Hacker Guides CMS Documentation

Release .1

Pluralsight LLC

Jul 19, 2018

Contents

Contents:

This repository is no longer used or maintained.

CHAPTER 1

hack.guides()

hack.guides() is an open-source CMS based on Markdown and Github written by-developers, for-developers. The CMS is open-source (this repo). All the content i.e. hacker guides are also stored in an open-source Github repo located here. See the official hack.guides() website for an example of what a running version of this CMS looks like.

The aspiration of hack.guides() is to be an open-source community movement to help bring knowledge to the world. Software developers learn new skills, and collaborate together on technical writing. Ultimately we aspire to have this content delivered to other spoken languages through a community of volunteer translators. Our vision is to democratize professional technology learning.

hack.guides() is subsidized by Pluralsight. It is open-sourced via the AGPL 3.0 license.

1.1 Developer Documentation

Please see the following documentation on Read the Docs for more detailed documentation of the code of the CMS.

http://hacker-guides-cms.readthedocs.io/en/latest/

1.2 Getting Involved

We're working to keep making this community project better. You have great ideas and expertise that could help us! Take a look at our current issues and let us know where we can improve. See something you can fix? Send us a Pull Request!

Please join our Slack community to be in touch for fastest response.

CHAPTER 2

CHANGELOG

2.1 version .6 - 7/7/16

2.1.1 New Features

- Revamped design to be more colorful with stack images for every guide, etc.
- **Added ability to heart guides**
  - Requires use of redis and use of *ENABLE_HEARTING* and *REDIS_HEARTS_DB_URL* environment variables
- Support for Github Bio on profile and article pages
- Improved support for Facebook Open Graph tags on homepage and article
- Improved layout of review page and home page when there is no featured guide
- **New stack images**
  - Thanks @eh3rrera!
- **Use Github webhooks to keep list of branches and cache up to date**
  - Requires use of *WEBHOOK_SECRET* environment variable
- New page highlighting tutorial contest
- Properly escape all code on article pages, not just HTML

2.1.2 Bug Fixes

- Fix exception when running with empty REDISCLOUD_URL env variable
- Fix html escaping issues on article page
- Shorten long author names to prevent from breaking out of design boxes

- KeyError exception on some invalid page URLs
- Ignore invalid statuses when reading article
- Error when handling failed github authentication request
- Bug with removing a branch when it being added again
- Problem unpredictable featured guide when two guides have the same title

2.1.3 Changes

- Add CTA to article list when filtering returns no results
- Guides are now grouped by publish status on profile page
- All markdown is rendered on front-end with Javascript instead of using Github API
- Changed copy on login page to be more informative
- New logo highlighting our sponsor, Pluralsight
- Store featured guide with redis
- Remove case-insensitive comparison for featured guide environment variable

2.2 version .5 - 5/9/16

2.2.1 New Features

- **Logging of Github API rate limit**
  - See *bin/rate_limit_watcher.py* which can be used with Heroku Scheduler add-on or *cron* in any UNIX environment
- **Added *newrelic* to requirements for performance monitoring**
  - This is optional, but still in the *requirements.txt* file.
- **Added full-screen editor as default and removed non-full screen**
  - This resulted in a lot of improvments including simpler CSS, better integrated help, tooltips, modal error dialogs, and a full-screen view with all possible controls readily available.
- Big speed improvements to editor
- Added links to hack.pledge and hack.summit in header
- Show list of contributors on guide page

2.2.2 Bug Fixes

- Edit guide link is broken after changing publish status
- Editor removes escape characters even if in a codeblock
- Prevent extra commit to metadata file on first edit
- Image uploader doesn't set committer name correctly on commits
- Fixed URLs involving a branch name with special characters

- Fixed problems with guide titles having special URL characters
- Remove unecessary Github API request when fetching contributor lists
- Fetching contributors lists twice for guides with no contributors
- Prevent mobile share buttons and email signup box overlapping
- Incorrectly including any user with a branch as a contributor
- Faster loading of rendered markdown from Github API
- Fixed escape <script> tag in article content
- Properly serialize file listing to cache with unicode
- Properly show featured guide on my-drafts page
- **Add contributor page with leaderboards**
  - Introduces *IGNORE_STATS_FOR* environment variable

2.2.3 Changes

- Disable save button on editor until a title has been chosen
- Improved 'Live Markdown Tutorial' UI to include a more prominent 'Close Tutorial' button
- Renamed 'Cancel' button on editor to 'Back'
- **Branches are named after user and guide, not just user**
  - Makes merging changes much easier since each branch only deals with a single guide
- Improved load time of FAQ page
- Redirect to master branch if branched guide is missing
- Do not show users in IGNORE_STATS_FOR environment variable in contributor lists
- Use username/login in profile page title
- 'Allow redirect URLs file to contain markdown lists https://github.com/pluralsight/guides-cms/commit/a83155605492dd7da65af662de1e3d937f56be68

2.3 version .4 - 4/5/16

2.3.1 New Features

- Live markdown tutorial in new editor
- Auto save guide text using HTML5 local storage
- Side-by-side markdown preview
- Optional scroll-sync between text and markdown preview panes
- Ability to add images to guides via standard file dialog
- Support for 301 redirects for guides (see *redirects file*)
- Easier signup to Slack community via popup box on FAQ page

2.3.2 Bug Fixes

- Links in editor preview open in new tabs
- Use proper HTTP status codes for redirects requiring authentication
- Properly escape characters in Table of Contents (see issue
- Incorrect links to branched guides on main guide page
- Overlapping of table of contents with footer
- Do not show users' drafts on profile page unless logged in as user
- Prevent errors on redundant publish status changes
- Prevent making API calls for URLs that do not look like guides on guide page
- Issue losing list of branches when saving original article after branched
- Issue with /user/ returning articles of repo owner instead of error
- Making a commit with wrong user name by incorrectly reading user cache (see commit
- Maintain social share counts for po.st with new URL structure introduced in v.3

2.3.3 Changes

- Changed editor from Bootstrap Markdown editor to Ace
- Show published guides instead of error page when unable to find requested guide
- Improved caching of file listings for homepage and review pages
- Add better explanation of publish workflow after submitting a new guide
- Improve error message when creating duplicate guide with title/stack
- Removed redundant 'Edit guide' link in header on guide page
- Removed form to set featured article
- Use /author/<name>/ URL for authors instead of user, 301 redirect from /user/<name>

2.4 version .3 - 3/11/16

2.4.1 Bug Fixes

- Fix bug with not checking for article existence on editor page
- Fix link for featured article after redesign
- Fix bug with file listing getting updated with publish status before it changed

2.5 version .2 - 3/11/16

2.5.1 Changes

1. Three stage publish workflow

Draft

The initial stage where all guides start out in. Guides in this stage are not visible by anyone other than the original author. [1]

All guides marked as unpublished will be moved to draft stage during the upgrade process. Therefore, initially there will be no guides in the in-review stage.

In-review

The second stage were guides go that are ready for community editing help. Any user can mark their guide as 'in-review' from dropdown at the bottom of the guide page.

Guides should only be marked as 'in-review' when they are complete and ready editing help.

Please don't mark partially completed guides as in-review. This will necessarily waste community editors time reviewing guides that are not completed.

Guides marked as 'in-review' will show up on the 'Review' page.

Published

The final stage for fully edited articles is published. This is the stage where the community editors have decided a guide is ready for the world to see. Only community editors can move a guide into the published stage.

Published articles will be available on the homepage of the site.

2. Redesign of the content repository

The content repository is currently a flat structure. This means all the guides are directly at the top level of the repository, which makes it difficult to easily navigate on the github.com repository view. This pull request reorganizes the repository to use a more intuitive and nested layout based on the publish status of the guide as well as the stack. For example, each publish stage will have a folder with a nested folder for each stack:

This will make quickly browsing the content much easier on github.com.

3. URL redesign (with backwards compatability)

The URL scheme has been redesigned to include the stack. This gives visitors more insight into the type of guide by looking only at the URL.

Therefore, the guide URL will now be something like:

- */python/my-awesome-guide*

instead of

- */my-awesome-guide*

All the old URLs with only the title remain intact with a 301 redirect at the */review/* endpoint.

Also, the status of a guide is represented by a query string, not directly in the URL as before. So, the following URL will point to a guide in the in-review stage:

- */python/my-awesome-guide?status=in-review*

instead of

- */review/my-awesome-guide*

This will allow articles to keep the same URL through the entire publish workflow, improving their SEO and link maintainability. In addition, visitors can clearly see in the URL the publish status of a guide. Soon there will be a more visual way to see the status on the guide page itself, but not in this change.

Note that changing the stack of your article **will** change the URL of your guide. Therefore, change this with caution to avoid losing any SEO you might have gathered on the old URL. Typically you should not be changing your stack after you're in the 'in-review' stage.

