COS Development

Release 1.0

Sep 27, 2017

Contents

1 Process 3

2 Style Guides 9

3 OSF Development 21

4 Getting Help 55

5 Contributing To These Docs 57

Welcome to the COS development documentation page. Here you'll find style guides and best practices for working on COS projects.

NOTE: These docs are a work in progress. Expect them to change often!

CHAPTER 1

Process

1.1 Version Control

1.1.1 General Guidelines

We use git for version control. Some general guidelines:

- Use Vincent Driessen's Successful Git Branching Model (aka git-flow)
- In your **feature** branches, pull from `develop` frequently.
- **DO NOT** merge `develop` into **hotfix** branches.
- Follow the *Git style guide*.
- Package and app maintainers: Use semantic versioning.

1.1.2 Useful Tools

- SourceTree - Mac OS X; GUI.
- gitflow - Cross-platform; CLI.
- hub - GitHub integration with the git CLI. Very useful for checking out pull requests.

1.1.3 An Important Note About Hotfixes

Many git-flow tools will try to force you to merge hotfix branches into master *before* publishing code to a hosting service (e.g. Github). Do **not** do this. Just push your hotfix branch on its own and *send a pull request*.

```
# Push a new hotfix branch up to origin
$ git push -u origin hotfix/nav-header-size
```

1.1.4 Package and App Maintainers: Release How-to

Hotfix releases

- Once a hotfix PR has been checked out locally and *code review* is complete, rename the hotfix branch with the incremented PATCH number.

```
# hotfix/fix-serialization-bug is currently checked out
# rename with version number
$ git branch -m hotfix/0.16.3
```

- Finish the hotfix with git-flow.

```
$ git flow hotfix finish 0.16.3
```

- When prompted to add a tag message, write a brief (1-2 sentence) description of the hotfix.

Note: You can also use the `hotfix` invoke task to automatically rename the current hotfix branch and finish it.

```
$ invoke hotfix --finish
```

- Push `develop` and `master`. Push tags.

```
$ git push origin develop
$ git push origin master
$ git push --tags
```

- Once Travis tests pass, deploy to production and staging.

Feature releases

- Once `develop` is ready for release, start a release branch with git-flow.

```
$ git flow release start 0.17.0
```

- Update the CHANGELOG and bump the version where necessary. Commit changes.
- Finish the release with git-flow

```
$ git flow release finish 0.17.0
```

- If prompted to add a tag message, write `See CHANGELOG`.
- Push `develop` and `master`. Push tags.

```
$ git push origin develop
$ git push origin master
$ git push --tags
```

- Once Travis tests pass, deploy to production and staging.

1.2 Testing

Note: The below examples are in Python, but the concepts apply to testing in any language.

See also:

Looking for OSF-specific testing guidelines? See the *Testing the OSF* page.

1.2.1 General Testing Guidelines

- Use long, descriptive names. This often obviates the need for doctrings in test methods. This also makes it easier to locate tests that fail.
- Tests should be isolated. Don't interact with a real database or network. Use a separate test database that gets torn down or use mock objects.
- Prefer factories to fixtures.
- Never let incomplete tests pass, else you run the risk of forgetting about them. Instead, add a placeholder like `assert False, "TODO: finish me"`. If you are stubbing out a test that will be written in the future, use the `@unittest.skip()` decorator.
- Strive for 100% code coverage, but don't get obsess over coverage scores.
- When testing the contents of a dictionary, test the keys individually.

```
# Yes
assert_equal(result['foo'], 42)
assert_equal(result['bar'], 24)

# No
assert_equal(result, {'foo': 42, 'bar': 24})
```

1.2.2 Unit Tests

- Focus on one tiny bit of functionality.
- Should be fast, but a slow test is better than no test.
- It often makes sense to have one testcase class for a single class or model.

```
import unittest
import factories

class PersonTest(unittest.TestCase):
    def setUp(self):
        self.person = factories.PersonFactory()

    def test_has_age_in_dog_years(self):
         assert self.person.dog_years == self.person.age / 7
```

1.2.3 Functional Tests

Functional tests are higher level tests that are closer to how an end-user would interact with your application. They are typically used for web and GUI applications.

- Write tests as scenarios. Testcase and test method names should read like a scenario description.

- Use comments to write out stories, *before writing the test code*.

```
class TestAUser(unittest.TestCase):
    def test_can_write_a_blog_post(self):
        # Goes to the her dashboard
        ...
        # Clicks "New Post"
        ...
        # Fills out the post form
        ...
        # Clicks "Submit"
        ...
        # Can see the new post
        ...
```

Notice how the testcase and test method read together like "Test A User can write a blog post".

1.2.4 Supporting Libraries

Python

- nose: Extends Python's unittest. Includes a test runner and various utilities.
- pytest: A powerful test runner and library for writing automated tests.
- factory-boy: Utility library for creating test objects. Replaces fixtures with "factories".
- mock: Allows you to mock and patch objects for testing purposes.
- webtest / webtest-plus : Provides a `TestApp` with which to send test requests and make assertions about the responses.
- faker : A fake data generator.

Javascript

- Karma: Test runner.
- Mocha: Test framework/interface.
- Chai: Assertion library.
- Sinon: Test spies and mocks.

1.3 Sending a Pull Request

Use the following checklist to make sure your pull request can be reviewed and merged as efficiently as possible:

- For projects that use git-flow (such as the OSF): Feature branches should request to the `develop` branch. Hotfix branches should request to the `master` branch.
- New features must be *tested appropriately* (views, functional, and/or unit tests). Fixes should include regression tests.
- Your code must be sufficiently documented. Add docstrings to new classes, functions, and methods.
- Your code must be passing on TravisCI.

- On Github, rename your PR title with the prefix "[feature]", "[feature fix]" (fixes to develop), or "[hotfix]", as appropriate.
- If you are sending the PR for code review only and *not* for merge, add the "[WIP]" prefix to the PR's title.
- **Write a descriptive Pull Request description (See sample below). Ideally, it should communicate:**
  - **Purpose**
    - The overall philosophy behind the changes you've made, so that, if there are questions as to whether an implementation detail was appropriate, this can be consulted before talking with the developer.
    - Which Github issue the PR addresses, if applicable.
  - **Changes.**
    - The details of the implementation as you intended them to be. If you did front-end changes, add screenshots here.
  - **Side effects.**
    - Potential concerns, esp. regarding security, privacy, and provenance, which will requires extra care during review.
- Once your PR is ready, ask for code review on Flowdock.

Note: Make sure to follow the *Git style guidelines*.

1.4 Sample Pull Request Description

1.4.1 Purpose

Currently, the only way to add projects to your Dashboard's Project Organizer is from within the project organizer. There are smart folders with your projects and registrations, and you can search for projects from within the info widget to add to folders, but if you are elsewhere in the OSF, it's a laborious process to get the project into the Organizer. This PR allows you to be on a project and add the current project to the Project Organizer.

Closes Issue https://github.com/CenterForOpenScience/osf.io/issues/1186

1.4.2 Changes

Puts a button on the project header that adds the current project to the Dashboard folder of the user's Project Organizer. Disabled if the user is not logged in, if the folder is already in the dashboard folder of the user's project organizer, or if the user doesn't have permissions to view.

1.4.3 Side Effects

Also fixes a minor issue with the watch button count being slow to update.

1.5 Code Review

Guidelines for our code review process.

1.5.1 Everyone

- Ask for clarification. ("I didn't understand this comment. Can you clarify?")
- Talk in person if there are too many "I didn't understand" comments.

1.5.2 Having Your Code Reviewed

Before sending a pull request for code review, make sure you have met the *PR guidelines*.

- It may be difficult not to perceive code review as personal criticism, but, keep in mind, it is a review of the code, not the person. We can all learn from eachother, and code reviews provide a good environment to do so.
- After addressing all comments from a review, ping your on Flowdock for the next pass.
- If there is a style guideline that affects your PR and you believe the guideline is incorrect, post an issue or send a PR to the COSDev repo rather than discussing it in the PR.

1.5.3 Reviewing Code

- Make sure you understand the purpose of the code being reviewed.
- Checkout the branch being reviewed, and manually test the intended behavior.
- In your comments, keep in mind the fact that what you're saying can easily be perceived as personal criticism (even if it's not–it shouldn't be) and adjust your tone accordingly.
- After doing a pass of code review, "Approve" or "Request Changes" in the GitHub UI.
- Style fixes should refer to the style guides, when possible.

Example style comment:

```
> Use parentheses for line continuation.

From http://cosdev.readthedocs.org/en/latest/style_guides/python.html:
```

CHAPTER 2

Style Guides

2.1 Python

Follow PEP8, when sensible.

