-06-21 13:20:26",
            "inserted_by": null,
            "is_private": true,
            "tag_str": "test bookmarks",
            "clicks": 3,
            "hash_id": "c605a21cf19560",
            "url": "https://bmark.us/recent",
            "total_clicks": 5
        },
        {
            "username": "admin",
            "updated": "2011-07-15 14:25:16",
            "extended": "Bookie Documentation Home",
            "description": "Bookie Website",
            "tags": [
                {
                    "tid": 2,
                    "name": "bookmarks"
                }
            ],
            "bid": 1,
            "stored": "2011-06-20 11:42:47",
            "inserted_by": null,
            "is_private": true,
            "tag_str": "bookmarks",
            "clicks": 1,
            "hash_id": "c5c21717c99797",
            "http://docs.bmark.us",
```

```
            "total_clicks": 4
        }
    ],
    "tag_filter": null,
    "page": 0,
    "max_count": 10
}
```

/:username/extension/sync

Usage *GET /api/v1/admin/extension/sync*

This is experimental and very likely to change, so use at your own risk. We're investigating syncing bookmarks with browsers via their extensions. This api call will be the trigger point to allow a browser to request all of the data it needs for loading knowledge of existing bookmarks into a new browser installation.

query param api_key *required* - the api key for your account to make the call with

query param callback - wrap JSON response in an optional callback

Status Codes

success 200 If successful a "200 OK" will be returned, with json body of message: done

error 403 if the api key is not valid or missing then this is an unauthorized request

Example

```
requests.get('http://127.0.0.1:6543/api/v1/admin/extension/sync?api_key=12345...')
```

```
>>> {
        "94a2b635d965bc",
        "cf01b934863be8",
        ...
    }
```

/:username/bmarks/search/:terms

Usage *GET /api/v1/admin/bmarks/search/:terms*

Return a list of the user's bookmarks based on the fulltext search of the given terms. There can be one or more search terms. All search terms are *OR*'d together. Fulltext search will find matches in the *description*, *extended*, and *tag_string* fields of a bookmark. You can also perform fulltext search against the readable content of pages with the correct query parameter from below.

query param api_key *optional* - the api key for your account to make the call with

query param count - the number in the result you wish to return

query param page - the page number to get results for based off of the count specified

query param with_content - include the readable text in the fulltext search. This can slow down the response.

query param callback - wrap JSON response in an optional callback

Status Codes

success 200 If successful a "200 OK" will be returned, with json body of message: done

error 403 if the api key is not valid or missing then this is an unauthorized request

Example

```
requests.get('http://127.0.0.1:6543/api/v1/admin/bmarks/search/ubuntu/linux?api_key=12345...')
>>>> {
         "page": null,
         "phrase": "ubuntu",
         "result_count": 2,
         "search_results": [
           {
             "bid": 3,
             "clicks": 0,
             "description": "nickelanddime.png (PNG Image, 1200x1400 pixels) - Scaled (64%)",
             "extended": "This is the extended description",
             "hash_id": "adb017923e1f56",
             "inserted_by": "importer",
             "is_private": true,
             "stored": "2011-02-25 15:13:00",
             "tag_str": "nickelanddime kerfuffle banshee amazon ubuntu ubuntu-one canonical",
             "tags": [
               {
                 "name": "nickelanddime",
                 "tid": 4
               },
               {
                 "name": "canonical",
                 "tid": 10
               }
             ],
             "total_clicks": 0,
             "updated": "",
             "url": "http://www.ndftz.com/nickelanddime.png",
             "username": "admin"
           },
           {
             "bid": 77,
             "clicks": 0,
             "description": "My title: ubuntu forum archive about echolinux",
             "extended": "",
             "hash_id": "3e9a37d4f7cd74",
             "inserted_by": "importer",
             "is_private": true,
             "stored": "2010-07-08 19:30:18",
             "tag_str": "ham linux",
             "tags": [
               {
                 "name": "ham",
                 "tid": 89
               },
               {
                 "name": "linux",
                 "tid": 103
               }
             ],
```

```
            "total_clicks": 0,
            "updated": "",
            "url": "http://ubuntuforums.org/archive/index.php/t-973929.html",
            "username": "admin"
          }
        ],
        "username": "admin",
        "with_content": false
    }
```

/:username/social_connections/

Usage *GET /api/v1/admin/social_connections/*

Get a json dump of the social connections count for a user's account, usernames used in the social connections and refresh date i.e last time respective bot parsed the data from the social connection.

query param api_key *required* - the api key for your account to make the call with

Status Codes

success 200 If successful a "200 OK" will be returned

error 403 if the api key is not valid or missing then this is an unauthorized request

Example