4. Github commits only involve guide author

Previously all commits to guides were pushed to github with a different author and committer. The committer was marked as the owner of the content repository. This lead to a commit having a different author and committer, which is confusing on github.com. Now all commits will have the same committer and author to avoid this confusion. **You as the author still get full contribution credit, which will show up on your github.com profile.** This change just gives you commit credit **by youreself.**

5. Ability to change stack guide

This is not a recommended action because it will change a guides URL, which is not ideal for SEO and link preservation. However, it is now allowed.

Upgrading

See the upgrade_repo_layout_fromv.1.py script for details on the content repository conversion process. The upgrade script will use *git mv* to move all guide diretories to their new locations thereby retaining the commit history.

All guides marked as unpublished will be moved to draft stage during the upgrade process. Therefore, initially there will be no guides in the in-review stage.

1. Run upgrade script on your content repository
2. Run merge_branches.py and use the branch you used from step 1 to merge with.
3. Push all remote branches to origin
4. Push your master branch to origin
5. Deploy new version of the CMS
6. Run disqus redirect crawler to update URLs for all comments.

[1] We don't have strict privacy since the guides are also available on github.com. So, technically a draft guide can still be viewed directly on github, but there will be no way for users to see draft guides directly on the content website.

2.5.2 Bug Fixes

- Improve commit messages when removing guides

2.6 version .1 - 2/23/16

Initial open source release during http://hacksummit.org.

CHAPTER 3

Requirements

- Programming lanuage: Python 2.7.10
- Web framework: Flask
- HTTP server: Gunicorn
- See requirements.txt for additional Python package requirements
- Background jobs: Redis

3.1 Optional requirements

- Caching: Redis

CHAPTER 4

Install for local development

1. **Clone repo**
    - *git clone <repo> <location_to_clone>*
2. Install virtualenv
3. **Create virtual environment for project**
    - *virtualenv <env>* where <env> is location to where you want to store project environment. <env> is typically the directory of your git repo or something like *~/.virtualenvs/env_name*.
4. **Activate virtualenv**
    - *source <path_to_env>/bin/activate*
5. **Install requirements**
    - *pip install -r requirements.txt*
6. Setup *Github application*

4.1 Setup environment variables

1. **Copy example_config.py to config.py and populate config.DevelopmentConfig with your own data.**
    - This is where you'll copy and paste your Github OAuth application's credentials.
    - **The defaults are all set in the config.Config so override only values you need. The following are required:**
        - SECRET_KEY
        - GITHUB_CLIENT_ID
        - GITHUB_SECRET
        - REPO_OWNER - Name of your github user

- REPO_NAME - Name of repository you'll store the guide content
- REPO_OWNER_ACCESS_TOKEN - OAuth token of your github user or owner of the repository where the guide content is stored. You cannot set this until after you run the application locally and authorize it with your github account as described below.
- CELERY_BROKER_URL - URL of Redis (or another broker) for handling background jobs (see *instructions for Celery on heroku* for help).
- DOMAIN - Base URL where your site will be running. This can be the URL of your Heroku deployment or localhost like *127.0.0.1:5000* or *0.0.0.0:5000*.

4.2 First-time Setup

1. Run *python run.py* or *make run_flask* if you have Make installed
2. Browse to *http://127.0.0.1:5000/login/*
3. **Login with your github account and authorize your newly created application**
    - **Login with the account you set as the REPO_OWNER**
4. Check your logs for the new token printed. This will be a CRTICAL level log message.
5. You'll need to place that token in the *REPO_OWNER_ACCESS_TOKEN* environment variable.
6. Shutdown the local flask webserver with *Ctrl-C* and run *python run.py* again

Now you can *test things from the CLI* if that's more your speed. However, there's one more step to setting up the ability to publish articles. This requires *running a celery process for background jobs*.

You can try creating a guide once you have celery running locally or your *CELERY_BROKER_URL* configured to a running Redis server.

Note now you *can* use this local server and *expose it to the Internet through a secure tunnel* if you want!

4.3 Run locally with Flask webserver

1. Follow the *first-time setup instructions*
2. Run *python run.py* or *make run_flask* and use *Ctrl-C* to stop the server.

4.4 Run locally with Gunicorn webserver

1. Follow the *first-time setup instructions*
2. Run *make run_gunicorn*

4.5 Setting up Celery for background processing

You already have Celery installed if you used the requirements.txt file. However, now you need Redis running to process background jobs from Celery and fully enable publishing articles.

Setting up Redis locally is outside the scope of this document. You can refer to the Redis documentation for that. However, you can easily setup Redis on Heroku by following the *these instructions*.

CHAPTER 5

Testing

5.1 Testing without browser

You can test a lot of the functionality of the application without a web browser. In general, much of the interaction with the Github API can be used directly from the command-line. To do this run the following:

python manage.py shell

Now you have access to the entire application. To test a Github API response try the following:

```
from pskb_website import remote
remote.read_user_from_github(username='octocat')
```

You should now see the description of the famous Github octocat user!

5.2 Adding tests

New tests can be added under a **test** directory in the appropriate package. The convention right now is to name the file as **test_*.py**

5.3 Running tests locally

To run tests locally, execute the following command from project root:

python py.test

This will find and run all tests in the current working directory.

CHAPTER 6

Github Application Setup

We make heavy use of the Github API since all of the persistent storage is a Git repository. So, you'll need to register your own Github OAuth token to have the CMS make requests to the Github API. The following steps will walk you through that setup process on github.com for an application running locally.

6.1 Create a repository for guides

First you'll need a new repository for all your content. This can be an empty repository at this point.

6.2 Registering a Github Application

1. Login to github.com
2. Go to the OAuth applications for developers section and click the register new application button
3. **Set the Authorization callback URL to *http://127.0.0.1:5000/github/authorized***
    - You can also use *http://0.0.0.0:5000* if you're running locally with the *heroku local* command.
    - This is the URL Github will sent requests back to once a user has allowed your application to access their account.
4. You can fill out the other details as you see fit. The callback URL is the most important. Now click register.
5. Copy the Client ID and Client Secret on your newly created Github application. You'll need these to continue the *installation*.

CHAPTER 7

Layout of content repository

The CMS expects a specific layout for the content repository, but you don't have to create the structure manually. You can start with an empty repository and the structure will fill itself out as guides are added. However, it's useful to understand how the content repository is structured and there are a few 'static' pages you have to create yourself. Below is the basic layout from a high-level view. You can also see a working example here.

7.1 Layout components

```
|---- faq.md
|---- published.md
|---+ published
|----   + c-c++
|----   + ruby-ruby-on-rails
|----   + python
|----   +   + guide-1
|----   +       article.md
|----   +       details.json
|
|---- in-review.md
|---+ in-review
|----   + c-c++
|----   +   + guide-2
|----   +       article.md
|----   +       details.json
|----   + ruby-ruby-on-rails
|----   + python
|
|---- draft.md
|---+ draft
|----   + c-c++
|----   + ruby-ruby-on-rails
|----   +   + guide-3
|----   +       article.md
```

(continues on next page)

(continued from previous page)

```
|----   +       details.json
|----   + python
|
|---- redirects.md
```

The layout consists of a the following 'types' of files/objects:

7.1.1 Page

A page is just a markdown file at the top-level of the repository. Currently there's one page being used called *faq.md*. The markdown for this page will automatically be rendered at the */faq.md* URL.

7.1.2 redirects.md

This file contains mapping of old guide URLs to new URLs. The purpose of this file is to accomodate changing guide titles/paths and maintaining old URLs with temporary 301 redirects. The format of this file is '<*old_url*> <*new_url*>' or '- <*old_urL*> <*new_url*>' i.e space separated and as an optional markdown list item. Keep in mind the URLs must be fully formed including the domain otherwise the redirect will be based on the current domain.

This file is optional and must be manually created.

7.1.3 Guide Listing

The guide listing files are meant to be an easy way to read the listing of the guides in the various *publish workflow stages*, published, draft, or in-review. These files make it much faster to render the contents of the */* and */in-review/* URLs. Currently there is no other persistent storage other than the Github repository. So, these files aggregate the base-essentials of a list of guides into a single file. This way listing guides only results in 1 Github API call instead of several to search the entire repository.

The three listing files currently used are *published.md* for the listing of published files appearing on the homepage, *in-review.md* for the listing of files appearing on the */in-review/* page, and *draft.md*. The *draft.md* file is not currently used by the reference website implementation. This is because guides in the draft stage are considered *private* by the web application. However, all guides are easily visible on github.com. So, the *draft.md* file provides an easy way to browse the draft guides solely for the github.com repository view.

Listing file structure

7.1.4 Guide directory

A guide consists of a directory named after the 'slug-ifed' version of the guide title. This directory consists of two files, 1 for the content and 1 for the metadata.

7.1.5 article.md

This file is the raw content in the markdown format.

7.1.6 details.json

This is the metadata for the guide in the json format. We chose JSON because it's fairly readable and easy to use withlots of languages.

Why two files?

Metadata is necessary for computers, not for humans. We want guide data to easily render great everywhere including the CMS front-end, Github.com, and even in your own text editor. This isn't very easy to achieve if you have to hide your metadata somewhere in the same file.