2.1.1 Naming

- **Variables, functions, methods, packages, modules**
  - `lower_case_with_underscores`
- **Classes and Exceptions**
  - `CapWords`
- **Protected methods and internal functions**
  - `_single_leading_underscore(self, ...)`
- **Private methods**
  - `__double_leading_underscore(self, ...)`
- **Constants**
  - `ALL_CAPS_WITH_UNDERSCORES`

General Naming Guidelines

Use singlequotes for strings, unless doing so requires lots of escaping.

Avoid one-letter variables (esp. `l`, `O`, `I`).

Exception: In very short blocks, when the meaning is clearly visible from the immediate context

```
for e in elements:
    e.mutate()
```

Avoid redundant labeling.

```
# Yes
import audio

core = audio.Core()
controller = audio.Controller()

# No
import audio

core = audio.AudioCore()
controller = audio.AudioController()
```

Prefer "reverse notation".

```
# Yes
elements = ...
elements_active = ...
elements_defunct = ...

# No
elements = ...
active_elements = ...
defunct_elements ...
```

Avoid getter and setter methods.

```
# Yes
person.age = 42

# No
person.set_age(42)
```

2.1.2 Indentation

Use 4 spaces–never tabs. You may need to change the settings in your text editor of choice.

2.1.3 Imports

Import entire modules instead of individual symbols within a module. For example, for a top-level module `canteen` that has a file `canteen/sessions.py`,

```
# Yes

import canteen
import canteen.sessions
from canteen import sessions

# No
from canteen import get_user  # Symbol from canteen/__init__.py
from canteen.sessions import get_session  # Symbol from canteen/sessions.py
```

Exception: For third-party code where documentation explicitly says to import individual symbols.

Rationale: Avoids circular imports. See here.

Put all imports at the top of the page with three sections, each separated by a blank line, in this order:

1. System imports
2. Third-party imports
3. Local source tree imports

Rationale: Makes it clear where each module is coming from.

If you have intentionally have an unused import that exists only to make imports less verbose, be explicit about it. This will make sure that someone doesn't accidentally remove the import (not to mention that it keeps linters happy)

```
from my.very.distant.module import Frob

Frob = Frob
```

2.1.4 String formatting

Prefer `str.format` to "%-style" formatting.

```
# Yes
'Hello {}'.format('World')
 # OR
'Hello {name}'.format(name='World')

# No

'Hello %s' % ('World', )
```

2.1.5 Print statements

Use the `print()` function rather than the `print` keyword (even if you're using Python 2).

```
# Yes
print('Hello {}'.format(name))

# No
print 'Hello %s ' % name
```

2.1.6 Documentation

Follow PEP257's docstring guidelines. reStructured Text and Sphinx can help to enforce these standards.

All functions should have a docstring - for very simple functions, one line may be enough:

```
"""Return the pathname of ``foo``."""
```

Multiline docstrings should include:

- Summary line
- Use case, if appropriate

- Args
- Return type and semantics, unless `None` is returned

```
"""Train a model to classify Foos and Bars.

Usage::

    >>> import klassify
    >>> data = [("green", "foo"), ("orange", "bar")]
    >>> classifier = klassify.train(data)

:param train_data: A list of tuples of the form ``(color, label)``.
:return: A trained :class:`Classifier <Classifier>`
"""
```

Notes

- Use action words ("Return") rather than descriptions ("Returns").
- Document `__init__` methods in the docstring for the class.

```
class Person(object):
    """A simple representation of a human being.

    :param name: A string, the person's name.
    :param age: An int, the person's age.
    """
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

2.1.7 On Comments

Use them sparingly. Prefer code readability to writing a lot of comments. Often, small methods and functions are more effective than comments.

```
# Yes
def is_stop_sign(sign):
    return sign.color == 'red' and sign.sides == 8

if is_stop_sign(sign):
    stop()

# No
# If the sign is a stop sign
if sign.color == 'red' and sign.sides == 8:
    stop()
```

When you do write comments, use them to explain *why* a piece code was used, not *what* it does.

Method Overrides

One useful place for comments are method overrides.

```
class UserDetail(generics.RetrieveUpdateAPIView, UserMixin):

    # overrides RetrieveUpdateAPIView
    def get_serializer_context(self):
        return {'request': self.request}
```

2.1.8 Calling Superclasses' Methods

Use `super` when there is only one superclass.

```
class Employee(Person):

    def __init__(self, name):
        super(Employee, self).__init__(name)
        # or super().__init__(name) in Python 3
        # ...
```

Call the method directly when there are multiple superclasses.

```
class DevOps(Developer, Operations):

    def __init__(self):
        Developer.__init__(self)
        # ...
```

2.1.9 Line lengths

Don't stress over it. 80-100 characters is fine.

Use parentheses for line continuations.

```
wiki = (
    "The Colt Python is a .357 Magnum caliber revolver formerly manufactured "
    "by Colt's Manufacturing Company of Hartford, Connecticut. It is sometimes "
    'referred to as a "Combat Magnum". It was first introduced in 1955, the '
    "same year as Smith & Wesson's M29 .44 Magnum."
)
```

2.1.10 Recommended Syntax Checkers

We recommend using a syntax checker to help you find errors quickly and easily format your code to abide by the guidelines above. Flake8 is our recommended checker for Python. It will check for both syntax and style errors and is easily configurable. It can be installed with pip:

```
$ pip install flake8
```

Once installed, you can run a check with:

```
$ flake8
```

Note: We highly recommend that you add a git hook to check your code before you commit it. You only need to run the following command once:

```
# Current directory must be a git repo
$ flake8 --install-hook
```

This adds the proper hook to `.git/hooks/pre-commit`.

There are a number of plugins for integrating Flake8 with your preferred text editor.

Vim

- syntastic (multi-language)

Sublime Text

- Sublime Linter with SublimeLinter-flake8 (must install both)

2.1.11 Credits

- PEP8 (Style Guide for Python)
- Pythonic Sensibilities
- Python Best Practice Patterns

2.2 Javascript

Style guidelines for writing Javascript.

See also:

Writing a JS module for the OSF? See the *Javascript Modules* page in the OSF section.

2.2.1 Style

Follow Felix's Node Style and airbnb's Style Guide with a few exceptions:

- Use **4 spaces** for indentation.
- Use `self` to save a reference to `this`.

2.2.2 Errors

- Always throw `Error` instances, not strings.

```
// Yes
throw new Error('Something went wrong');

// No
throw 'Something went wrong';

// No
throw Error('Something went wrong');
```

2.2.3 CommonJS Modules

- Group imports in the following order, separated by a blank line:

1. Third party libraries
2. Local application/library-specific imports

- `module.exports` are always grouped at the end of a file. Do not use `export` throughout the file.
- Keep testability in mind in deciding what to export.

```
// Yes
module.exports = {
    SomeClass: SomeClass,
    _privateFunction: privateFunction
}

// Yes
function SomeClass() { ... }
SomeClass._privateFunction = function() {...}

module.exports = SomeClass;

// No
var SomeClass = exports.SomeClass = function() { ... };
var privateFunction = exports._privateFunction = function() { ... };
```

2.2.4 Documentation

Use the YUIDoc standard for writing JS comment blocks.

Example:

```
/**
* A wrapper around the ACE editor that asynchronously loads
* and publishes its content.
*
* @param {String} selector Selector for the editor element
* @param {String} url URL for retrieving and posting content.
*/
```

For simple functions and methods, a single-line docstring will suffice.

```
/** Update the viewModel with data fetched from a server. */
```

2.2.5 jQuery

Follow Abhinay Rathore's jQuery Coding Standards Guide.

AJAX

For PUTting and POSTing to JSON endpoints in the OSF, use the `$osf.postJSON` and `$osf.putJSON` functions (located in `osfHelpers.js`). This will handle JSON stringification as well as set the correct dataType and contentType.

When using `$osf.postJSON`, `$osf.putJSON`, or `jQuery.ajax`, use the Promises interface.

```
function successHandler(response) { ... }
function failureHandler(jqXHR, status, error) {...}


var request = $.ajax({ ... });
request.done(successHandler);
request.fail(failureHandler);

// OR
$.ajax({ ... }).then(successHandler, failureHandler);
```

2.2.6 Promises

- Prefer promises to callbacks.

```
// Yes
function makeRequest() {
    var request = $.getJSON('/api/projects/');
    return request;
}
var request = makeRequest();
request.done(function(response) { console.log(response); })

// No
function makeRequest(callback){
    $.getJSON('/api/projects/', function(response) {
        callback && callback(response);
    }) ;
}
makeRequest(function(response) {console.log(response)});
```

- When doing AJAX requests or other async work, it's often useful to return a promise that resolves to a useful value (e.g. model objects or "unwrapped" responses).