```
requests.get('http://127.0.0.1:6543/api/v1/admin/social_connections/api_key=12345..')
>>> {
        "count": 2
        "social_connections": [{
         "username": "admin",
         "last_connection": "2014-06-12 17:39:41.855184",
         "uid": "1234",
         "type": "TwitterConnection"
         "twitterConnection": {
            "twitter_username": "bookie",
            "refresh_date": "2014-06-12 17:39:41.855202"
         }
        },{
         "username": "admin",
         "last_connection": "2014-06-12 17:41:09.720954",
         "uid": "1234",
         "type": "TwitterConnection"
         "twitterConnection": {
              "twitter_username": "bookie",
              "refresh_date": "2014-06-12 17:41:09.720954"
          }
        }]
    }
```

/:username/stats/bmarkcount

Usage *GET /api/v1/admin/stats/bmarkcount*

Get a json dump of the bookmark count for a user's account for a time period. The time period can be specified or else a json dump of the bookmark count of the past 30 days will be returned. If the start_date is specified to be the first day of the month and the end_date is not supplied, a json response of the bookmark count of the whole month will be returned.

query param api_key *required* - the api key for your account to make the call with

query param start_date *optional* - Find the bookmark count in the specified time window, beginning with start_date.

query param end_date *optional* - Find the bookmark count in the specified time window, ending with end_date.

Status Codes

success 200 If successful a "200 OK" will be returned

error 403 if the api key is not valid or missing then this is an unauthorized request

Example

```
requests.get('http://127.0.0.1:6543/api/v1/admin/stats/bmarkcount?start_date=2014-03-01&end_date=201
>>> {
        "count": [
          {
            "attrib": "user_bookmarks_admin",
            "data": 0,
            "id": 1,
            "tstamp": "2014-03-02 20:50:52"
          },
          {
            "attrib": "user_bookmarks_admin",
            "data": 3,
            "id": 10,
            "tstamp": "2014-03-03 20:50:52"
          },
          {
            "attrib": "user_bookmarks_admin",
            "data": 5,
            "id": 21,
            "tstamp": "2014-03-04 20:50:52"
          }
        ]
    }
```

/:username/tags/complete

Usage *GET /api/v1/admin/tags/complete*

Return a list of potential tags to use for the given *tag*.

query param api_key *optional* - the api key for your account to make the call with

query param tag *required* - the part of the word we want completions for

query param current - a space separated list of the current tags selected that we should take into account when selecting a potential completion option.

query param callback - wrap JSON response in an optional callback

Status Codes

success 200 If successful a "200 OK" will be returned, with json body of message: done

error 403 if the api key is not valid or missing then this is an unauthorized request

Example

```
requests.get('http://127.0.0.1:6543/api/v1/admin/tags/complete?api_key=12345...&tag=ubu')
>>> {
        current: "",
        tags: [
          "ubuntu",
          "ubuntuone"
        ]
    },
```

Account Information Calls

/:username/account

Usage *GET /api/v1/admin/account*

Return the name and email for the given user account.

query param api_key *required* - the api key for your account to make the call with

query param callback - wrap JSON response in an optional callback

Status Codes

success 200 If successful a "200 OK" will be returned, with json body of message: done

error 403 if the api key is not valid or missing then this is an unauthorized request

Example

```
requests.get('http://127.0.0.1:6543/api/v1/admin/account?api_key=12345...')
>>> {
        "username": "admin",
        "name": null,
        "signup": null,
        "activated": true,
        "last_login": null,
        "email": "testing@dummy.com"
    }
```

Usage *POST /api/v1/admin/account*

Update the user's name or email address

query param api_key *required* - the api key for your account to make the call with

query param callback - wrap JSON response in an optional callback

post param name - a new name for the user account

post param email - a new email for the user account

Status Codes

success 200 If successful a "200 OK" will be returned, with json body of message: done

error 403 if the api key is not valid or missing then this is an unauthorized request

Example

```
requests.post('http://127.0.0.1:6543/api/v1/admin/account?api_key=12345...')
>>> {
        "username": "admin",
        "name": null,
        "signup": null,
        "activated": true,
        "last_login": null,
        "email": "testing@dummy.com"
    }
```

/:username/api_key

Usage *GET /api/v1/admin/api_key*

Fetch the api key for the user from the system. We don't go waving the api key around so we have to ask for it on its own. Keep this safe. If it's exposed someone can get at about anything in the system for that user.

I know it's strange to require the api key to get the api key, but hey, you tell me how to fix it.

query param api_key *required* - the api key for your account to make the call with

query param callback - wrap JSON response in an optional callback

Status Codes

success 200 If successful a "200 OK" will be returned, with json body of message: done

error 403 if the api key is not valid or missing then this is an unauthorized request

Example

```
requests.post('http://127.0.0.1:6543/api/v1/admin/api_key?api_key=12345...')
>>> {
        "username": "someuser",
        "api_key": "12345..."
    }
```

/:username/api_key

Usage *POST /api/v1/admin/reset_api_key*

Request a brand new API key. The old API key will be invalidated. A new key will be generated and tied to your account. Please do not forget to update the API key in the browser extensions and other places where the API is used.

post param api_key *required* - the api key for your account to make the call with

post param username *required* - the username whose api key has to be reset

Status Codes

success 200 If successful a "200 OK" will be returned, with json body of message: done

error 403 If the api key is not valid or missing then this is an unauthorized request

Example