Another bonus is we can modify the metadata indpendently of the content. This allows for easier reading of the history for the most important part of the guide, the content.

7.2 Branches

Branches are currently used for suggested 'edits' to guides by the community editors. The branches are named to match the editor's login, stack, and title of the guide.

Each time a user edits an existing users' guide a branch is created (or updated). You can easily use Github's compare functionality to see the edits a particular user is suggesting.

You can try out the compare feature by going here. You can also append a login name to the end of this url *https://github.com/pluralsight/guides/compare/* to see that users' suggestions.

7.2.1 Why not forks?

Forks are great, but we decided to use branches for tracking user suggestions. We're pushing for simplicity from the start so using branches is definitely an experiment. However, there are several benefits of using branches instead of forks:

Pros of branches

- **Creating branches is synchronous via the Github API.**
  - Forks via the Github API is asynchronous. Doing anything asynchronous is more difficult because it requires a queue, etc.
- Prevents forking potentially large repository of unrelated guides to users' account
  - Editors will most likely be editing a single guide at a time. So, it's seems overkill to fork a repository full of guides when the user is only trying to edit one.
- Ideally we'd like to request as few permissions from users' github accounts as possible. Forking requires the 'public_repo' scope. However, we can create branches using our own repository and add the user as the 'commit author' on the changes. This workflow doesn't require the 'public_repo' scope.

  Currently we're requesting the 'public_repo' scope because that enables us to star public repositories, which we anticipate using. So, this point is somewhat irrelevant. However, it's something to consider regardless.

Ultimately we can move to use forking in the future if branching becomes limited or poses problems that forking would solve. Again, the major driver here is simplicity.

7.2.2 Why not Gists?

We heavily considered using Github gists for the guides. However, it's not possible to create gist for one user in another users' account. The CMS github user cannot create a gist in a single gist account **and** maintain the original author.

This is a problem because we're striving to give all credit to original authors and editors when making changes via the Github API. This allows any contributions users to flow back to their account. This means every change your make to a guide counts towards you total Github contributions.

The other issue with gists is tracking. We could solve the contribution problem by creating gists in every users' account. However, then the CMS would need external persistent storage to track all the gists. Also, users would not be able to easily browse all the guides in a single location on github.com.

CHAPTER 8

Deployment

Currently the application has only been deployed using Heroku but there are no reasons it cannot be deployed to any hosting platform or server that supports the Flask framework.

8.1 Heroku

Heroku has a good guide for Python apps that gives a nice overview of the concepts you'll need to know to get going, but some of the specifics for this setup are slightly different.

Real Python also has a great guide on setting up a basic flask app on Heroku.

The following steps assume you have the basic Heroku toolbelt installed.

1. **Create Heroku app**
    - *heroku create [name]*
    - You can specify a name but it must be unique. You can also leave it blank and Heroku will create a unique name for you.
2. **Add git remote for your app**
    - *git remote add heroku git@heroku.com:<name>.git* where <name> is the name of your Heroku app from step 1.
3. **Setup Heroku config**
    - See example_config.py for a listing of the environment variables that must be setup in your Heroku config.
    - You'll have to wait to setup the *REPO_OWNER_TOKEN* variable until the application is fully running.
    - **Do not forget to set HEROKU=1 in the heroku environment variables!**
    - **You can set Heroku config variables with the following syntax:**
        - *heroku config:set REPO_NAME=<name>*

  - Or something like the following if you have multiple remotes for Heroku
  - *heroku config:set REPO_NAME=<name> –app pro*
  - *heroku config:set REPO_NAME=<name> –app stage*

4. *Setup Redis add-on for background jobs <celery_on_heroku>*
5. **Deploy changes**
    - *git push heroku master*
    - Or something like the following if you have multiple remotes for Heroku
    - *git push stage master* where <stage> is remote name for Heroku and master is local branch you want to push.
    - **Make sure your changes are committed locally first!**
6. Go to your *heroku dashboard settings <https://dashboard.heroku.com/>* resources for your app and verify the worker task is running.
7. **Change the callback URL for your github application to the heroku URL**
    - Typically something like *http://<app_name.herokuapp.com>/github/authorized*
8. **Visit your running heroku application in the browser**
    - URL will be something like *http://<app_name.herokuapp.com>/*
9. **Login with your github account and authorize your newly created application**
    - **Login with the account you set as the REPO_OWNER**
10. Check your logs for the new token printed. This will be a CRTICAL level log message.
11. Set the printed token equal to the *REPO_OWNER_ACCESS_TOKEN* environment variable * *heroku config:set REPO_OWNER_ACCESS_TOKEN=<token>*

By default the application will be served up by Gunicorn.

You can slightly improve your performance on Heroku by using setting the *WEB_CONCURRENCY* environment variable, which gunicorn automatically honors. You can set that variable with the following command:

- *heroku config:set WEB_CONCURRENCY=3*

You'll want to set this to something suitable for the size of your Heroku dyno and the memory requirements of your the flask application.

8.1.1 Run application locally with Heroku Procfile

You'll need to complete the setup below for getting things running on Heroku before doing this, or at least setting up your Heroku environment variables as described below. Then:

1. Run *heroku config –app <app_name>* to see all the configuration
2. Copy all the these configuration values into a file with the *key=value* format instead of *key:value* which is the output of the Heroku command.
3. Change the callback URL on your github application to *http://0.0.0.0:5000/*
4. Run *heroku local –env <file_from_step_2>*

Useful Heroku add-ons

1. Papertrail

- Provides bigger log for debugging issues and enables easy searching
- Install the CLI tools for Papertrail if you prefer using the CLI over their website
- Below are a few useful search queries:

Description	Query
All app output along with heroku routing requests	(“app/web” OR “heroku/router”) -“newrelic”
All output minus Heroku stats	-“newrelic.core.agent” or -“newrelic.core.data_collector” or -“sample#memory_total” or -“sample#load_avg_1m” or -“sample#active-connections”
Only web app output	“app/web” -“newrelic”
Github API rate usage	“core remaining:”
Heroku scheduled task output	program:scheduler
Exceptions from Celery tasks	CalledProcessError
Celery output worker	-(dyno= OR exiting OR Booting OR Autorestarting OR State changed)

2. New Relic

- Excellent performance analysis tool

8.1.2 Redis for background tasks

- Run *heroku addons:create heroku-redis:hobby-dev* to add the free Redis add-on
- This automatically sets up your *REDIS_URL* environment variable.
- Now run *heroku config –app <heroku_app_name>* to see the value of *REDIS_URL*
- **Copy that value to a new environment variable on heroku and set it like this:**
  - *heroku config:set CELERY_BROKER_URL=<REDIS_URL>*
- We don’t set *CELERY_BROKER_URL* directly equal to *REDIS_URL* so that you’re free to setup Celery with whatever broker you choose.

Adding Redis caching on Heroku

1. Determine if you want to use a caching addon or redis addon.
   - This application has been tested with the redis cloud addon for caching data from the Github API.
   - **Redis was chosen for the following reasons:**
     - Cache value larger than 1MB (for large articles)
     - Use the same service for other things later instead of just caching
2. **Add your addon**
   - *heroku addons:create rediscloud:30 –app <app_name>*

3. The application will automatically start caching if you used the redis cloud addon described above. You can use a different Redis caching add-on, but you'll need to change the setup of the caching layer in *cache.py* appropriately.
4. See docs related to using Python with redis on Heroku

8.2 Deploying with local instance

Using this deployment method is only recommended for testing. However, often times we've noticed this method is effective for testing locally and can be faster than using 'localhost' with your Github API callbacks.

You can also 'deploy' the application running simply on *localhost* and expose your *localhost* port through a secure tunnel using ngrok. Ngrok is recommended directly by Github for testing Github webhooks. It's also useful if you have everything running locally and want to get quick feedback from testers, etc. without having to setup Heroku or another hosting machine.

1. Download ngrok
2. Run locally using *one of the available methods*
3. Run ngrok and take note of the unique *Forwarding* URL
4. Set this base URL in your Github application as described in the *Github setup*

Now anyone can go to the ngrok URL and they'll get a secure tunnel to your local machine for testing!

8.3 Setting Featured Guide

By default, the featured guide is stored in an environment variable called *FEATURED_GUIDE*. This environment variable can be 1 of 2 types of values:

1. JSON-ified tuple of (title, stack)
2. String of title

Version 1 is more *correct* since guides can have duplicate titles but not duplicate titles **and** stack. However, it's easier to use version 2 because it's a simple string. Therefore, you can use whichever suits your situation, if you don't think you'll have duplicate titles then version 2 is preferred.

8.3.1 Using environment variable

This environment variable must be set in a way that will persist across all running instances of the application. You can do this with the Heroku CLI or admin panel, if you're running on Heroku.

8.3.2 Using Redis

A better solution for managing the featured guide is to use Redis. The CMS will automatically use a single key in the 'caching' Redis database mentioned above if you're using the *REDISCLOUD_URL* setup. So, there's no need to worry about this if you are using the standard caching setup with *REDISCLOUD_URL*.