```
function User(data) {
    this._id = data._id;
    this.username = data.username;
}

/** Return a promise that resolves to a list of Users */
var getUsers = function() {
    var ret = $.Deferred();

    var request = $.getJSON('/users/');
    request.done(function(response) {
        var users = $.map(response.users, function(data){
            return User(data);
        });
        ret.resolve(users);
    });
    request.fail(function(xhr, status, error) {
        Raven.captureMessage(...);
        ret.reject(xhr, status, error);
    });
    return ret.promise();
```

```
};

getUsers().done(function(users){
    users.forEach(function(user)){
        console.log(user._id);
        console.log(user.username);
    };
})
```

2.2.7 Encapsulation

Use the Combination Constructor/Prototype pattern for encapsulation. You can use the following functions to provide syntactic sugar for creating "classes":

```
function noop() {}

function defclass(prototype) {
    var constructor = prototype.hasOwnProperty('constructor') ? prototype.constructor␣
↪: noop;
    constructor.prototype = prototype;
    return constructor;
}

function extend(cls, sub) {
    var prototype = Object.create(cls.prototype);
    for (var key in sub) { prototype[key] = sub[key]; }
    prototype.super = cls.prototype;
    return defclass(prototype);
}

// Example usage:
var Animal = defclass({
    constructor: function(name) {
        this.name = name || 'unnamed';
        this.sleeping = false;
    },
    sayHi: function() {
        console.log('Hi, my name is ' + this.name);
    }
});

var Person = extend(Animal, {
    constructor: function(name) {
        this.super.constructor.call(name);
        this.name = name || 'Steve';
    }
});
```

Note: In the OSF, the `defclass` and `extend` functions are available in the `oop.js` module.

2.2.8 Recommended Syntax Checkers

We recommend using a syntax checker to help you find errors quickly and easily format your code to abide by the guidelines above. JSHint is our recommended checker for Javascript. It can be installed with `npm`:

```
$ npm install -g jshint
```

There are a number of plugins for integrating jshint with your preferred text editor.

Vim

- syntastic (multi-language)

Sublime Text

- Sublime Linter with SublimeLinter-jshint (must install both)

PyCharm

- Follow these docs: http://www.jetbrains.com/pycharm/webhelp/jshint.html

2.3 HTML and CSS

- Follow mdo's Code Guide, with one exception: Use four spaces for indentation (instead of two).
- Use `.lowercase-and-dashes` for class names and `#camelCase` for IDs.
- Add a comment marking the end of large blocks. Use `<!-- end class-name -->`

```
<div class="container-fluid">
    Lots of markup...
</div><!-- end container-fluid -->
```

- Avoid inline CSS. Prefer CSS classes for maintainability and reuseability.

2.4 Git

- Use the imperative mode (e.g, "Fix rendering of user logs") in commit messages.
- If your patch addresses a JIRA ticket, add the JIRA ticket ID to the commit message.

```
Improve UI for changing names

- Change button color for Auto-fill
- Add help text

[#OSF-4251]
```

- If your patch fixes a Github issue, you can add the issue to your commit message so that the issue will automatically be closed when the patch is merged.

```
Fix bug in loading filetree

[fix CenterForOpenScience/osf.io#982]
```

- Write good commit messages.

Here's a model message, taken from the above post:

```
Capitalized, short (50 chars or less) summary

More detailed explanatory text, if necessary.  Wrap it to about 72
characters or so.  In some contexts, the first line is treated as the
subject of an email and the rest of the text as the body.  The blank
line separating the summary from the body is critical (unless you omit
the body entirely); tools like rebase can get confused if you run the
two together.

Write your commit message in the imperative: "Fix bug" and not "Fixed bug"
or "Fixes bug."  This convention matches up with commit messages generated
by commands like git merge and git revert.

Further paragraphs come after blank lines.

- Bullet points are okay, too

- Typically a hyphen or asterisk is used for the bullet, followed by a
  single space, with blank lines in between, but conventions vary here

- Use a hanging indent
```

2.5 Ansible

- Prefer dictionary syntax for passing many arguments to a module.

```
- name: Ensure proper permissions on apps directory
  file:
    path: "/opt/apps/"
    mode: 0755
    group: "osf"
    owner: "www-data"
```

- Do *not* prefix task names with the name of the role.

```
# YES
- name: Make user python is installed
  apt: name="python-dev"

# NO
- name: uwsgi | Make user python is installed
  apt: name="python-dev"
```

- Prefix all default variables with the role name and an underscore.

```
# OSF role

osf_virtualenv: "/opt/envs/osf/"
osf_repo_branch: "master"
```

- Document default variables using comments.

CHAPTER 3

OSF Development

3.1 Setting up the OSF

Use Docker and Docker Compose for local development. See https://github.com/CenterForOpenScience/osf.io/blob/develop/README-docker-compose.md for up-to-date docs for running the OSF in Docker locally.

3.2 OSF Guidelines

3.2.1 General

- For node endpoints, use `node.url_for` and `node.api_url_for` for URL lookup

```
# Assuming a URL Rule:
# Rule(
#     [
#         '/project/<pid>/tags/<tid>',
#         '/project/<pid>/node/<nid>/tags/<tid>/',
#     ],
#      'put',
#      node_views.node_tags_put,
#      json_renderer,
# )

# Yes
# Pass the name of the view function and URL params as keyword arguments
url = node.api_url_for('node_tags_put', tid=tag._id)
# => /project/1rdsf/tags/mytag/

# No
url = os.path.join('/api', 'v1', node._primary_key, 'tags', tag._id)
```

- Use `website.utils.api_url_for` and `website.utils.web_url_for` for general URL lookup.

```
# Yes
from website.utils import api_url_for
url = api_url_for('user_settings')

# No
url = os.path.join('/user', 'settings')
```

- Use the above functions in Mako templates; they are available by default.

```
<!-- Yes -->
<p>Visit your <a href="${ web_url_for('user_settings') }">user settings</a>.

<!-- No -->
<p>Visit your <a href="/settings/">user settings</a>.
```

3.2.2 Views

- If a decorator injects keyword arguments, declare the keyword arguments whenever possible. Avoid pulling them from the kwargs dictionary.

```
# Yes
@must_be_logged_in
def user_settings_put(auth, **kwargs):
    #...

@must_be_contributor_or_public
def get_project_comments(auth, node, **kwargs):
    # ...

# No
@must_be_logged_in
def user_settings_put(**kwargs):
    auth = kwargs['auth']
    #...
```

- Use `framework.flask.redirect` to return redirect responses. It has the same functionality as `flask.redirect` except that it will reappend querystring parameters for view-only links when necessary. Do **not** use `flask.redirect`.

3.2.3 Responses

- Use correct HTTP status codes. You can used the constants in `httplib` to help.

```
# Yes
@must_be_logged_in
def user_token_post(auth, **kwargs):
    #...
    return serialized_settings, 201
    # OR
    # return serialized_settings, httplib.CREATED

# No
@must_be_logged_in
def user_token_post(auth, **kwargs):
```

```
    #...
    return serialized_settings  # Implicitly returns 200 response
```

- Be consistent with your response format.

  TODO: Come up with a standard format. The Dropbox add-on uses the following, though we may decide on a different convention later.

```
{
    "result": {"name": "New Project", "id": ...} # ... the requested object(s) ,
    "message": "Successfully created project" # ... an optional message
}
```

- Prefer namespaced representations to arbitrary prefixes in response data.

```
// Yes
{
    'node': {
        '_id': '123abc',
        'urls': {
            'api': '/api/v1/123abc',
            'web': '/123abc/'
        }
    },
    'urls': {
        'latest': '/files/some-file-id/latest/',
        'detail': '/files/some-file-id/'
    }
}

// No
{
    'node_id': '123abc',
    'node_api_url': '/api/v1/123abc',
    'node_web_url': '/123abc/',
    'latest_file_url': '/files/some-file-id/latest/',
    'file_detail_url': '/files/some-file-id/'
}
```

3.2.4 Running Migrations

Migrations are located in the `scripts` directory.

To run them:

```
$ python -m scripts.script_name
```

To migrate search records:

```
invoke migrate_search
```

3.2.5 Error Handling

Server-side

If a view should return an error response, raise a `framework.exceptions.HTTPError`, optionally passing a short and long message. This will ensure that a properly formatted HTML or JSON response is returned (depending on whether the route is an API or web route). **Do NOT** return a dictionary.