```
requests.post('http://127.0.0.1:6543/api/v1/admin/api_key?api_key=12345...')
>>> {
        "api_key": "98765...",
        "message": "API key was..."
    }
```

/:username/password

Usage *POST /api/v1/admin/account/password*

Change the user's password to the new value provided. Note that the current password is required to perform the step.

query param api_key *required* - the api key for your account to make the call with

query param callback - wrap JSON response in an optional callback

post param current_password *required* - the current password string from the user

post param new_password *required* - the string to change the password to

Status Codes

success 200 If successful a "200 OK" will be returned, with json body of message: done

error 403 if the api key is not valid or missing then this is an unauthorized request

error 406 if the new password is not of acceptable strength. We're not letting 2 char passwords to be set, sorry.

Example

```
requests.post('http://127.0.0.1:6543/api/v1/admin/password?api_key=12345...')
>>> {
        "username": "someuser",
        "api_key": "12345..."
    }
```

System wide calls

/bmarks

Usage *GET /api/v1/bmarks*

Return a list of the most recent bookmarks

query param api_key *optional* - the api key for your account to make the call with

query param count - the number in the result you wish to return

query param page - the page number to get results for based off of the count specified

query param with_content - do you wish the readable content of the urls if available

query param callback - wrap JSON response in an optional callback

Status Codes

success 200 If successful a "200 OK" will be returned, with json body of message: done

error 403 if the api key is not valid or missing then this is an unauthorized request

Example

```
requests.get('http://127.0.0.1:6543/api/v1/bmarks?count=2&api_key=12345...')
>>>{
    "count": 2,
    "bmarks": [
        {
            "username": "admin",
            "updated": "2011-07-29 22:23:42",
            "extended": "",
            "description": "Bookie: Recent Bookmarks",
            "tags": [
                {
                    "tid": 3,
                    "name": "test"
                },
                {
                    "tid": 2,
                    "name": "bookmarks"
                }
            ],
            "bid": 2,
            "stored": "2011-06-21 13:20:26",
            "inserted_by": null,
            "tag_str": "test bookmarks",
            "clicks": 1,
            "hash_id": "c605a21cf19560"
        },
        {
            "username": "admin",
            "updated": "2011-07-15 14:25:16",
            "extended": "Bookie Documentation Home",
            "description": "Bookie Website",
            "tags": [
                {
                    "tid": 2,
                    "name": "bookmarks"
                }
            ],
            "bid": 1,
            "stored": "2011-06-20 11:42:47",
            "inserted_by": null,
            "tag_str": "bookmarks",
            "clicks": 1,
            "hash_id": "c5c21717c99797"
        }
    ],
    "tag_filter": null,
    "page": 0,
```

```
    "max_count": 10
}
```

/bmarks/popular

Usage *GET /api/v1/bmarks/popular*

Return a list of the most clicked on bookmarks.

query param api_key *optional* - the api key for your account to make the call with

query param count - the number in the result you wish to return

query param page - the page number to get results for based off of the count specified

query param with_content - do you wish the readable content of the urls if available

query param callback - wrap JSON response in an optional callback

Status Codes

success 200 If successful a "200 OK" will be returned, with json body of message: done

error 403 if the api key is not valid or missing then this is an unauthorized request

Example

```
requests.get('http://127.0.0.1:6543/api/v1/bmarks/popular?count=2&api_key=12345...')
>>>{
    "count": 2,
    "bmarks": [
        {
            "username": "admin",
            "updated": "2011-07-29 22:23:42",
            "extended": "",
            "description": "Bookie: Recent Bookmarks",
            "tags": [
                {
                    "tid": 3,
                    "name": "test"
                },
                {
                    "tid": 2,
                    "name": "bookmarks"
                }
            ],
            "bid": 2,
            "stored": "2011-06-21 13:20:26",
            "inserted_by": null,
            "tag_str": "test bookmarks",
            "clicks": 3,
            "hash_id": "c605a21cf19560",
            "url": "https://bmark.us/recent",
            "total_clicks": 5
        },
        {
            "username": "admin",
            "updated": "2011-07-15 14:25:16",
```

```
            "extended": "Bookie Documentation Home",
            "description": "Bookie Website",
            "tags": [
                {
                    "tid": 2,
                    "name": "bookmarks"
                }
            ],
            "bid": 1,
            "stored": "2011-06-20 11:42:47",
            "inserted_by": null,
            "tag_str": "bookmarks",
            "clicks": 1,
            "hash_id": "c5c21717c99797",
            "http://docs.bmark.us",
            "total_clicks": 4
        }
    ],
    "tag_filter": null,
    "page": 0,
    "max_count": 10
}
```

/bmarks/search/:terms

Usage *GET /api/v1/bmarks/search/:terms*

Return a list of the user's bookmarks based on the fulltext search of the given terms. There can be one or more search terms. All search terms are *OR*'d together. Fulltext search will find matches in the *description*, *extended*, and *tag_string* fields of a bookmark. You can also perform fulltext search against the readable content of pages with the correct query parameter from below.

query param api_key *optional* - the api key for your account to make the call with

query param count - the number in the result you wish to return

query param page - the page number to get results for based off of the count specified

query param search_content - include the readable text in the fulltext search. This can slow down the response.

query param with_content - do you wish the readable content of the urls if available

query param callback - wrap JSON response in an optional callback

Status Codes

success 200 If successful a "200 OK" will be returned, with json body of message: done

error 403 if the api key is not valid or missing then this is an unauthorized request

Example