The CMS will automatically use version 1 of the *FEATURED_GUIDE* variable when using Redis so you don't have to worry about duplicate titles.

You will not be able to set the featured guide via the CMS UI if you're not using Redis to store the featured guide. This is because setting an environment variable via the application itself is unreliable if you're running multiple instances of the application on multiple dynos or servers.

CHAPTER 9

Publish Workflow

The publish workflow of this CMS consists of the following 3 *stages*. The stage determines who can see the guide as well as where the guide appears on the website front-end.

9.1 Draft

The initial stage where all guides start out in. Guides in this stage are not visible by anyone other than the original author. [1]

9.2 In-review

The second stage were guides go that are ready for community editing help. Any user can mark their guide as 'in-review' from dropdown at the bottom of the guide page.

Guides should only be marked as 'in-review' when they are complete and ready editing help.

Please don't mark partially completed guides as in-review. This will necessarily waste community editors time reviewing guides that are not completed.

Guides marked as 'in-review' will show up on the 'Review' page.

9.3 Published

The final stage for fully edited articles is published. This is the stage where the community editors have decided a guide is ready for the world to see. Only community editors can move a guide into the published stage.

Published articles will be available on the homepage of the site.

CHAPTER 10

Merging guide changes

Hopefully you'll be getting lots of suggestions from readers on how to improve the guides or fix bugs in the code. This page describes the process of merging those changes into your *master* branch so everyone can see the results of this collaborative process.

Currently merging changes to guides' is a manual process handled via the github.com website or locally with *git*. This is because merging suggestions needs to be verified by an editor and/or the original guide author. Here's the normal process after a user has suggested a change via the CMS website:

10.1 Simple merges with github.com

1. **Create a pull request on github.com for the branch**
    - Browse the branches of the content repository and click the 'New pull request' button associated with the branch you want to integrate.
2. Review the changes in the pull request on github.com
3. You can automatically merge the changes via github.com if the changes merge cleanly and you want **all** of the changes.
4. **Edit the *details.json* file for the guide you just merged changes into and remove the branch you just integrated.**
    - This can also be done via github.com by browsing to the *details.json* file and using the 'edit' button.
5. **Delete the branch you just merged on github.com**
    - Browse the branches again and click the trash can icon next to the branch

10.2 Complex merges aka the hacker way

Often times you'll only want part of the changes, or you want to handle any conflicts. This is a more involved process, which we recommend using the command-line Git interface. You can also use any Git GUI you prefer, but we're

describing the command-line approach since it's the most universal.

10.2.1 Integrating all changes from branch

1. **Clone the content repository locally**
    - Use *git clone <url>* where *<url>* is the URL for the repository from the main github repository page.
2. **Make sure all the remote branches are up to date**
    - Run *git fetch origin*
3. **Checkout the remote branch you want to integrate**
    - Run *git checkout -b <branch_name> origin/<branch_name>* where *<branch_name>* is the name of the branch you want to integrate. This is typically the username of the github user who's suggesting the changes.
4. **Merge the master branch to make sure the branch only introduces new changes**
    - Run *git merge master*
    - Fix any conflicts and commit the merge
5. **Switch back to the master branch and merge the branch**
    - Run *git checkout master*
    - Run *git merge <branch_name>*
6. Edit the *details.json* file for the guide you just merged changes into and remove the branch you just integrated.
    - You can do this directly by editing the *details.json* file or using github.com by by browsing to the *details.json* file and using the 'edit' button.
    - **Be careful to remove any trailing commas from the branches list if you remove the last branch. Remember, this file must be valid JSON syntax!**
7. **Delete the branch you just merged on github.com**
    - Run *git push origin :<branch_name>* to remove the branch from github.com. You can also do this via github.com by browsing the branches again and clicking the trash can icon next to the branch.
8. **Push the changes to github.com**
    - Run *git push origin master*

10.2.2 Integrating some changes from branch

1. **Clone the content repository locally**
    - Use *git clone <url>* where *<url>* is the URL for the repository from the main github repository page.
2. **Look at the commit(s) you want to integrate a portion of**
    - Run *git log -p <sha>* to see the changes.
    - You can use *git diff -b <sha>..<prev_sha>* to see the changes without any whitespace and/or line-ending changes.
3. Manually apply the changes you want to the master branch.
4. **Commit the changes as the original user to make sure they get credit**

- Copy the 'Author:' line from the original commit you're integrating. See output of *git log -p <sha>* from step 2.
- Add the changes to staging with *git add <filename>*
- Finally, commit the changes as the original author with *git commit –author=<author_info>* where *<author_info>* is the information for the original author.

5. Edit the *details.json* file for the guide you just merged changes into and remove the branch you just integrated.
    - You can do this directly by editing the *details.json* file or using github.com by by browsing to the *details.json* file and using the 'edit' button.
    - **Be careful to remove any trailing commas from the branches list if you remove the last branch. Remember, this file must be valid JSON syntax!**
6. **Delete the branch you just merged on github.com**
    - Run *git push origin :<branch_name>* to remove the branch from github.com. You can also do this via github.com by browsing the branches again and clicking the trash can icon next to the branch.
7. **Push the changes to github.com**
    - Run *git push origin master*

10.3 Easier visualizing of complex diffs

Often times prose is harder to diff than code because the length of a line can be very long. For example, it's common for an entire paragraph to be a single line in prose whereas software is usually broken up into small lines with hard linebreaks.

This means a diff for prose could show a large change when in reality on a few words were changed. The diff tools on github.com and *git* can help here if you know the right options to use.

10.3.1 Github.com

Github.com defaults to 'source diff view, but you can change this in the top-right hand corner of any commit page. Try clicking the 'rich diff' icon next to the 'view' button for a different view.

10.3.2 Git

First, try using *git log –word-diff=color -p* to see diffs. Another trick is to find the two adjacent commits on a file and do something like the *git diff –word-diff=color d98909743b32df2f44e835162f50e5b6b7f92c1c..8bc2725698b84d95014b0124c141a08b1946718 in-review/ruby-ruby-on-rails/handling-file-upload-using-ruby-on-rails-5-api/article.md*

You can get the two adjacent commits for a file by running *git log –follow <path_to_file>*.

CHAPTER 11

Github API usage

The CMS heavily uses the Github API. All of the raw API interaction takes places in `pskb_website/remote.py`.

11.1 Logging API Rate Limits

The CMS uses authenticated API requests to ensure a higher rate limit, which at the time of this writing is 5000 requests/hour. This is sufficient if caching and conditional requests are used.

It's worth noting that the limit is per user and per application. Therefore, the CMS can make 5000 API requests/hour on behalf of a user. However, not all requests can be made with a specific user. For example, all requests to commit data to the content repository use the *REPO_OWNER* Github API account. This is necessary because regular users do not have commit rights to the content repository. The requests using the *REPO_OWNER* Github API account are:

- Reading guides for a non-logged in user
- Committing guides to any branch
- Uploaded images for a guide

The *REPO_OWNER* account is the only account reasonably affected by the rate limiting because typical usage will not lead to a logged-in user reading thousands of guides in an hour.

It can be useful to monitor your usage since there's an upper limit. You can log the CMS' Github API usage with the *bin/rate_limit_watcher.py* script. There are a few ways to automate this data collection.

11.1.1 Heroku Scheduler

You can use the Heroku Scheduler add on to run *bin/rate_limit_watcher.py*. Just add this add-on to your account and set the add-on to run *bin/rate_limit_watcher.py* with your own arguments.

11.1.2 New Relic Insights

You can also graph your API usage overtime by using Custom Events from New Relic. To do this you'll need to configure a few environment variables for your setup:

- *NEW_RELIC_ACCT_ID* - Your New Relic account ID
- *NEW_RELIC_INSIGHTS_API_KEY* - Your New Relic Insights API Key

You can get help finding these values by using the official New Relic docs.

Finally, run *bin/rate_limit_watcher.py –report-to-new-relic* to log your usage to New Relic.

We're using the New Relic Insights API instead of New Relic custom metrics and custom events because the timestamps of the Github API data is not that critical. The API limits do not need to be synchonized with all the other New Relic data. In addition, the Insights API is easier to use from a script that's not embedded in the main WSGI application.

CHAPTER 12

Github Webhooks

The CMS uses Github webhooks to get notifications of changes happening on Github.com. These are not required, but they are useful if you're using the built-in *Caching*. These webhooks can clear the cache when something changes on Github.com directly so that the CMS is always using the most up-to-date guide information.

12.1 Configuring Push Events

This event is used to clear the cache of a guide when it's changed via a commit from the Github API and/or Github.com

1. Go to the settings area of your content repository where all of your guides are stored and click on 'Webhooks & services'.
2. Click 'Add webhook'
3. Set the *Payload URL* to *<your_domain>/github_push*
4. The *Content type* should be *application/json*
5. Setup your *secret* field according to Github's instructions or use the same *secret* you configured for Delete Events if you configured those first.
6. Only subscribe to the push event
7. Make sure the webhook is marked as **active**
8. Click 'Add webhook'
9. Add a new environment variable to your you instance of the CMS called WEBHOOK_SECRET and set it to the value you used in step 5.