```
from framework.exceptions import HTTPError

@must_be_logged_in
def user_settings_get(auth, **kwargs):
    """Return the current user's settings."""
    try:
        settings = get_user_settings(auth)
    except ModularOdmException:
        raise HTTPError(404,
            msg_short='User not found',
            msg_long='The user could not be in our database.'
        )
    return serialized_settings(settings), 200
```

Client-side

All client-side HTTP requests should have proper error handlers. As an example, you might display an error message in a modal if a request fails.

Note: Use RavenJS (a JS client for Sentry) to log unexpected errors to our Sentry server.

```
var url = '/api/v1/profile';
var request = $osf.putJSON(url, {'email': 'foo@bar.com'});

request.done(function(response) { ... });

request.fail(function(jqxhr, status, error) {
    bootbox.alert({
        title: "Error",
        message: "We're sorry. Your profile could not be updated at this time. Please␣
↪try again later."
    });
    // Log error to Sentry
    // Add context (e.g. error status, error messages) as the 2nd argument
    Raven.captureMessage('Error while updating user profile', {
        url: url, status: status, error: error
    });
});
```

When appropriate, you can use the generic `$osf.handleJSONError`, which will display a generic error message in a modal to the user if a failure occurs.

```
var $osf = require('osfHelpers');
// ...
request.fail($osf.handleJSONError);
```

3.2.6 Documentation

Docstrings

- Write function docstrings using Sphinx conventions (see here).
- For parameters that are not passed directly to the function (e.g. query string arguments, POST arguments), include the source of the parameter in the docstring:

```
def my_view(my_param):
    """Do something rad.

    :param str my_param: My directly passed parameter
    :param-query str foo: A parameter included in the query string; look me up in
↪`request.args`
    :param-post str bar: A parameter included in the POST payload; look me up in
↪`request.form`
    :param-json str baz: A parameter included in the JSON payload; look me up in
↪`request.json`

    """
    # Rad code here
```

3.2.7 Misc

Generating fake data

1. Install fake-factory

```
$ pip install fake-factory
```

2. Create your an account on your local osf. Remember the email address you use.

3. Run the fake data generator script, passing in your username (email)

```
$ python -m scripts.create_fakes --user fred@cos.io
```

where `fred@cos.io` is the email of the user you created.

After you run the script, you will have 3 fake projects, each with 3 fake contributors (with you as the creator).

Dialogs

We use Bootbox to generate modal dialogs in the OSF. When calling a `bootbox` method, always pass in an object of arguments rather than positional arguments. This allows you to include a title in the dialog.

```
// Yes
bootbox.confirm({
    title: 'Permanently delete file?',
    message: 'Are you sure you want to delete this file?',
    callback: function(confirmed) {
        // ..
    }
})

// No
```

```
bootbox.confirm('Are you sure you want to delete this file?',
    function(confirmed) {
        // ...
    }
)
```

3.3 Testing the OSF

Warning: This page contains outdated information. We now use pytest and pytest-django instead of nose for tests.

This page includes information about testing the OSF codebase.

See also:

For more general testing guidelines, see the *Testing* page.

3.3.1 The `OsfTestCase`

The `tests.base.OsfTestCase` class is the base class for all OSF tests that require a database. Its class setup and teardown methods will create a temporary database that only lives for the duration of the test class.

A few things to note about the `OsfTestCase`:

- Its `setUp` method will instantiate a `webtest_plus.TestApp`. You should **not** instantiate a `TestApp` yourself. Just use `self.app`.
- If you override `setUp` or `tearDown`, you must **always** call `super(YourTestClass, self).setUp` or `super(YourTestClass, self).tearDown()`, respectively.
- Following the above two rules ensures that your tests execute within a Flask app context.
- The test database lives for the duration of a test class. This means that database records created within a TestCase's methods may interact with each other in unexpected ways. Use *factories* and the `tests.base.fake` generator for creating unique test objects.

3.3.2 Factories

We use the factory-boy library for defining our factories. Factories allow you to create test objects customized for the current test, while only declaring test-specific fields.

Using Factories

```
from tests.factories import UserFactory
from tests.base import fake

class TestUser(OsfTestCase):

    def test_a_method_of_the_user_class(self):
        user = UserFactory()  # creates a user
        user2 = UserFactory()  # creates a user with a different email address
```

```
        # You can also specify attributes when needed
        user3 = UserFactory(username='fredmercury@queen.io')
        user4 = UserFactory(password=fake.md5())
        # ...
```

3.3.3 Unit Tests

Testing Models

Unit tests for models belong in `tests/test_models.py`. Each model should have its own test class. You can have multiple test classes for a single model if necessary.

```
from frameworks.auth.core import User

from tests.base import OsfTestCase, fake

class TestUser(OsfTestCase):

    def test_check_password(self):
        user = User(username=fake.email(), fullname='Nick Cage')
        user.set_password('ghostrider')
        user.save()
        assert_true(user.check_password('ghostrider'))
        assert_false(user.check_password('ghostride'))

    # ...
```

3.3.4 Views Tests

Views tests are used to test that our endpoints return the expected responses. We use the webtest library to interact with our application under test.

The `OsfTestCase` provides a `self.app` attribute that is a `webtest_plus.TestApp` object.

Things to test:

- Status codes
- JSON responses
- Records are updated appropriately in the database

```
from tests.base import OsfTestCase
from tests.factories import ProjectFactory, AuthUserFactory

class TestProjectViews(OsfTestCase):

    def setUp(self):
        OsfTestCase.setUp(self)
        # The AuthUserFactory automatically generates an
        # API key for the user. It can be accessed from the
        #  `auth` attribute
        self.user = AuthUserFactory()
        self.project = ProjectFactory(creator=self.user)
```

```
    # Status codes should be tested
    def test_get_project_returns_200_with_auth(self):
        url = self.project.api_url_for('project_get')
        # This endpoint requires authentication. We use the user's API key to
        # circumvent the login process
        res = self.app.get(url, auth=self.user.auth)
        assert_equal(res.status_code, 200)

        # The JSON response is correct
        assert_equal(res.json['id'], self.project._id)
        assert_equal(res.json['title'], self.project.title)
        # ...

    def test_get_project_returns_403_with_no_auth(self):
        url = self.project.api_url_for('project_get')
        # Make sure to pass expect_error=True if you expect an error response.
        res = self.app.get(url, auth=self.user.auth, expect_errors=True)
        assert_equal(res.status_code, 403)
```

3.3.5 Functional Tests

Functional tests in the OSF also use webtest. These tests mimic how a user would interact with the application through their browser.

Things to test:

- User interactions, such as clicking on links, filling out forms
- Content that you expect to appear on the page.

```
from tests.base import OsfTestCase
from tests.factories import ProjectFactory, AuthUserFactory

class TestProjectDashboard(OsfTestCase):

    def setUp(self):
        OsfTestCase.setUp(self)
        self.user = AuthUserFactory()
        self.project = ProjectFactory(creator=self.user)

    # Use line comments to write out user stories
    def test_can_access_wiki_from_project_dashboard(self):
        # Goes to project dashboard (user is logged in)
        url = self.project.web_url_for('view_project')
        res = self.app.get(url, auth=self.user.auth)

        # Clicks the Wiki link,
        # follows redirect to wiki home page
        res = res.click('Wiki').follow()

        # Sees 'home' on the page
        assert_in('home', res)
```

Note: The `TestResponse.showbrowser()` method is especially useful for debugging functional tests. It allows you to open the current page in your browser at a given point in the test.

```
res = self.app.get(url)
res.showbrowser()  # for debugging
```

Just be sure to remove the line when you are done debugging.

3.3.6 Regression Tests

Regression tests may fall under any one of the categories above (unit, model, views, functional). If you write a regression test for a specific issue, it is often helpful to link to the issue in a line comment above the test.

```
# Regression test for https://github.com/CenterForOpenScience/osf.io/issues/1136
def test_cannot_create_project_with_blank_name(self):
    # ...
```

3.3.7 Javascript Tests

Running tests

Before running tests, make sure you have the dependencies installed.

```
$ npm install
```

Javascript tests are run with

```
$ inv karma
```

This will start a Karma process which will run the tests on every JS code change.

You can specify which browser to run your tests against by passing the `--browser` (or `-b`, for short) option.

```
$ inv karma -b Chrome
```

Chrome and Firefox are supported after you've run `npm install`. To run on other browsers, install the appropriate launcher with `npm` (see here for available launchers).