```
requests.get('http://127.0.0.1:6543/api/v1/bmarks/search/ubuntu?api_key=12345...')
>>>> {
         "page": null,
         "phrase": "ubuntu",
         "result_count": 2,
```

```
    "search_results": [
      {
        "bid": 3,
        "clicks": 0,
        "description": "nickelanddime.png (PNG Image, 1200x1400 pixels) - Scaled (64%)",
        "extended": "This is the extended description",
        "hash_id": "adb017923e1f56",
        "inserted_by": "importer",
        "stored": "2011-02-25 15:13:00",
        "tag_str": "nickelanddime kerfuffle banshee amazon ubuntu ubuntu-one canonical",
        "tags": [
          {
            "name": "ubuntu",
            "tid": 4
          },
          {
            "name": "canonical",
            "tid": 10
          }
        ],
        "total_clicks": 0,
        "updated": "",
        "url": "http://www.ndftz.com/nickelanddime.png",
        "username": "admin"
      },
      {
        "bid": 77,
        "clicks": 0,
        "description": "My title: ubuntu forum archive about echolinux",
        "extended": "",
        "hash_id": "3e9a37d4f7cd74",
        "inserted_by": "importer",
        "stored": "2010-07-08 19:30:18",
        "tag_str": "ham linux",
        "tags": [
          {
            "name": "ham",
            "tid": 89
          },
          {
            "name": "linux",
            "tid": 103
          }
        ],
        "total_clicks": 0,
        "updated": "",
        "url": "http://ubuntuforums.org/archive/index.php/t-973929.html",
        "username": "admin"
      }
    ],
    "username": "admin",
    "with_content": false
}
```


/suspend

Usage *POST /api/v1/suspend*

Creates a reset of the account. The user account is locked, an email is fired to the user's email address on file, and an activation code is contained within that is required to unlock the account.

query param api_key *required* - the api key for your account to make the call with

query param email *required* - the email address of the user we're wanting to reset

query param callback - wrap JSON response in an optional callback

Status Codes

success 200 If successful a "200 OK" will be returned, with json body of message: done

error 404 Could not find a user for this email address to suspend the account

error 406 No email address submitted in the request so we can't suspend anyone

Example

```
requests.post('http://127.0.0.1:6543/api/v1/suspend?api_key=12345...&email=testing@dummy.com')
>>> {
    "message":  """Your account has been marked for reactivation. Please check your email for instruc
}

requests.post('http://127.0.0.1:6543/api/v1/suspend?api_key=12345...')
>>> {
    "error":  "Please submit an email address",
}

requests.post('http://127.0.0.1:6543/api/v1/suspend?api_key=12345...&email=testing@dummy.com')
>>> {
    "error":  "You've already marked your account for reactivation.  Please check your email for the
    "username": admin
}
```

Usage *DELETE /api/v1/suspend*

Reactive the account. Basically we're "deleting the suspend" on the account. This requires the reactivation key that was sent to the user in the activation email.

query param username - string username of the user we're activating

query param activation - string activation code returned emailed from the POST call

query param password - a new password to reactivate this account to

query param callback - wrap JSON response in an optional callback

Status Codes

success 200 If successful a "200 OK" will be returned, with json body of message: done

error 406 The password supplied doesn't satisfy complexity requirements.

error 500 There was some issue restoring the account. Send for help

Example

```
requests.delete('http://127.0.0.1:6543/api/v1/suspend?api_key=12345&activation=behehe&password=admin'
>>> {
    "message": "Account activated, please log in",
    "username": "admin"
}

requests.delete('http://127.0.0.1:6543/api/v1/suspend?api_key=12345&activation=behehe&password=12')
>>> {
    "error": "Come on, pick a real password please"
}
```

/:username/invite

Usage *POST /api/v1/admin/invite*

Allows a user to create an invitation to another user in the system.

query param api_key *required* - the api key for your account to make the call with

query param email *required* - the email address of the new user to invite

query param callback - wrap JSON response in an optional callback

Status Codes

success 200 If successful a "200 OK" will be returned, with json body of message: done

error 406 No email address submitted in the request so we can't invite anyone

Example

```
requests.post('http://127.0.0.1:6543/api/v1/admin/invite?api_key=12345...&email=testing@dummy.com')
>>> {
    "message":  """done""",
}

requests.post('http://127.0.0.1:6543/api/v1/admin/invite?api_key=12345...')
>>> {
    "error":  "Please submit an email address",
}

requests.post('http://127.0.0.1:6543/api/v1/admin/invite?api_key=12345...&email=testing@dummy.com')
>>> {
    "error":  "This user has already been invited to the system.",
    "email": "testing@dummy.com"
}
```

/stats/bookmarks

Usage *GET /api/v1/stats/bookmarks*

Returns basic statistics about number of bookmarks in the database

Status Codes

success 200 If successful a "200 OK" will be returned, with json body of count, and unique_count

Example