12.2 Configuring Delete Events

This event is used to clean up the list of branches associated with a guide.

1. Go to the settings area of your content repository where all of your guides are stored and click on 'Webhooks & services'.
2. Click 'Add webhook'
3. Set the *Payload URL* to *<your_domain>/github_delete*
4. The *Content type* should be *application/json*
5. Setup your *secret* field according to Github's instructions or use the same *secret* you configured for Push Events above.
6. **Only subscript to the delete event**
    - Click 'Let me select individual events'
7. Make sure the webhook is marked as **active**
8. Click 'Add webhook'
9. Add a new environment variable to your you instance of the CMS called WEBHOOK_SECRET and set it to the value you used in step 5.

The same secret is used for all webhooks for simplicity and to cutdown on the number of environment variables needed.

12.3 Testing

Github has great documentation on testing webhooks and a solution for testing locally.

CHAPTER 13

Frequently Asked Questions

13.1 How do I change the URL for an existing guide?

First, you must understand how a URL for a guide is created. The URLs for a guide are based on the following components:

- Stack
- Title
- publish status

The stack and title are the slug versions of the stack and title, which are generated by the `utils.slugify()` function. In short, all characters that are not ascii letters or numbers are translated to -, which results in readable, SEO-friendly URLs.

The publish status is an optional query string to hint to the CMS which folder to read the guide from. The CMS will search all possible publish statuses to find a guide if this is missing.

This scheme causes a problem if you want to change the stack or title of an existing guide, namely the old URL will not work. Therefore, it's possible to setup 301 redirects for guides in the *redirects file*.

Finally, to change a URL for an existing guide you need to make an entry in the *redirects file* with the old URL that you want to replace followed by the new URL and update the URL in the associated *guide listing file*.

13.2 How do I change the title for an existing guide?

This process is a bit involved since the URL for a guide is based on the URL (see previous question). To do this you need to do a bit of manual work with the underlying git repository:

1. Clone your CMS content repository locally
2. Edit the *title* attribute in the *details.json* file for the guide you want to change
3. Save the file

4. **Determine new slug for the new title**
    - You can do this by replacing all non-ascii letters or numbers with the - character or by running the `utils.slugify()` function on your new title.
5. **Run *git mv <curr_path_to_guide> <new_path_to_guide>***
    - The <new_path_to_guide> should include your new title
6. **Make an entry in the *redirects file*.**
    - Make sure to use the new title slug in the new URL.
7. Edit the URL for your guide in the *guide listing file* your guide belongs to
8. Commit these changes to your CMS content repository and push to github.com.
9. You can manually flush your redis cache or wait a few minutes for things to automatically refresh.

13.3 How do I change the stack for an existing guide?

1. Change the stack of your guide in the CMS web interface and save it
2. Clone your CMS content repository locally
3. **Determine new slug for the new stack**
    - You can do this by replacing all non-ascii letters or numbers with the - character or by running the `utils.slugify()` function on your new stack.
4. **Make an entry in the *redirects file*.**
    - Make sure to use the new stack slug in the new URL.
5. Edit the URL for your guide in the *guide listing file* your guide belongs to
6. Commit these changes to your CMS content repository and push to github.com.
7. You can manually flush your redis cache or wait a few minutes for things to automatically refresh.

CHAPTER 14

Views

Main views of PSKB app

pskb_website.views.**all_authors**(**args*, ***kwargs*)
Get listing of all authors who've contributed a guide

pskb_website.views.**article_view**(*stack*, *title*)
Find article with given stack/stack combination and display it

Note all publish statuses are searched and the first one found is returned. This allows us to keep the same URL through the publish workflow process since the status is only a 'hint' and query string.

By default, the statuses are searched in the order of importance: published, in-review, and finally draft.

GET parameters used:

- **status: Hint on what publish status to search for FIRST**
  - Default is 'published' which makes the published articles have clean URLs without any query string.
- **branch: Branch of article to display**
  - Default is master

pskb_website.views.**authorized**()
URL for Github auth callback

pskb_website.views.**change_publish_status**(**args*, ***kwargs*)
Publish or unpublish article via POST

pskb_website.views.**contest**()
Contest page

pskb_website.views.**contributors**()
Contributors page

pskb_website.views.**delete**(**args*, ***kwargs*)
Delete POST page

pskb_website.views.**faq**()
FAQ page

pskb_website.views.**get_sitemap**()
sitemap

pskb_website.views.**get_social_redirect_url**(*article*, *share_domain*)
Get social redirect url for po.st to enable all counts to follow us regardless of where we're hosted.

pskb_website.views.**github_login**()
Callback for github oauth

pskb_website.views.**in_review**()
In review page

pskb_website.views.**index**()
Homepage

pskb_website.views.**internal_error**(*error=None*)
Unknown error page

pskb_website.views.**login**()
Login page

pskb_website.views.**logout**(**args*, ***kwargs*)
Logout page

pskb_website.views.**missing_article**(*requested_url=None*, *stack=None*, *title=None*, *branch=None*)
Handle missing articles by checking if URL is should be 301 redirect or showing published articles in the URL is truly bad

pskb_website.views.**my_drafts**(**args*, ***kwargs*)
Users drafts

pskb_website.views.**not_found**(*error=None*)
Not found error page

pskb_website.views.**partner**(*article_path*)
URL for articles from hackhands blog – these articles are not editable.

pskb_website.views.**partner_import**(**args*, ***kwargs*)
Special 'hidden' URL to import articles to secondary repo

pskb_website.views.**render_article_list_view**(*status*)
Render list of articles with given status

Parameters **status** – PUBLISHED, IN_REVIEW, or DRAFT

pskb_website.views.**render_article_view**(*request_obj*, *article*, *only_visible_by_user=None*)
Render article view

Parameters

- **request_obj** – Request object
- **article** – Article object to render view for
- **branch** – Branch of article to read
- **only_visible_by_user** – Name of user that is allowed to view article or None to allow anyone to read it

pskb_website.views.**render_published_articles**(*status_code=200*)
Render published article listing and featured article

This is extracted into a stand-alone function so we can render this in multiple locations without redirects which could hurt SEO and usability.

pskb_website.views.**review**(*title*)
This URL only exists for legacy reasons so try to find the article where it is in the new scheme and return 301 to indicate moved.

pskb_website.views.**set_featured_title**(**args*, ***kwargs*)
Form POST to update featured title

pskb_website.views.**strip_subfolder**(*url*)
Strip off the subfolder if it exists so we always use the exact same share url for saving counts.

pskb_website.views.**subscribe**()
Subscribe POST page

pskb_website.views.**sync_listing**(**args*, ***kwargs*)
Sync listing page

pskb_website.views.**template_globals**()
Global variables available to all responses

pskb_website.views.**url_components**(*url*)
Get URL path components as a list (leading slash is removed!)

pskb_website.views.**url_for_domain**(*url*, *domain=None*)
Get url for domain from environment

pskb_website.views.**user_profile**(*author_name*)
Profile page

pskb_website.views.**write**(**args*, ***kwargs*)
Editor page

CHAPTER 15

Model API

This API layer provides access to the higher-level objects stored in Github repositories. A single Github repository serves as the persistent storage for the CMS. All data is fetched from Github through the *Remote API* layer and turned into formal objects by this model layer.

15.1 Article

Article related model API

class pskb_website.models.article.**Article**(*title*, *author_name*, *filename='article.md'*, *repo_path=None*, *branch=u'master'*, *stacks=None*, *sha=None*, *content=None*, *external_url=None*, *image_url=None*, *author_real_name=None*)

Object representing article

contributors

List of tuples representing any 'author' i.e user who has contributed at least 1 line of text to this article. Each tuple is in the form of (name, login) where name can be None.

We use plain tuples instead of named tuples or User objects so we can easily seralize the contributors to JSON.