```
$ npm install karma-safari-launcher
$ inv karma -b Safari
```

Writing Tests

We use the following libraries for writing tests:

- Mocha: Provides the interface for test cases.
- Chai: Provides assertion functions.
- Sinon: Provides test spies, stubs, and mocks.

See the official docs for these libraries for more information.

OSF-specific Guidelines

- Core OSF tests go in `website/static/js/tests/`. Addons tests go in `website/addons/<addon_name>/static/tests/`
- Karma will run every module that has the `.test.js` extension.
- Use Chai's `assert` interface.
- To mock HTTP requests, use the `createServer` utility from the `js/tests/utils` module.

Gotchas and Pitfalls

- When mocking out endpoints with sinon, be careful when dealing with URLs that accept query parameters. You can pass a regex as a `url` value to `createServer`.

```
var endpoints = {
    // Use regex to handle query params
    {url: /\/api\/users\/.+/, response: {...}}
];
server = utils.createServer(sinon, endpoints);
```

- Remember for async tests, you need to pass and call the 'done' callback. Failing to pass and call done in async tests can cause unpredictable and untracable errors in your test suite.

In particular you might see failed assertions from another test being printed to the console as if they're happening in some other test. Since we're concatenating test files together with webpack, this error could be coming from any of the tests run before the error occurs (maybe from another file altogether).

```
describe('My feature', () => {
  ...
  it('Does something asnyc', (done) => {
     myFeature.myAsyncFunction()
       .always(function() {
         // make some assertions
         done();
       });
  });
});
```

Test Boilerplate

The following boilerplate should be included at the top of every test module.

```
/*global describe, it, expect, example, before, after, beforeEach, afterEach, mocha,␣
↪sinon*/
'use strict';
var assert = require('chai').assert;
// Add sinon asserts to chai.assert, so we can do assert.calledWith instead of sinon.
↪assert.calledWith
sinon.assert.expose(assert, {prefix: ''});
```

Debugging tests

- Run karma: `inv karma`

- Browse to `localhost:9876` in your browser.
- Click the DEBUG button on the top right.
- Open dev tools and open up the debugger tab.
- Add breakpoints or `debugger;` statements where necessary.

Testing Internet Explorer on a Mac

- Install Virtualbox and Internet Explorer.
- Pick a name for your Microsoft IE localhost, in this example, we will use "windows.fun"
- Add the following lines of code to your website/settings/local.py.

```
ELASTIC_URI = 'windows.fun:9200'
DOMAIN = 'http://windows.fun:5000/'
API_DOMAIN = 'http://windows.fun:8000/'
ELASTIC_URI = 'windows.fun:9200'
WATERBUTLER_URL = 'http://windows.fun:7777'
CAS_SERVER_URL = 'http://windows.fun:8080'
MFR_SERVER_URL = 'http://windows.fun:7778'
```

- Add the following to your /etc/hosts file on the mac with this line.

```
129.0.0.1 windows.fun
```

- In Virtualbox, update your windows hosts file with the following line.

```
10.0.2.2 windows.fun
```

- In Virtualbox preferences, set the network adaptor Attached To setting to Nat
- As of this writing, Internet Explorer's debugger doesn't work without an update. To update, go to this link and patch IE.
- Restart everything in the OSF, and how you can access the osf on Internet Explorer from http://windows.fun:5000. The http://locahost:5000 url will still work on your mac browser.

3.4 Javascript Modules How-To

This section describes how to write Javascript modules for the OSF, use webpack to build assets, and include built assets in HTML. We also provide starter templates for new JS modules.

See also:

Looking for the JS style guidelines? See *here* .

3.4.1 Writing Modules

- Use the CommonJS module style.
- Reuseable modules go in `website/static/js/`. Name modules in `lowerCamelCase`.
- Initialization code for a page goes in a module within `website/static/js/pages/`. Name page modules with `lower-dashed-case`.

A Note on Utility Functions

Put reusable utility functions in `website/static/js/osfHelpers.js`.

```
// osfHelpers.js

var myCopaceticFunction = function() {...}

// ...
module.exports = {
    // ...
    myCopaceticFunction: myCopaceticFunction
};
```

Example

Let's say you're creating a reuseable Markdown parser module for the wiki edit page. Your module would go in `website/static/js/`.

website/static/js/osfMarkdownParser.js

```
/**
 * A Markdown parser with special syntax for linking to
 * OSF projects.
**/
'use strict';

// CommonJS/Node-style imports at the top of the file

var $osf = require('js/osfHelpers');

// Private methods go up here
function someHelper() {
    // ....
}
// This is the public API
// The constructor
function OSFMarkdownParser (selector, options) {
    this.selector = selector;
    this.options = options;
    this.init();
}
// Methods
OSFMarkdownParser.prototype.init = function() {
    //...
}

OSFMarkdownParser.prototype.somePublicMethod = function() {
    //...
}

// Export the constructor
module.exports = OSFMarkdownParser;
```

The initialization of your Markdown parser would go in `website/static/js/pages/wiki-edit-page.js` (assume that this file already exists).

website/static/js/pages/wiki-edit-page.js

```
// Initialization of the Markdown parser
var OSFMarkdownParser = require('js/osfMarkdownParser');

new OSFMarkdownParser('#wikiInput', {...});

// ... other wiki-related initialization.
```

3.4.2 Third-party Libraries

The following libraries can be imported in your JS modules (using `require('name')`):

- Any library listed in bower.json
- Any library listed in package.json
- Any library listed in the resolve.alias entry of `webpack.common.config.js`

3.4.3 Building and Using Modules

Webpack Entry Points

Each module in `website/static/js/pages` corresponds to an entry point in webpack and has a rough one-to-one mapping with a page on the OSF. Here is what the `wiki-edit-page` entry would look like in the webpack configuration file.

webpack.common.config.js

```
// Entry points built by webpack. The keys of this object correspond to the
// names of the built files which are put in /website/static/public/js/. The values
// in the object are the source files.
var entry = {
    //...
    'wiki-edit-page': staticPath('js/pages/wiki-edit-page.js'),
    // ...
}
```

Note: You will seldom have to modify `webpack.common.config.js`. The only time you may need to care about it is when a completely new page is added to the OSF.

Building with Webpack

Webpack parses the dependency graphs of the modules defined in the entry points and builds them into single files which can be included on HTML pages. The built files reside in `website/static/public/js/`. Therefore, the built file which would include your Markdown parser initialization would be in `/static/public/js/wiki-edit-page.<hash>.js`. This is the file that would be included in the HTML template.

Note: Webpack will add a hash to the filenames of the built files to prevent users' browsers from caching old versions (example: `wiki-edit-page.js` becomes `wiki-edit-page.4ec1318376695bcd241b.js`).

Therefore, we need to resolve the short filenames to the full filenames when we include them in the HTML. More on that in the next section.

To build the assets for local development, use the `assets` invoke task.

```
$ inv assets --debug --watch
# OR
$ inv assets -dw
```

Loading the Modules in HTML with `webpack_asset`

Once you have the built assets, you can include them on HTML pages with a `<script>` tag. In order to resolve the short filenames to the filenames on disk (which include hashes), use the `webpack_asset` Mako filter.

website/templates/wiki/edit.mako

```
<%def name="javascript_bottom()">
<script src=${"/static/public/js/wiki-edit-page.js" | webpack_asset}></script>
</%def>
```

Examples

- js/folderPicker.js
- js/nodeControl.js is used within js/pages/project-base-page.js. The built file is included in templates/project_base.mako.

Todo: Document how to use mako variables in JS modules (`contextVars`)

3.4.4 Knockout Modules

A module contains the Knockout model(s) and ViewModel(s) for a single unit of funtionality (e.g. login form, contributor manager, log list, etc.)

Knockout modules aren't much different from regular modules.

- Apply bindings in the constructor.
- Use the `osfHelpers.applyBindings` helper. This will ensure that your ViewModel will be bound to the element that you expect (and not fall back to <body>, as `ko.applyBindings` will sometimes do). You can also pass `$osf.applyBindings` a selector instead of an `HTMLElement`.
- Name the HTML ID that you bind to with "Scope". Example: `<div id="logfeedScope">`.
- Adding the `scripted` CSS class to the div you bind to will hide the div until `$osf.applyBindings` finishes executing. This is useful if you don't want to show any HTML for your component until the ViewModel is bound.

website/static/js/logFeed.js

```
/**
 * Renders a log feed.
 **/
'use strict';
var ko = require('knockout');

var $osf = require('js/osfHelpers');
```

```
/**
* Log model.
*/
var Log = function(params) {
    var self = this;
    self.text = ko.observable('');
    // ...
};

/**
* View model for a log list.
* @param {Log[]} logs An array of Log model objects to render.
*/
var LogViewModel = function(logs) {
    var self = this;
    self.logs = ko.observableArray(logs);
    // ...
};