```
requests.get('http://127.0.0.1:6543/api/v1/stats/bookmarks')
>>> {
    "count": 115047,
    "unique_count": 108060
}
```

/stats/users

Usage *GET /api/v1/stats/users*

Returns basic statistics about number of users in the database

Status Codes

success 200 If successful a "200 OK" will be returned, with json body of count, activations, and with_bookmarks

Example

```
requests.get('http://127.0.0.1:6543/api/v1/stats/users')
>>> {
    "count": 875,
    "activations": 133,
    "with_bookmarks": 388
}
```

Admin only calls

These are calls meant to help the admin with the system. Their documented for the project's need.

/a/accounts/invites

Usage *GET /api/v1/a/accounts/invites*

Return a list of the users and the number of invites they have.

query param api_key *required* - the api key for your account to make the call with

query param callback - wrap JSON response in an optional callback

Status Codes

success 200 If successful a "200 OK" will be returned

Example

```
requests.get('http://127.0.0.1:6543/api/v1/a/accounts/invites?api_key=12345...')
>>>{
        "users": [
            [
                "admin",
                11
            ],
```

```
            [
                "user2",
                0
            ]
        ]
    }
```

Usage *POST /api/v1/a/accounts/invites/:username/:count*

Set the invite_ct for the specified user to the specified count

query param api_key *required* - the api key for your account to make the call with

query param callback - wrap JSON response in an optional callback

Status Codes

success 200 If successful a "200 OK" will be returned.

Example

```
requests.get('http://127.0.0.1:6543/api/v1/a/accounts/invites/admin/10?api_key=12345...')
>>>{
       "count": 1,
       "users": [
           {
               "activated": false,
               "api_key": "12345",
               "email": "testing@someting.com",
               "id": 2,
               "invite_ct": 0,
               "invited_by": "admin",
               "is_admin": false,
               "last_login": "",
               "name": null,
               "password": null,
               "signup": "2010-04-07 17:50:18",
               "username": "admin"
           }
       ]
   }
```

/a/accounts/inactive

Usage *GET /api/v1/a/accounts/inactive*

Return the account info for users that are not set to active. Useful to see new signups that haven't activated or users with password/reset issues. New users will have their email address as their username since they've not set one yet.

query param api_key *required* - the api key for your account to make the call with

query param callback - wrap JSON response in an optional callback

Status Codes

success 200 If successful a "200 OK" will be returned.

Example

```
requests.get('http://127.0.0.1:6543/api/v1/a/accounts/invites?api_key=12345...')
>>>{
       "count": 1,
       "users": [
           {
               "activated": false,
               "api_key": "12345",
               "email": "newuser@something.com",
               "id": 2,
               "invite_ct": 0,
               "invited_by": "admin",
               "is_admin": false,
               "last_login": "",
               "name": null,
               "password": null,
               "signup": "2011-04-07 17:50:18",
               "username": "newuser@something.com"
           }
       ]
   }
```

/admin/readable/todo

GET /api/v1/admin/readable/todo

Return a list of urls that need to have content fetched for their readable views. This is used from external tools that will fetch the content and feed back into the api for readable parsing.

query param api_key *required* - the api key for your account to make the call with

query param callback - wrap JSON response in an optional callback

```
requests.get('http://127.0.0.1:6543/api/v1/a/readable/todo?api_key=12345...')
>>> {
      message: ""
      payload: {
        urls: [
            ...
        ]
      }
      success: true
    }
```

/admin/readable/statuses

@todo Provide statics of the status code of readable attempts

/admin/readable

@todo Provide some readable details, number of outstanding bookmarks to read, number with readable content, etc.

/admin/:username/deactivate

@todo Mark a user as disabled. Will not allow them to login, save bookmarks, use the api

/a/users/list

Usage *GET /api/v1/a/users/list*

Return a list of the users in the system.

query param api_key *required* - the api key for your account to make the call with

query param callback - wrap JSON response in an optional callback

Status Codes

success 200 If successful a "200 OK" will be returned

Example

```
requests.get('http://127.0.0.1:6543/api/v1/a/users/list?api_key=12345...')
>>>{
        "count": 10,
        "users": [
            [
                "admin",
                ...
            ],
            [
                "user2",
                ...
            ]
        ]
    }
```

/a/users/add

Usage *POST /api/v1/a/users/add*

Admin override and add a new user to the system.

query param api_key *required* - the api key for your account to make the call with

query param username *required* - the username of the new user

query param email *required* - the email address of the new user

query param callback - wrap JSON response in an optional callback

Status Codes

success 200 If successful a "200 OK" will be returned

Example

```
requests.post('http://127.0.0.1:6543/api/v1/a/users/list?api_key=12345...', {
    'email': 'test@dummy.com',
    'username': 'test',
})
>>>{
        "username": "admin",
        "email": "test@dummy.com",
        "id": 11,
        "random_pass": "blah123",
        ...
    }
```

/a/users/delete/:username

Usage *DELETE /api/v1/a/users/delete/:username*

Admin endpoint to remove a user from the system.

Currently meant for bad new user accounts that removes activation and user account. Does not reach into bmarks/tags.

query param api_key *required* - the api key for your account to make the call with

query param callback - wrap JSON response in an optional callback

Status Codes

success 200 If successful a "200 OK" will be returned

Example