NOTE: This property automatically removes users set to ignore via the contributors_to_ignore() function! To get the full list use _read_contributors_from_api(remove_ignored_users=False).

static from_json(*str_*)

Create article object from json string

Parameters **str** – json string representing article

Returns Article object

full_path

Get full path to article including repo information :returns: Full path to article

heart_count
Read number of hearts for article

Returns Number of hearts

open_graph_image_url
Get full URL suitable for open graph meta tags

stack_image_url
Get path to static image for article based on stack

None will be returned for articles without a stack image FB open graph meta tags.

pskb_website.models.article.**articles_from_json**(*json_str*)
Generator to iterate through list of article objects in json format

Parameters **json_str** – JSON string

Returns Generator through article objects

pskb_website.models.article.**author_stats**(*statuses=None*)
Get number of articles for each author

Parameters

- **statuses** – List of statuses to aggregate stats for
- **statuses** – Optional status to aggregate stats for, all possible statuses are counted if None is given

Returns

Dictionary mapping author names to number of articles:

```
{author_name: [article_count, avatar_url]}
```

Note avatar_url can be None and is considered optional

pskb_website.models.article.**branch_article**(*article*, *message*, *new_content*, *author_name*, *email*, *image_url*, *author_real_name=None*)
Create branch for article with new article contents

Parameters

- **article** – Article object to branch
- **message** – Message describing article suggestions/changes
- **new_content** – New article text
- **author_name** – Name of author for article changes
- **email** – Email of author for article changes
- **image_url** – Image to use for article
- **author_real_name** – Optional real name of author, not username

Returns New article object

New branch will be named after author of changes and title

pskb_website.models.article.**branch_or_save_article**(*title*, *path*, *message*, *content*, *author_name*, *email*, *sha*, *image_url*, *repo_path=None*, *author_real_name=None*, *stacks=None*, *first_commit=None*)

Save article as original or as a branch depending on if given author is the same as original article (if it already exists)

Parameters

- **title** – Title of article
- **path** – Short path to article, not including repo or owner, or empty for a new article
- **message** – Commit message to save article with
- **content** – Content of article
- **author_name** – Name of author who wrote content
- **email** – Email address of author
- **sha** – Optional SHA of article if it already exists on github
- **branch** – Name of branch to commit file to (branch must already exist)
- **image_url** – Image to use for article
- **repo_path** – Optional repo path to save into (<owner>/<name>)
- **author_real_name** – Optional real name of author, not username
- **stacks** – Optional list of stacks to associate with article (this argument is ignored if article is branched)
- **first_commit** – SHA of first commit to save with article

Returns Article object updated, saved, or branched

pskb_website.models.article.**change_article_stack**(*orig_path*, *orig_stack*, *new_stack*, *title*, *author_name*, *email*)

Change article stack

Parameters

- **orig_path** – Current path to article without repo or owner
- **orig_stack** – Original stack
- **new_stack** – New stack
- **author_name** – Name of author who wrote article
- **email** – Email address of author

Returns New path of article or None if error

Note this function only makes changes to articles on the master branch!

pskb_website.models.article.**delete_article**(*article*, *message*, *name*, *email*)

Delete article from repository

Parameters

- **article** – Article object to remove
- **message** – Message to include as commit when removing article

- **name** – Name of user deleting article
- **email** – Email address of user deleting article

Returns True if article was successfully removed or False otherwise

This removes the article from the repository but not the history of the file.

Only original author can remove file from master branch. Articles can be removed from non-master branches only by the user who created that branch.

pskb_website.models.article.**delete_branch**(*article*, *branch_to_delete*)
Delete branch of guide and save to github

Parameters

- **article** – Article object to delete branch from
- **branch_to_delete** – Branch of guide to delete

Returns True if deleted or False otherwise

pskb_website.models.article.**find_article_by_title**(*articles*, *title*)
Search through list of article objects looking for article with given title

Parameters

- **articles** – List of article objects
- **title** – Title to search for

Returns article object or None if not found

pskb_website.models.article.**get_articles_for_author**(*author_name*, *status=None*)
Get iterator for articles from given author

Parameters

- **author_name** – Name of author to find articles for
- **status** – PUBLISHED, IN_REVIEW, DRAFT, or None to read all articles

Returns Iterator through article objects

pskb_website.models.article.**get_available_articles**(*status=None*, *repo_path=None*)
Get iterator for current article objects

Parameters

- **status** – PUBLISHED, IN_REVIEW, DRAFT, or None to read all articles
- **repo_path** – Optional repo path to read from (<owner>/<name>)

Returns Iterator through article objects

Note that article objects only have path, title, author name, and stacks filled out. You'll need to call read_article() to get full article details.

pskb_website.models.article.**get_available_articles_from_api**(*status=None*, *repo_path=None*)
Get iterator for current article objects

Parameters

- **status** – PUBLISHED, IN_REVIEW, DRAFT, or None to read all articles
- **repo_path** – Optional repo path to read from (<owner>/<name>)

Returns Iterator through article objects

Note that article objects only have path, title and author name filled out. You'll need to call read_article() to get full article details.

pskb_website.models.article.**get_public_articles_for_author**(*author_name*)
Get iterator for all public i.e. non-draft articles from given author

Parameters **author_name** – Name of author to find articles for

Returns Iterator through article objects

pskb_website.models.article.**group_articles_by_status**(*articles*)
Group articles by publish status

Parameters **articles** – Iterable of Article objects

Returns Iterable like itertools.groupby with a key as the publish_status and a list of articles for that status

pskb_website.models.article.**meta_data_path_for_article_path**(*full_path*)
Get path to meta data file for given article path

Parameters **full_path** – Article object

Returns Full path to meta data file for article

pskb_website.models.article.**parse_full_path**(*path*)
Parse full path and return tuple of details embedded in path

Parameters **path** – Full path to file including repo and owner

Returns path_details tuple

class pskb_website.models.article.**path_details**(*repo*, *filename*)

filename
Alias for field number 1

repo
Alias for field number 0

pskb_website.models.article.**read_article**(*path*, *rendered_text=False*, *branch=u'master'*, *repo_path=None*, *allow_missing=False*, *cache_timeout=7200*)
Read article

Parameters

- **path** – Short path to article, not including repo or owner
- **rendered_text** – Boolean to read rendered or raw text
- **branch** – Name of branch to read file from
- **repo_path** – Optional repo path to read from (<owner>/<name>)
- **allow_missing** – False to log warning for missing or True to allow it i.e. when you're just seeing if an article exists
- **cache_timeout** – Number of seconds to keep guide in cache if cached

Returns Article object

pskb_website.models.article.**read_article_from_metadata**(*file_details*)
Read article object from json metadata

Parameters **file_details** – remote.file_details object

Returns Article object with metadata filled out or None

Note the article contents are NOT filled out here!

pskb_website.models.article.**read_meta_data_for_article_path**(*full_path*)
Read meta data for given article path from master branch

Parameters **full_path** – Full path to article

Returns Meta-data for article as json

Always read meta data from master branch because metadata is never altered or updated in branches to keep merging simple.

pskb_website.models.article.**save_article**(*title*, *message*, *new_content*, *author_name*, *email*, *sha*, *branch=u'master'*, *image_url=None*, *repo_path=None*, *author_real_name=None*, *stacks=None*, *status=u'draft'*, *first_commit=None*)
Create or save new (original) article, not branched article

Parameters

- **title** – Title of article
- **message** – Commit message to save article with
- **content** – Content of article
- **author_name** – Name of author who wrote article
- **email** – Email address of author
- **sha** – Optional SHA of article if it already exists on github (This must be the SHA of the current version of the article that is being replaced.)
- **branch** – Name of branch to commit file to (branch must already exist)
- **image_url** – Image to use for article
- **repo_path** – Optional repo path to save into (<owner>/<name>)
- **author_real_name** – Optional real name of author, not username
- **stacks** – Optional list of stacks to associate with article
- **status** – PUBLISHED, IN_REVIEW, or DRAFT
- **first_commit** – Optional first commit of article if it already exists

Returns Article object updated or saved or None for failure

This function is not suitable for saving branched articles. The article created here will be attributed to the given author_name whereas branched articles should be created with branch_article() so the correct author information is maintained.

pskb_website.models.article.**save_article_meta_data**(*article*, *author_name=None*, *email=None*, *branch=None*, *update_branches=True*)

Parameters

- **article** – Article object
- **author_name** – Name of author who wrote article (optional)
- **email** – Email address of author (optional)

- **branch** – Optional branch to save metadata, if not given article.branch will be used
- **update_branches** – Optional boolean to update the metadata branches of the article with the given branch (True) or to save article branches as-is (False)

Returns SHA of commit or None if commit failed

Note that author_name and email can be None if the site 'admin' is changing the meta data. However, author_name and email must both exist or both be None.

pskb_website.models.article.**save_branched_article_meta_data**(*article*, *author_name*, *email*, *add_branch=True*)

Save metadata for branched article

Parameters

- **article** – Article object with branch attribute set to branch name
- **name** – Name of author who wrote branched article
- **email** – Email address of branched article author
- **add_branch** – True if article should be saved as a branch False if article should be removed as a branch

Returns SHA of commit or None if commit failed

Metadata for branched articles should be identical to the original article. This makes it easier for automatically merging changes because metadata differences won't get in the way. The author_name is the only thing useful for a branched article. However, that should already be encoded in the branch name and the commits. So, editors of original articles will get credit for helping via those mechanisms, not metadata.

pskb_website.models.article.**search_for_article**(*title*, *stacks=None*, *status=None*)

Search for an article by the title and optionally stack and status

Parameters

- **title** – Title of article to search for
- **stacks** – Optional list of stacks to search All stacks are searched if None is given
- **status** – Optional status to search for All possible statuses are searched if None is given

Returns Article object if found or None if not found

15.2 User

User related model code

class pskb_website.models.user.**User**(*name*, *login*)

Object representing user

static from_json(*str_*)

Create user object from json string

Parameters **str** – json string representing user

Returns User object

is_collaborator

Determine if user is a collaborator on repo

Parameters

- **owner** – Owner of repository defaults to REPO_OWNER config value
- **repo** – Name of repository defaults to REPO_NAME config value

pskb_website.models.user.**find_user**(*username=None*)
Find a user object with given username

Parameters **username** – Optional username to search for, if no username given the currently logged in user will be returned (if any)

Returns User object

Note the email field on the returned user object is only valid when reading the logged in user (i.e. when NOT passing a username). We cannot read email information for users who have not authenticated the application.