////////////////
// Public API //
////////////////

var defaults = {
    data: null,
    progBar: '#logProgressBar'
};

function LogFeed(selector, options) {
    var self = this;
    self.selector = selector;
    self.options = $.extend({}, defaults, options);
    self.$progBar = $(self.options.progBar);
    self.logs = self.options.data.map(function(log) {
        return new Log(log.params);
    })
};
// Apply ViewModel bindings
LogFeed.prototype.init = function() {
    var self = this;
    self.$progBar.hide();
    $osf.applyBindings(new LogViewModel(self.logs), self.selector);
};

module.exports = LogFeed;
```

website/static/pages/some-template-page.js

```
'use strict';

var LogFeed = require('js/logFeed');

// Initialize the LogFeed
new LogFeed('#logScope', {data: ...});
```

website/templates/some_template.mako

```
<div class="scripted" id="logScope">
    <ul data-bind="foreach: {data: logs, as: 'log'}">
        ...
    </ul>
</div>

<%def name="javascript_bottom()">
<script src=${"/static/public/js/some-template-page.js" | webpack_asset}></script>
</%def>
```

Examples

- Full LogFeed module
- comment.js

3.4.5 Templates

To help you get started on your JS modules, here are some templates that you can copy and paste.

JS Module Template

```
/**
 * [description]
 */
'use strict';
var $ = require('jquery');


function MyModule () {
    // YOUR CODE HERE
}

module.exports = {
    MyModule: MyModule
};
```

Knockout Module Template

```
/**
 * [description]
 */
'use strict';
var ko = require('knockout');

var $osf = require('js/osfHelpers');

function ViewModel(url) {
    var self = this;
    // YOUR CODE HERE
}
```

```
function MyModule(selector, url) {
    this.viewModel = new ViewModel(url);
    $osf.applyBindings(this.viewModel, selector);
}

module.exports = {
    MyModule
};
```

3.5 Developing An Addon

In Progress: Help out by sending a PR!

3.5.1 Notes and gotchas

- The words SHALL, MUST, MAY, etc are to be interpreted as defined here.
- The add-on system is module based not class based
- Everything you touch should be in the website/addons/ directory
- You MUST NOT instantiate an `AddonSettings` object yourself
- `to_json` returns the mako context for the settings pages
- Log templates: the `id` of each script tag correspond to log actions.
- Don't forget to do error handling! This includes handling errors that might occur if 3rd party HTTP APIs cause a failure and any exceptions that a client library might raise
- Any static assets that you put in `website/addons/<addon_name>/static/` will be served from `/static/addons/<addon_name>/`. This means that `<link>` and `<script>` tags should always point to URLs that begin with `/static/`.

3.5.2 Installing Add-ons

Open terminal and switch to the folder where your OSF installation is located. We will install the addons to the website folder. So navigate to

```
cd website/addons
```

During your installation you created a virtual environment for OSF. Switch to the environment by typing workon followed by the name of your virtual environment

```
# If you use virtualenvwrapper
$ workon osf
```

3.5.3 Bare minimums

- `__init__.py` declares all views/models/routes/hooks for your add-on
- Your add-on MUST declare the following in its `__init__.py`
- `SHORT_NAME` (string)

  - The name that will be used to refer to your add-on on the backend
  - EX:
    * Amazon Simple Storage Service is s3
    * Google Drive is googledrive
- `FULL_NAME` (string)
  - The name "display name" of your add-on, whenever the user is interacting with your add-on this is the name they will see
- `ROUTES` (list of *routes* dicts)
  - A list containing all *routes* defined by your add-on
  - Maps Urls to views
- `MODELS` (list of *StoredObjects*)
  - A list of all ODM objects defined by your add-on
  - If your model is not in this list it will not be usable
- `ADDED_DEFAULT` (list of strings)
  - A list of `AddonMixin` models that your add-on SHALL be added to when they are created
  - Valid options are `user` and `node`
  - EX:
  - The Wiki addon is added by default for nodes
- `ADDED_MANDATORY` (list of strings)
  - A list of AddonMixin models that your add-on MUST be attached/connected to at all times
  - Valid options are `user` and `node`
  - EX:
  - OsfStorage is a required add-on for nodes
- `VIEWS` (list of strings)
  - Additional builtin views for your add-on
  - Valid options are `page` and `widget`
  - EX: The wiki defines both a `page` view and a `widget` view
- `CATEGORIES` (list of strings)
  - A list of categories this add-on should be displayed under when the user is "browsing" add-ons
  - SHOULD be one of `documentation`, `storage`, `citations`, `security`, `bibliography`, and `other`
    * Additional categories can be added to `ADDON_CATEGORIES` in `website.settings.defaults`
- `INCLUDE_JS` and `INCLUDE_CSS`
  - Deprecated field, define as empty dict (`{}`)
- `OWNERS` (list of strings)
  - Valid options are `user` and `node`

- `CONFIGS` (list of strings)
  - Valid options are `accounts` and `node`

3.5.4 Optional Fields

Your add-on MAY define the following fields

- `HAS_HGRID_FILES` (boolean)
- A boolean that indicated that this add-on's `GET_HGRID_DATA` function should be used to populate the files grid
- `GET_HGRID_DATA` (function)
- A function that returns HGrid/Treebeard formatted data to be included in a project's files grid
- `USER_SETTINGS_MODEL` (*StoredObject*)
- MUST inherit from `website.addons.base.AddonUserSettingsBase`
- A model that will be used to store settings for users
- Will be returned when `User.get_addon('YourAddon')` is called
- EX:
  - S3's User settings is used to store the user's AWS access keys
- `NODE_SETTINGS_MODEL` (*StoredObject*)
- MUST inherit from `website.addons.base.AddonNodeSettingsBase`
- A model that will be used to store settings for nodes
- Will be returned when `Node.get_addon('YourAddon')` is called
- `NODE_SETTINGS_TEMPLATE` (string to directory)
- A mako template for configuring your add-on's node settings object
- `USER_SETTINGS_TEMPLATE` (string to directory)
- A mako template for configuring your add-on's user settings object
- `MAX_FILE_SIZE`
- This maximum size, in MB, that can be uploaded to your add-on, supposing it supports files

3.5.5 Addon Structure

An add-on SHOULD have the following folder structure

```
website/addons/addonshortname/
├── __init__.py
├── model.py
├── requirements.txt
├── routes.py
├── settings
│   ├── __init__.py
│   └── defaults.py
├── static
│   ├── comicon.png
│   ├── node-cfg.js*
```

```
│   ├── tests
│   │   └── ...
│   └── user-cfg.js*
├── templates
│   ├── log_templates.mako
│   ├── addonshortname_node_settings.mako*
│   └── addonshortname_user_settings.mako*
├── tests
│   ├── __init__.py
│   ├── test_model.py
│   └── test_views.py
└── views
    └── ...
```

* optional

3.5.6 StoredObject

All models should be defined as subclasses of `framework.mongo.StoredObject`.

3.5.7 Routes

Routes are defined in a dictionary containing `rules` and an optional `prefix`.

Our url templating works the same way that flask's does.

```
my_route = {
  'rules': [
    Rule(
      [
        '/my/<templated>/path/',  # Note all routes SHOULD end with a forward slash (/
↪)
        '/also/my/<templated>/path/'
      ],
      ('get', 'post'),  # Valid HTTP methods
      view.my_view_function,  # The view method this route maps to
      json_renderer  # The renderer used for this view function, either␣
↪OsfWebRenderer or json_renderer
    )
  ]
}
```

Routes SHOULD be defined in `website.addons.youraddon.routes` but could be defined anywhere

3.5.8 Views

Our views are implemented the same way that flask's are.

Any value matched by url templating (`<value_name>`) will be passed to your view function as a keyword argument

Our framework supplies many python decorators to make writing view functions more pleasant.

Below are a few examples that are commonly used in our code base.

More can be found in `website.project.decorators`.

framework.auth.decorators.must_be_logged_in

Ensures that a user is logged in and imputes `auth` into keyword arguments

website.project.decorators.must_have_addon

`must_have_addon` is a decorator factory meaning you must supply arguments to it to get a decorator.