```
requests.post('http://127.0.0.1:6543/api/v1/a/users/delete/admin?api_key=12345...')
>>>{
        "success": true,
        "message": "Removed user: admin"
    }
```

/admin/stats/bmarks

GET */api/v1/admin/stats/bmarks*

Return the most recent counts of bookmarks, tags, and unique bookmarks

query param api_key *required* - the api key for your account to make the call with

query param count - the number in the result you wish to return

query param page - the page number to get results for based off of the count specified

query param callback - wrap JSON response in an optional callback

```
requests.get('http://127.0.0.1:6543/api/v1/admin/stats/bmarks?api_key=12345...')
>>> ...
```

/a/social/twitter_refresh/:username

GET */a/social/twitter_refresh/:username*

Refresh twitter fetch for specific user

query param api_key *required* - the api key for your account to make the call with

query param callback - wrap JSON response in an optional callback

```
requests.get('http://127.0.0.1:6543/api/v1/a/social/twitter_refresh/admin?api_key=12345...')
>>> {
        "message": "running bot to fetch user's tweets"
        "success": true,
    }
```

/a/social/twitter_refresh/all

GET */a/social/twitter_refresh/all*

Refresh twitter fetch for all the users

query param api_key *required* - the api key for your account to make the call with

query param callback - wrap JSON response in an optional callback