15.3 File

More direct wrapper around reading files from remote storage

This module serves as a way to read and parse common markdown file 'types' from the repository such as the file listings for published articles, etc.

pskb_website.models.file.**draft_article_path**()
Get path to draft article file listing

Returns Path to draft article file listing file

pskb_website.models.file.**draft_articles**(*branch=u'master'*)
Get iterator through list of draft articles from file listing

Parameters **branch** – Name of branch to save file listing to

Returns Generator to iterate through file_listing_item tuples

class pskb_website.models.file.**file_listing_item**(*title*, *url*, *author_name*, *author_real_name*, *author_img_url*, *thumbnail_url*, *stacks*)

author_img_url
Alias for field number 4

author_name
Alias for field number 2

author_real_name
Alias for field number 3

stacks
Alias for field number 6

thumbnail_url
Alias for field number 5

title
Alias for field number 0

url
Alias for field number 1

pskb_website.models.file.**get_removed_file_listing_text**(*text*, *title*)
Remove given title from file listing text and return result

Parameters **text** – Text of file listing file

Returns String of text with title removed

pskb_website.models.file.**get_updated_file_listing_text**(*text*, *article_url*, *title*, *author_url*, *author_name*, *author_img_url=None*, *thumbnail_url=None*, *stacks=None*)

Update text for new article listing

Parameters

- **text** – Text of file listing file
- **article_url** – URL to article
- **title** – Title of article to put in listing
- **author_url** – URL to author
- **author_name** – Name of author (i.e. login/username)
- **author_img_url** – Optional URL to image for author
- **thumbnail_url** – Optional URL to thumbnail image for article
- **stacks** – Optional list of stacks article belongs to

Returns String of text with article information updated

pskb_website.models.file.**in_review_article_path**()

Get path to in-review article file listing

Returns Path to in-review article file listing file

pskb_website.models.file.**in_review_articles**(*branch=u'master'*)

Get iterator through list of in-review articles from file listing

Parameters **branch** – Name of branch to save file listing to

Returns Generator to iterate through file_listing_item tuples

pskb_website.models.file.**published_article_path**()

Get path to published article file listing

Returns Path to published article file listing file

pskb_website.models.file.**published_articles**(*branch=u'master'*)

Get iterator through list of published articles from file listing

Parameters **branch** – Name of branch to save file listing to

Returns Generator to iterate through file_listing_item tuples

pskb_website.models.file.**read_file**(*path*, *rendered_text=True*, *branch=u'master'*, *use_cache=True*, *timeout=480*)

Read file contents

Parameters

- **path** – Short path to file, not including repo or owner
- **rendered_text** – Read rendered markdown text (True) or raw text (False)
- **branch** – Name of branch to read file from
- **use_cache** – Boolean to read from cache if available and save if not found in cache (use False to bypass any cache interaction, useful for very large files)

- **timeout** – Cache timeout to save contents with (in seconds) - only used if use_cache is True

Returns Text of file or None if file could not be read

pskb_website.models.file.**read_file_details**(*path*, *rendered_text=True*, *branch=u'master'*)
Read file details including SHA and contents

Parameters

- **path** – Short path to file, not including repo or owner
- **rendered_text** – Read rendered markdown text (True) or raw text (False)
- **branch** – Name of branch to read file from

Returns remote.file_details tuple or None if file is missing

pskb_website.models.file.**read_items_from_file_listing**(*text*)
Generator to yield parsed file_listing_item from text

Parameters **text** – Raw text as read from file listing file

Returns Generator to iterate through file_listing_item tuples

pskb_website.models.file.**read_redirects**(*branch=u'master'*)
Read redirects file and parse into a dictionary mapping an old url to a new url

Parameters **branch** – Branch to read redirect file from

Returns Dictionary with keys for old url and values for new url

The format of the redirect file is two URLs per line with whitespace between them:

```
http://www.xyz.com http://www.xyz.com/1
http://www.xyz.com/2 http://www.xyz.com/3
```

This means redirect http://www.xyz.com to http://www.xyz.com/1 and redirect http://www.xyz.com/2 to http://www.xyz.com/3.

Each line can start with an optional '- ', which will be ignored.

Any lines starting with a '#' or not containing two tokens is ignored.

pskb_website.models.file.**remove_article_from_listing**(*title*, *status*, *committer_name*, *committer_email*, *branch=u'master'*)
Remove article title from file listing

Parameters

- **title** – Title of article to remove from listing
- **status** – PUBLISHED, IN_REVIEW, or DRAFT
- **committer_name** – Name of user committing change
- **committer_email** – Email of user committing change
- **branch** – Name of branch to save file listing to

Returns True or False if file listing was updated

pskb_website.models.file.**sync_file_listing**(*all_articles*, *status*, *committer_name*, *committer_email*, *branch=u'master'*)
Synchronize file listing file with contents of repo

Parameters

- **all_articles** – Iterable of article objects that should be synced to listing
- **status** – PUBLISHED, IN_REVIEW, or DRAFT
- **committer_name** – Name of user committing change
- **committer_email** – Email of user committing change
- **branch** – Name of branch to save file listing to

Returns Boolean to indicate if syncing succeeded or failed

This can be a very expensive operation because it heavily calls the remote API so be careful calling this for API limits and performance. Ideally this should at least be run as some kind of background process.

pskb_website.models.file.**update_article_listing**(*article_url*, *title*, *author_url*, *author_name*, *committer_name*, *committer_email*, *author_img_url=None*, *thumbnail_url=None*, *stacks=None*, *branch=u'master'*, *status=u'draft'*)

Update article file listing with given article info

Parameters

- **article_url** – URL to article
- **title** – Title of article to put in listing
- **author_url** – URL to author
- **author_name** – Name of author (i.e. login/username)
- **committer_name** – Name of user committing change
- **committer_email** – Email of user committing change
- **author_img_url** – Optional URL to author's image
- **thumbnail_url** – Optional URL to thumbnail image for article
- **stacks** – Optional list of stacks article belongs to
- **branch** – Name of branch to save file listing to
- **status** – PUBLISHED, IN_REVIEW, or DRAFT to add article to file listing. All other file listings will also be updated to remove this article if it exists there.

Returns True or False if file listing was updated

15.4 Heart

Module to manage CRUD operations on 'heart'ing guides

15.5 Image

Save and read image files to/from github

pskb_website.models.image.**github_url_from_upload_path**(*path*, *name*, *branch='master'*)

Get URL to see raw image on github from the path the file was uploaded to

Parameters

- **path** – Path Full path file was save to github with
- **name** – Name file was saved with
- **branch** – Branch image was saved to

Returns URL to see content on github

pskb_website.models.image.**main_image_path**()
Get path to main repos images

pskb_website.models.image.**save_image**(*file_*, *extension*, *message*, *name*, *email*, *branch='master'*)
Save image to github as a commit

Parameters

- **file** – Open file object containing image
- **message** – Commit message to save image with
- **name** – Name of user committing image
- **email** – Email address of user committing image
- **branch** – Branch to save image to

Param Extension to use for saved filename

Returns Public URL to image or None if not successfully saved

15.6 Lib

Collection of shared functionality for models subpackage

pskb_website.models.lib.**contribution_stats**()
Get total and weekly contribution stats for default repository

Returns Ordered dictionary keyed by author login name and ordered by most commits this week Each value in dictionary is a dictionary of stats for that contributor

pskb_website.models.lib.**contributors_to_ignore**()
Get set of logins to ignore from all contribution stats

Returns Set of logins

pskb_website.models.lib.**to_json**(*object_*, *exclude_attrs=None*)
Return json representation of object

Parameters **exclude_attrs** – List of attributes to exclude from serialization

Returns json representation of object as a string

CHAPTER 16

Remote API

This API layer provides direct access to the Github storage layer through the remote OAuth API calls.