```
@must_have_addon('myaddon', 'user')
def my_view(...):
  pass


@must_have_addon('myaddon', 'node')
def my_node_view(...):
  pass
```

The above code snippet will only run the view function if the specified model as the requested addon.

Note: Routes whose views are with decorated `must_have_addon('addon_short_name', 'node')` MUST start with `/project/<pid>/...`.

---

website.project.decorators.must_have_permission

`must_have_permission` is another decorator factory that takes a `permission` argument (may be'write','read', or'admin').

It prevents the decorated view function from being called unless the user issuing the request has the required permission.

3.5.9 Logs

Some common log examples

- `dropbox_node_authorized`
- `dropbox_node_authorized`
- `dropbox_file_added`
- `dropbox_file_removed`
- `dropbox_folder_selected`, `github_repo_linked`, etc.

Use the `NodeLog` class's named constants when possible,

```
'dropbox_' + NodeLog.FILE_ADDED
```

Every log action requires a template in `youraddon/templates/log_templates.mako`. Each template's id corresponds to the name of the log action.

3.5.10 Static files for add-ons

Todo: Add detail.

First make sure your add-on's short name is listed in `addons.json`.

addons.json

```
   {
       "addons": [
           ...
           "dropbox",
           ...
       ]
   }

This adds the proper entry points for webpack to build your add-on's static files.
```

The following files in the `static` folder of your addon directory will be built by webpack:

- user-cfg.js : Executed on the user addon configuration page.
- node-cfg.js : Executed on the node addon configuration page.
- files.js : Executed on the files page of a node.

You do not have to include these files in a ''<script>'' tag in your templates. They will dynamically be included when your addon is enabled.

3.5.11 Rubeus and the FileBrowser

For an addon to be included in the files view they must first define the following in the addon's `__init__.py`:

```
HAS_HGRID_FILES = True
GET_HGRID_DATA = views.hgrid.{{addon}}_hgrid_data
```

Has hgrid files is just a flag to attempt to load files from the addon. get hgrid data is a function that will return FileBrowser formatted data.

Rubeus

Rubeus is a helper module for filebrowser compatible add ons.

`rubeus.FOLDER,KIND,FILE` are rubeus constants for use when defining filebrowser data.

`rubeus.build_addon_root`:

Builds the root or "dummy" folder for an addon.

```
:param AddonNodeSettingsBase node_settings: Addon settings

:param str name: Additional information for the folder title

    eg. Repo name for Github or bucket name for S3

:param dict or Auth permissions: Dictionary of permissions for the add-on's content␣
↪or Auth for use in node.can_X methods
```

```
:param dict urls: Hgrid related urls

:param str extra: Html to be appended to the addon folder name

    eg. Branch switcher for github

:param dict kwargs: Any additional information to add to the root folder

:return dict: Hgrid formatted dictionary for the addon root folder
```

Addons using OAuth and OAuth2

There are utilities for add-ons that use OAuth or Oauth2 for authentication. These include:

- `website.oauth.models.ExternalProvider` : a helper class for managing and acquiring credentials (see `website.addons.mendeley.model.Mendeley` as an example)
- `website.oauth.models.ExternalAccount` : abstract representation of stored credentials; you do not need to implement a subclass of this class
- `website.addons.base.AddonOAuthUserSettingsBase` : abstract interface to access user credentials (see `website.addons.mendeley.model.MendeleyUserSettings` as an example)
- `website.addons.base.AddonOAuthUserSettingsBase` : abstract interface for nodes to manage and access user credentials (see `website.addons.mendeley.model.MendeleyNodeSettings` as an example)
- `website.addons.base.serializer.AddonSerializer` & `website.addons.base.serializer.OAuthAddonSerializer`: helper classes to facilitate serializing add-on settings

Deselecting and Deauthorizing

Many add-ons will have both user and node settings. It is important to ensure that, if a user's add-on settings are deleted or authorization to that add-on is removed, every node authorized by the user is deauthorized, which includes resetting all fields including its user settings.

It is necessary to override the `delete` method for `MyAddonUserSettings` in order to clear all fields from the user settings.

```
class MyAddonUserSettings(AddonUserSettingsBase):

    def delete(self):
        self.clear()
        super(MyAddonUserSettings, self).delete()

    def clear(self):
        self.addon_id = None
        self.access_token= None
        for node_settings in self.myaddonnodesettings__authorized:
            node_settings.deauthorize(Auth(self.owner))
            node_settings.save()
        return self
```

You will also have to override the `delete` method for `MyAddonNodeSettings`.

```
class MyAddonNodeSettings(AddonNodeSettingsBase):

    def delete(self):
        self.deauthorize(Auth(self.user_settings.owner), add_log=False)
        super(AddonDataverseNodeSettings, self).delete()

    def deauthorize(self, auth, add_log=True):
        self.example_field = None
        self.user_settings = None

        if add_log:
            ...
```

3.5.12 IMPORTANT Privacy Considerations

Every add-on will come with its own unique set of privacy considerations. There are a number of ways to make small errors with a *large* impact.

General

- **Using** `must_be_contributor_or_public`, `must_have_addon`, **etc. is not enough.** While you should make sure that you correctly decorate your views, that does not ensure that *non-OSF*-related permissions have been handled.
- For file storage add-ons, make sure that contributors can only see the folder that the authorizing user has selected to share.
- Think carefully about security when writing the node settings view ({{addon}}_node_settings.mako / {{addon}}NodeConfig.js}}. For example, in the GitHub add-on, the user should only be able to see the list of repos from the authenticating account if the user is the authenticator for the current node. Most add-ons will need to tell the view (1) whether the current user is the authenticator of the current node and (2) whether the current user has added an auth token for the current add-on to her OSF account.

Example: When a Dropbox folder is shared on a project, contributors (and the public, if the project is public) should only perform CRUD operations on files and folders that are within that shared folder. An error should be thrown if a user tries to access anything outside of that folder.

```
@must_be_contributor_or_public
@must_have_addon('dropbox', 'node')
def dropbox_view_file(path, node_addon, auth, **kwargs):
    """Web view for the file detail page."""
    if not path:
        raise HTTPError(http.NOT_FOUND)
    # check that current user was the one who authorized the Dropbox addon
    if not is_authorizer(auth, node_addon):
        # raise HTTPError(403) if path is a not a subdirectory of the shared folder
        abort_if_not_subdir(path, node_addon.folder)
    ...
```

Make sure that any view (CRUD, settings views...) that accesses resources from a 3rd-party service is secured in this way.

3.6 Migrations

- Migrations go in the `scripts/` directory.

- Use a `main()` function which runs the migration. Avoid import side-effects.
- When possible, include a function that lists the records that will be affected by the migration. These are useful in an interactive session for doing dry runs.

```
def get_targets():
    """Generate the nodes to migrate."""
    return (node for node in Node.find(Q('category', 'nin', Node.VALID_CATEGORIES)))
```

- Use Python's `logging` module for logging output. In addition, use `scripts.utils.add_file_logger` to add a file handler that will add timestamped log file in `website.settings.LOG_PATH`.
- Add tests in the `scripts/tests` directory.

Below is the skeleton of an example migration.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""Script to migrate nodes with invalid categories."""

import sys
import logging

from website.app import init_app
from scripts import utils as script_utils

logger = logging.getLogger(__name__)

def do_migration(records, dry=True):
    # ... perform the migration ...

def get_targets():
    # ... return the StoredObjects to migrate ...

def main(dry=True):
    init_app(set_backends=True, routes=False)  # Sets the storage backends on all␣
↪models
    do_migration(get_targets(), dry=dry)

if __name__ == '__main__':
    dry = '--dry' in sys.argv
    if not dry:
        script_utils.add_file_logger(logger, __file__)
    main(dry=dry)
```

```
from tests.base import OsfTestCase

class TestMigrateNodeCategories(OsfTestCase):

    def test_get_targets(self):
        # ...

    def test_do_migration(self):
        # ...
```

3.7 Troubleshooting Common Problems

This document is intended to serve as a knowledge repository - it should contain solutions to commonly encountered problems when running the OSF, as well as solutions to hard-to-debug issues that developers have encountered that might be seen by others in the course of their work

3.7.1 "Emails not working on my development server"

Solution: You may not have a celery worker running. If you have Celery and RabbitMQ installed (see the README for installation instructions), run `invoke celery`.

Less ideally, you can turn Celery off and send emails synchronously by adding `USE_CELERY = False` to your `website/settings/local.py` file.

3.7.2 "My view test keeps failing"

Solution: You have to reload the database record.

```
def test_change_name_view(self):
    user = UserFactory()
    # Hit some endpoint that updates the user's database record
    res = self.app.post_json('/{}/changename/'.format(user._primary_key),
        {'name': 'Freddie Mercurial'})
    user.reload()  # Make sure object is up to date
    assert_equal(res.status_code, 200)
```

3.7.3 ImportError: No module named five

Celery may raise an exception when attempting to run the OSF tests. A partial traceback:

```
Exception occurred:
  File "<...>", line 49, in <module>
    from kombu.five import monotonic
ImportError: No module named five
```

3.7.4 error: [Errno 61] Connection refused‘ is raised in ampq/transport.py

Solution: You may have to start your Rabbitmq and Celery workers.

```
$ invoke rabbitmq
$ invoke celery_worker
```

3.7.5 Error when importing uritemplate

If invoking assets or server commands throw an error about uritemplate, run the following to resolve the conflict:

```
pip uninstall uritemplate.py --yes
pip install uritemplate.py==0.3.0
```

and then re run the command that failed.