```
requests.get('http://127.0.0.1:6543/api/v1/a/social/twitter_refresh/all?api_key=12345...')
>>> {
        "message": "running bot to fetch user's tweets"
        "success": true,
    }
```

Delicious API

Since we started out attempting to match the Delicious api, we support some of those features. Not all of them make sense, so not all are implemented. Currently, the browser extensions communicate to the server via the Delicious api calls. Eventually, we'll probably move those over to the official JSON api as I much prefer JSON and hate dealing with the XML calls that Delicious implemented.

All of our api calls are POST since we allow for some large content payloads.

API Key

All of our delicious.com api calls that make changes to the database, require an *api_key* parameter to be passed with the request. This is a slight deviation from the Delicious API since we do not currently support login.

Available API Calls

***/delapi/posts/add*:** See: http://www.delicious.com/help/api#posts_add We also support an extra parameter *content* that is html content for the bookmark you'd like parsed and stored as its readable content. The Chrome extension currently supports this as an option and is meant to help provide readable content immediately vs whenever a cron script can fetch and load a page.

***/delapi/posts/delete*:** See: http://www.delicious.com/help/api#posts_delete Other than the *api_key* parameter this is just pass a url and it'll get deleted.

***/delapi/posts/get*:** See: http://www.delicious.com/help/api#posts_get We only support passing a *url* and do not support getting by tag, hash, etc. This does not require an *api_key* since there are no changes to the database to be made.

***/delapi/tags/complete*:** This is not an delicious api call, but is currently stored in here. It's meant for providing tag autocomplete options to a widget based on current input. You must pass a *tag* with the characters entered so far. It also optionally supports a *current_tags* parameter so that completion will take into account existing tags. You can see this in action at the demo site tag filter at http://bmark.us

6.5.7 Things on the Todo list

For things on the todo list see the Trello board:

https://trello.com/board/bookie/4f18c1ac96c79ec27105f228

6.5.8 Hacking Events

Sprint: Saturday Aug 31st

Location @mitechie's house. Ping him for details.

Time 11am

Tasks

The main task is to work on test coverage of Bookie. We'll be using coverage.py to find areas missing tests and work on getting better coverage for them.

A couple of people have expressed interest in working on breadability and some sites it's not processing that well.

Lunch will be provided and if you're interesting in working on something else please let me know. The day will end when people get tired of sprinting.

Sprint: July 27th and 28th

Location PyOhio see @mitechie for where on Friday and Saturday info is on the

Site http://pyohio.org/Sprints/

Time Refer to the PyOhio site for details.

Tasks

We'll be working on any of the Bookie part of apps. This includes breadability, bookie_parser, bookie_firefox, bookie_android. So if you want to hack on a Pyramid app, celery processing, parsing algorigthms, Tornado apps on Heroku, or CoffeeScript we've got something you can work on.

Specific task ideas include:

- Introduction to bookie and working on small bugs from the issue list.
- Update Bookie's readable parsing to be Celery driven to the bookie_parser service api.
- Work on Bookie's mobile/responsive UI and a possible two pane reader layout.
- Update Bookie's celery backend and port more current cron scripts over to Celery tasks.
- Working on fixing the breadability parser for web sites it's failing to parse

- Update breadability to provide metadata on processing including potential backup nodes of content, timing information, etc.
- Update the bookie_parser application to have some better webui.
- Add full metadata fetching of content from bookie_parser.
- Work on turning Bookie's Firefox extension into a CoffeeScript application that works against the bookie API.

Sprint: PyCon! March 12th-15th

Location PyCon Sprints!

Time All the time!

Tasks

- Get 0.4 out the door, this means FF extension completed and do a release
- Start 0.5 release, possible items include:
  - Signup system with throttled registrations/waiting list + api/ui for it
  - In place editing
  - Easy reader UI for !toread bmarks
  - Look at adding smarter tag suggestions (js page parser + smarter server side)
  - Celery/out of process worker system for things
  - Rework the url parsing worker for the celery backend, requests, async
  - get yeti and browser functional tests running SST?
  - better mobile/responsive UI bits
  - Stats stats and more stats API/UI

Sprint: Feb 25th

Location: My Place Time: 10am-4pm

Tasks

The main task is to work on a full Firefox extension based off the add-on SDK. Current base is located:

- https://github.com/bookieio/bookie-firefox

Misc Tasks

A lot's changed. We could use help with:

- Installation testing
- Updating the current failing unit tests broken by recent code changes
- Documentation updates

- Working on UI updates/css tweaks for the mobile/media query drive UI
- Adding JS tests for the new JS driven UI
- Implementing DELETE in the chrome extension
- anything else we can think up.

Sprint: July 29th and 30th

Location: PyOhio see @mitechie for where on Friday and Saturday info is on the site: http://pyohio.org/Sprints/ Time: Friday will be in the evening and Saturday with the PyOhio peeps after the conference

Tasks

- Update documentation for the full set of api commands (see routes.py and views/api.py)
- Update documentation with screenshots from more recent builds (including the extension, website, and such)
- Generally documentation updates/clarifications regarding feature sets and such added in 0.3. A full changelog based on commits from 0.2 to current
- Testing of the mobile view against android and iOS for issues with jQuery mobile updates
- Adding a "save bookmark" form view. Basically implementing the extension popup in the website.
- Adding a bookmarklet that uses the above form to save the current page via the bookmarklet so it works from mobile and non-chrome browsers
- Keep working on updated FF extension using the extension builder code in the FF branch under the firefox_ext_addon_sdk

Current 0.4 Tasks

- Update to pyramid 1.1
- add some css styling to the readable view for website + mobile view
- Add a !private tag command which sets private on the bookmark of the user
- Start celery task runner for running stats against individual bookmarks daily

Sprint: April 22nd 2011:

The plan

We're going to have the first ever Bookie sprint on Friday the 22nd of April. Some potential goals:

- Work on getting the FF plugin working
- Some UI design/ideas pitching/feedback
- Test out the new readable parsing on everyone's batches of bookmarks
- Work on some docs updates and try to knock out a few items from the issue list for 0.2 : https://github.com/bookieio/Bookie/issues?milestone=5&state=open

Schedule

The doors open up at 11am and we'll have some lunch delivered around 12:30pm. I'll chase everyone away somewhere around 4pm.

6.6 Bookie Features

6.6.1 Open source!

We're big fans of Open Source development. Bookie is AGPL licensed and we encourage you to fork it from Github and try it out. Have a pet feature you'd love to work with? Then let us know. Jump into our IRC channel #bookie on freenode to chat with the other developers.

6.6.2 Importing bookmarks

At this time we support importing the Delicious and Google Bookmark exports. They're just html files. The importer detects which format you're got and will process them all in one swoop. It should run pretty quickly, though if you have over 5,000 bookmarks you might notice it taking a little bit to get through them.

When importing, make sure you enter the API key that is set in your installation's *.ini* file. That's the security measure that prevents others from importing into your Bookie installation. That key should be changed from the default.