Main entry point for interacting with remote service APIs

pskb_website.remote.**check_rate_limit**()
Get rate limit data

Returns None in case of an error or raw rate limit request data

pskb_website.remote.**commit_file_to_github**(*path*, *message*, *content*, *name*, *email*, *sha=None*, *branch=u'master'*, *auto_encode=True*)
Save given file content to github

Parameters

- **path** – Path to file (<owner>/<repo>/<dir>/. . . /<filename>)
- **message** – Commit message to save file with
- **content** – Content of file
- **name** – Name of author who wrote file
- **email** – Email address of author
- **sha** – Optional SHA of file if it already exists on github
- **branch** – Name of branch to commit file to (branch must already exist)
- **auto_encode** – Boolean to automatically encode data as utf-8

Returns SHA of commit or None for failure

Note that name and email can be None if you want to make a commit with the REPO_OWNER. However, name and email should both exist or both be None, which is a requirement of the underlying Github API.

pskb_website.remote.**commit_image_to_github**(*path*, *message*, *file_*, *name*, *email*, *sha=None*, *branch=u'master'*)
Save given image file content to github

Parameters

- **path** – Path to file (<owner>/<repo>/<dir>/. . . /<filename>)
- **message** – Commit message to save file with
- **file** – Open file object
- **name** – Name of author who wrote file
- **email** – Email address of author
- **sha** – Optional SHA of file if it already exists on github
- **branch** – Name of branch to commit file to (branch must already exist)

Returns SHA of commit or None for failure

pskb_website.remote.**contents_url_from_path**(*path*)
Get github API url for contents of file from full path

Parameters **path** – Path to file (<owner>/<repo>/<dir>/. . . /<filename>)

Returns URL suitable for a content call with github API

pskb_website.remote.**contributor_stats**(*repo_path=None*)
Get response of /repos/<repo_path>/stats/contributors from github.com

Parameters **repo_path** – Default repo or repo path in owner/repo_name form

Returns Raw response of contributor stats from https://developer.github.com/v3/repos/statistics/#get-contributors-list-with-additions-deletions-and-commit-counts

Note the github caches contributor results so an empty list can also be returned if the data is not available yet or there is an error

pskb_website.remote.**create_branch**(*repo_path*, *name*, *sha*)
Create a new branch

Parameters

- **repo_path** – Path to repo that branch should be created from
- **name** – Name of branch to create
- **sha** – SHA to branch from

Returns True if branch was created or False if branch already exists or could not be created

pskb_website.remote.**default_repo_path**()
Get path to main repo

pskb_website.remote.**default_repo_url**()
Get URL to default repo

pskb_website.remote.**file_contributors**(*path*, *branch=u'master'*)
Get dictionary of User objects representing authors and committers to a file

Parameters

- **path** – Short-path to file (<dir>/. . . /<filename>) i.e. without repo and owner
- **base** – Name of branch to read contributors for

Returns

Dictionary of the following form:

```
{'authors': set([(name, login), (name, login), ...]),
```

'committers': set([(name, login), (name, login), ...])}

Note that name can be None if user doesn't have their full name setup on github account.

class pskb_website.remote.**file_details**(*path*, *branch*, *sha*, *last_updated*, *url*, *text*)

branch
Alias for field number 1

last_updated
Alias for field number 3

path
Alias for field number 0

sha
Alias for field number 2

text
Alias for field number 5

url
Alias for field number 4

pskb_website.remote.**file_details_from_github**(*path*, *branch=u'master'*, *allow_404=False*)

Get file details from github

Parameters

- **path** – Path to file (<owner>/<repo>/<dir>/.../<filename>)
- **branch** – Name of branch to read file from
- **allow_404** – False to log warning for 404 or True to allow it i.e. when you're just seeing if a file already exists

Returns file_details namedtuple or None for error

pskb_website.remote.**files_from_github**(*repo*, *filename*, *limit=None*)

Iterate through files with a specific name from github

Parameters

- **repo** – Path to repo to read files from
- **filename** – Name of filename to search for recursively
- **limit** – Optional limit of the number of files to return

Returns Iterator through file_details tuples

pskb_website.remote.**get_github_oauth_token**()

Read github token from session

pskb_website.remote.**log_error**(*message*, *url*, *resp*, ***kwargs*)

Log an error from a request and include URL, response status, response data and additional error information

Params message Message to log

Parameters

- **url** – URL of request that failed
- **resp** – Response object holding failure information

- **kwargs** – Additional data to put in error message

Returns None

pskb_website.remote.**merge_branch**(*repo_path*, *base*, *head*, *message*)
Attempt merge between two branches

Parameters

- **repo_path** – Path to repo <owner>/<repo_name>
- **base** – Name of the base branch that the head will be merged into
- **head** – The name of the head to merge into base
- **message** – Commit message to use for merge

Returns True if merge was successful False otherwise

pskb_website.remote.**primary_github_email_of_logged_in**()
Get primary email address of logged in user

pskb_website.remote.**read_branch**(*repo_path*, *name*)
Read branch and get HEAD sha

Parameters

- **repo_path** – Path to repo of branch
- **name** – Name of branch to read

Returns SHA of HEAD or None if branch is not found

pskb_website.remote.**read_file_from_github**(*path*, *branch=u'master'*, *rendered_text=True*, *allow_404=False*)
Get rendered file text from github API

Parameters

- **path** – Path to file (<owner>/<repo>/<dir>/. . . /<filename>)
- **branch** – Name of branch to read file from
- **rendered_text** – Return rendered or raw text
- **allow_404** – False to log warning for 404 or True to allow it i.e. when you're just seeing if a file already exists

Returns file_details namedtuple or None if error

Note when requesting rendered text there will be no SHA or last_updated data available. This is a restriction from the github API (https://developer.github.com/v3/media/#repository-contents) Requesting file 'details' like SHA and rendered text are 2 API calls. Therefore, if you want all of that information you should call this function twice, once with rendered_text=True and one with rendered_text=False and combine the information yourself.

pskb_website.remote.**read_repo_collaborators_from_github**(*owner=None*, *repo=None*)
Generator for collaborator login/usernames for a given repo

Parameters

- **owner** – Owner of repository defaults to REPO_OWNER config value
- **repo** – Name of repository defaults to REPO_NAME config value

Returns Generator through login names

pskb_website.remote.**read_user_from_github**(*username=None*)
Read user information from github

Parameters **username** – Optional username to search for, if no username given the currently logged in user will be returned (if any)

Returns Dict of information from github API call

pskb_website.remote.**remove_file_from_github**(*path*, *message*, *name*, *email*, *branch*)
Remove file from github repo

Parameters

- **path** – Path to file (<owner>/<repo>/<dir>/. . . /<filename>)
- **message** – Commit message to remove file with
- **name** – Name of author who wrote file
- **email** – Email address of author
- **branch** – Name of branch to delete file from

Returns True if file was removed or False otherwise

Note the file is only removed from the repository, not the history of the file.

pskb_website.remote.**rendered_markdown_from_github**(*path*, *branch=u'master'*, *allow_404=False*)
Get rendered markdown file text from github API

Parameters

- **path** – Path to file (<owner>/<repo>/<dir>/. . . /<filename.md>)
- **branch** – Name of branch to read file from
- **allow_404** – False to log warning for 404 or True to allow it i.e. when you're just seeing if a file already exists

Returns HTML file text

pskb_website.remote.**repo_sha_from_github**(*repo*, *branch=u'master'*)
Get sha from head of given repo

Parameters

- **repo** – Path to repo (owner/repo_name)
- **branch** – Name of branch to get sha for

Returns Sha of branch

pskb_website.remote.**split_full_file_path**(*path*)
Split full file path into owner, repo, and file_path

Parameters **path** – Path to file (<owner>/<repo>/<dir>/. . . /<filename>)

Returns (owner, repo, file_path)

pskb_website.remote.**update_branch**(*repo_path*, *name*, *sha*)
Update branch to new commit SHA

Parameters

- **repo_path** – Path to repo that branch should be created from
- **name** – Name of branch to create

- **sha** – SHA to branch from

Returns True if branch was update or False if branch could not be updated

CHAPTER 17

Utility functions

The following functions are for general use.

Generic functions for global use

pskb_website.utils.**configure_redis_from_url**(*url*)
Create and configure a redis instance from the given url

Parameters **url** – URL encoded in the popular *scheme://netloc/path;parameters?query#fragment* that urlparse.urlparse supports

Returns configured redis.Redis object or None if there was a problem

pskb_website.utils.**slugify**(*text*, *delim=u'-'*)
Generates an slightly worse ASCII-only slug.

pskb_website.utils.**slugify_stack**(*stack*)
Generates an ASCII-only slug version of the stack

CHAPTER 18

Release Process

The following document explains the manual release process.

1. **Prepare release notes**
    - Run *git tag* to see release names
    - Run *git log <prev_tag>..HEAD* to see all changes since last release.
    - We typically only pick out the large changes that will affect users or developers.
2. Add notes to *CHANGELOG* file in restructed text format
3. **Pick a release name**
    - We're loosely using semantic versioning.
4. **Create a tag locally for the release name**
    - *git tag <name>*
5. **Push tag to github.com**
    - *git push origin <name>*
6. **Add release notes to github.com**
    - Click 'releases' tab on main github project page
    - Click 'tags'
    - Click 'Add release notes'
    - Fill out info in markdown!

    ** Yes, it's annoying we have release notes in rst and markdown.** We could potentially automate this or remove the redundancy in the future. Pull Requests for this would be accepted. :)

CHAPTER 19

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

S

T

U

W