6.6.3 Google Chrome extension

Most of the details are available over in the extension docs. Make sure that after you install it, that you go into the options to configure it to talk to your specific Bookie installation.

6.6.4 Firefox Extension

In the works, see the extension docs.

6.6.5 Bookmarklet

You can use a bookmarklet to save bookmarks from any other browser, including mobile browsers. Log into your account and you can get a copy/paste-able bookmarklet from your account page.

6.6.6 Fulltext indexing and seach

When you load bookmarks into the system they are put into a fulltext index that makes things very searchable. The description field, extended description, and the tags are put into this index.

If you have enabled the *readable* parsed version of your bookmarks then these are also available for search, however it's a separate checkbox as it might make searches slightly slower.

These indexes are updated as you add and delete bookmarks and should make finding things much easier thank just looking for tags.

6.6.7 *Readable* cached copy of page content

Bookie now supports storing a cleaned version of the content of the page you just bookmarked. You can either enable it via the Chrome extension options or you can run the server side script provided in *scripts/readability/existing.py* to collect the page content. It's not perfect, but in testing it provides a decent trimmed down version of the pages. This content is then indexed and made searchable. In the future we hope to use this to provide fast mobile viewing of your bookmarks and possibly even the ability to build 'books' of content that can be packaged together and turned into e-book material.

This is all very much inspired by Instapaper and Readability.

If you find pages of content that don't work well please let us know and we can see if the code used to do the parsing can be tweaked to do a better job with your content.

6.6.8 Popularity Tracking

Every time a bookmark url is clicked on it's tracked and counted. This allows us to provide a view of your bookmarks by popularity. We're hoping this provides a very useful interface when we start working on our mobile views. In the future we might also be able to provide some analytics much like bit.ly does for shortened urls.

6.6.9 Multi database support

Bookie currently supports Sqlite, MySQL, and Postgresql as a storage backend. This includes all of the fulltext indexing and searching. Obviously, how each database performs these is a little bit different so you might find better luck with some backends over others.

Mobile Friendly

We're working on making the website mobile friendly using responsive design techniques.

6.7 Hacking Events

6.7.1 Sprint: Saturday Aug 31st

Location @mitechie's house. Ping him for details.

Time 11am

Tasks

The main task is to work on test coverage of Bookie. We'll be using coverage.py to find areas missing tests and work on getting better coverage for them.

A couple of people have expressed interest in working on breadability and some sites it's not processing that well.

Lunch will be provided and if you're interesting in working on something else please let me know. The day will end when people get tired of sprinting.

6.7.2 Sprint: July 27th and 28th

Location PyOhio see @mitechie for where on Friday and Saturday info is on the

Site http://pyohio.org/Sprints/

Time Refer to the PyOhio site for details.

Tasks

We'll be working on any of the Bookie part of apps. This includes breadability, bookie_parser, bookie_firefox, bookie_android. So if you want to hack on a Pyramid app, celery processing, parsing algorigthms, Tornado apps on Heroku, or CoffeeScript we've got something you can work on.

Specific task ideas include:

- Introduction to bookie and working on small bugs from the issue list.
- Update Bookie's readable parsing to be Celery driven to the bookie_parser service api.
- Work on Bookie's mobile/responsive UI and a possible two pane reader layout.
- Update Bookie's celery backend and port more current cron scripts over to Celery tasks.
- Working on fixing the breadability parser for web sites it's failing to parse
- Update breadability to provide metadata on processing including potential backup nodes of content, timing information, etc.
- Update the bookie_parser application to have some better webui.
- Add full metadata fetching of content from bookie_parser.
- Work on turning Bookie's Firefox extension into a CoffeeScript application that works against the bookie API.

6.7.3 Sprint: PyCon! March 12th-15th

Location PyCon Sprints!

Time All the time!

Tasks

- Get 0.4 out the door, this means FF extension completed and do a release
- Start 0.5 release, possible items include:
  - Signup system with throttled registrations/waiting list + api/ui for it
  - In place editing
  - Easy reader UI for !toread bmarks
  - Look at adding smarter tag suggestions (js page parser + smarter server side)
  - Celery/out of process worker system for things
  - Rework the url parsing worker for the celery backend, requests, async
  - get yeti and browser functional tests running SST?
  - better mobile/responsive UI bits

  - Stats stats and more stats API/UI

6.7.4 Sprint: Feb 25th

Location: My Place Time: 10am-4pm

Tasks

The main task is to work on a full Firefox extension based off the add-on SDK. Current base is located:

- https://github.com/bookieio/bookie-firefox

Misc Tasks

A lot's changed. We could use help with:

- Installation testing
- Updating the current failing unit tests broken by recent code changes
- Documentation updates
- Working on UI updates/css tweaks for the mobile/media query drive UI
- Adding JS tests for the new JS driven UI
- Implementing DELETE in the chrome extension
- anything else we can think up.

6.7.5 Sprint: July 29th and 30th

Location: PyOhio see @mitechie for where on Friday and Saturday info is on the site: http://pyohio.org/Sprints/ Time: Friday will be in the evening and Saturday with the PyOhio peeps after the conference

Tasks

- Update documentation for the full set of api commands (see routes.py and views/api.py)
- Update documentation with screenshots from more recent builds (including the extension, website, and such)
- Generally documentation updates/clarifications regarding feature sets and such added in 0.3. A full changelog based on commits from 0.2 to current
- Testing of the mobile view against android and iOS for issues with jQuery mobile updates
- Adding a "save bookmark" form view. Basically implementing the extension popup in the website.
- Adding a bookmarklet that uses the above form to save the current page via the bookmarklet so it works from mobile and non-chrome browsers
- Keep working on updated FF extension using the extension builder code in the FF branch under the firefox_ext_addon_sdk

Current 0.4 Tasks

- Update to pyramid 1.1
- add some css styling to the readable view for website + mobile view
- Add a !private tag command which sets private on the bookmark of the user
- Start celery task runner for running stats against individual bookmarks daily

6.7.6 Sprint: April 22nd 2011:

The plan

We're going to have the first ever Bookie sprint on Friday the 22nd of April. Some potential goals:

- Work on getting the FF plugin working
- Some UI design/ideas pitching/feedback
- Test out the new readable parsing on everyone's batches of bookmarks
- Work on some docs updates and try to knock out a few items from the issue list for 0.2 : https://github.com/bookieio/Bookie/issues?milestone=5&state=open

Schedule

The doors open up at 11am and we'll have some lunch delivered around 12:30pm. I'll chase everyone away somewhere around 4pm.

CHAPTER 7

Upcoming Bookie Events

Check out the events schedule page for some dates of sprints and hacking times for Bookie.

Our next sprint is at PyOhio so come see us there. Check out the events page for details on what we'll be hacking on!