3.7.6 Using PyCharm's Remote Debugger

Some debugging tasks make it difficult to use the standard debug tools (i.e. pdb, ipdb, or PyCharm's debugger). Usually this is becuase you're running code in a way where you don't have ready access to the process's standard in/out. Examples of this include:

- celery tasks
- local testing/debugging using uWSGI

One way to debug code running in these kinds of enviornments is to use the PyCharm remote debugger. Follow the JetBrains documentation for creating a new run configuration for the remote debugger: https://www.jetbrains.com/pycharm/help/remote-debugging.html. At some point you may be required to add pycharm-debug.egg to your system's PYTHONPATH. The easist way to do this is to modify your ~/.bash_profile to automatically append this module to the python path. This looks like:

```
export PYTHONPATH="$PYTHONPATH:<your_path_to>/pycharm-debug.egg"
```

To apply these changes to the current bash session, simply

```
source ~/.bash_profile
```

When you start the remote debug server in PyCharm you will get some console output like:

```
Use the following code to connect to the debugger:
import pydevd
pydevd.settrace('localhost', port=54735, stdoutToServer=True, stderrToServer=True)
```

So to use, simple copy paste the bottom two lines wherever you need to run a debugger. In celery tasks for example, this often means inside a task definition where it would be otherwise impossible to step into the code. Trigger whatever is needed to queue the celery task, and watch the PyCharm console to see when a new connection is initiated.

Happy remote-debugging.

3.8 Ops

In progress

3.8.1 Common deployment tasks

Generating a new SSL certificate

- **Generate a certificate signing request (see instructions from this post)**
  - goto osf:/opt/certs/namecheap
  - `openssl genrsa -des3 -out osf.io.key 2048`
  - `openssl rsa -in osf.io.key -out osf.io.key.nopass`
  - **`openssl req -new -key osf.io.key.nopass -out osf.io.csr`**
    - fqdn: osf.io
    - don't enter "challenge password"
- **Get signed certificate**

  - submit CSR to NameCheap
  - follow verification email
  - download and expand zip file of certs
  - `cat osf_io.crt COMODORSADomainValidationSecureServerCA.crt COMODORSAAddTrustCA.crt AddTrustExternalCARoot.crt > osf.io.bundle.crt`

- **On staging**
  - copy osf.io.bundle.crt to /opt/certs/namecheap
  - **edit `/opt/nginx/sites-enabled/000-osf`**
    - `ssl_certificate /opt/certs/namecheap/osf.io.bundle.crt;`
    - `ssl_certificate_key /opt/certs/namecheap/osf.io.key.nopass;`
- **On production**
  - goto linode nodebalancer config
  - edit production settings
  - paste osf.io.bundle.crt into "Certificate" field
  - paste osf.io.key.nopass into "Private Key" field

Upgrading Unsupported releases of Ubuntu

- EOLUpgrades
- How to install software or upgrade from old unsupported release? (AskUbuntu)

NOTE: The command from the AskUbuntu answer needs slight modification to include replacement of `us.archive.ubuntu.com`:

```
sudo sed -i -e 's/archive.ubuntu.com\|security.ubuntu.com\us.archive.ubuntu.com/old-
↪releases.ubuntu.com/g' /etc/apt/sources.list
```

NOTE: When prompted if you want to replace `/etc/mongodb.conf` and `/etc/nginx/nginx.conf`, etc., press `X` to enter the shell and back these files up (`sudo cp /etc/mongodb.conf /etc/mongodb.conf.bak`)

Migrating to a new machine

Todo: This is incomplete. Add final steps and clean this up.

- On Linode dashboard, go to the Linode you want to restore *from*.
- Go to Backups -> Daily Backups.
- Click "Restore to this Linode" next to the Linode you want to restore *to*.
- Once restoration is complete, resize the data image.
- On the new machine, add a new SSH key pair.

```
# Replace "sloria" with your username
$ ssh-keygen -t rsa -C "sloria-osf"
```

- Copy the new public key to the old machine at `/home/<your-username>/.ssh/authorized_keys`
- On the new machine, edit /root/.ssh/config with the correct SSH settings.

```
Host osf-old
    HostName <ip-of-old-linode>
    User sloria
    IdentityFile /home/sloria/.ssh/id_rsa
```

- rsync /opt/data/mongodb.

```
# On the new machine
$ sudo rsync -vaz --delete osf-old:/opt/data/mongodb /opt/data
```

Note: You'll probably want to use `screen` or `nohup` to run this as a background process.

- rsync /opt/data/uploads.

```
$ sudo rsync -vaz --delete osf-old:/opt/data/uploads /opt/data/
```

put up osf_down page,private ips on node balancer, restart new server then old

3.9 Running Google Analytics Locally

This section describes how to set up and test Google Web Analytics features locally.

3.9.1 Getting a Google Analytics Account

Open a web browser and navigate to Google Analytics.

Click **Access Google Analytics**. If you have not used Google Analytics before, you will be prompted to sign up. Follow the process and confirm your account.

Click **Admin** on the nav bar.

Click the **Property** drop down and select **Create New Property**.

Fill out **Website Name** and **Website URL** – they don't need to point to real URLs. In this example the former will be *osf analytics testing* and the latter will be **test.osf.io**

Once completed, click **Get Tracking ID** and you will be redirected to a page containing the tracking info you will need.

Take note of copy the **Tracking ID**. It should look similar to *UA-43352009-1*.

3.9.2 Update Your Local local.py

Ensure that you have set up your `website/settings/local.py` file as outlined in *Setting up the OSF* .

Add `GOOGLE_ANALYTICS_ID = <your_tracking_id>` to your `website/settings/local.py` file.

Note: Now any events that trigger Google Analytics, currently or new things you implement, will be sent to the property we set up in the previous section.

3.9.3 Viewing Live Events Within Google Analytics

Click **Home** on the nav bar.

Note: You will see one, or more if you have previously created properties, folders. The descriptions match the **Website Name** of the associated property.

Click on the property you created, **osf analytics testing** in your example.

Click **All Website Data** and you will be redirected to an overview page for this property.

On the left hand navigation bar, click **Real-Time** -> **Overview**.

(Optional) Follow the **ANALYTICS EDUCATION** if you are unfamiliar with Google Analytics.

Click **Events** flag underneath the **Real-Time** section of the left hand navigation bar.

Note: From here you can view live events (events received within the last 30 minutes) triggered by either you or your test code from your local environment.

3.10 How to add a preprint provider

3.10.1 Requirements

New preprint providers must have the following:

- A name
- A square logo
- A rectangular/wide logo
- 3 unique colors
- A list of acceptable subjects (taxonomies)
- A list of acceptable licenses
- Emails for:
  - Contact (defaults to `contact+provider_name@osf.io`)
  - Support (defaults to `support+provider_name@osf.io`)

Recommended:

- A Twitter account
- A Facebook Page
- An Instagram account

3.10.2 Creating a new Preprint Provider

1. Update the Populate Preprint Providers script
2. Format and save images to the in the ember-preprints assets directory
3. Add the config entry
4. Create the stylesheet

osf.io Updates

Create a new feature branch in the **osf.io** repository `git checkout develop && git checkout -b feature/add-preprint-provider-provider_id`

ember-preprints Updates

Create a new feature branch in the **ember-preprints** repository `git checkout develop && git checkout -b feature/add-preprint-provider-provider_id`

Formatting the images

In the ember-preprints repository: `public/assets/img/provider_logos`. Commit the original images (before edits) to the repository, but don't reference the original images in the CSS. SVG images are preferred, but if the provider does not have those available, use or convert to the PNG format.

You'll need:

- White foreground with transparent background (Used on the main OSF Preprints landing page, under the *Preprint Services* Section)
  - width: <= `350px`
  - height: `140px`
- Black foreground with transparent background (Used on the OSF Preprints discover page, under *Partner Repositories*)
  - width: <= `350px`
  - height: `120px`
- Square logo with a transparent background (can have color foreground)
  - width: `48px`
  - height: `48px`
- Rectangular/wide logo with a transparent background (can have color foreground, this can be the white foreground image)
  - width: <= `600px`
  - height: `140px`
- A *sharing logo* that will be displayed on social media sites
  - width: >= `200px`, `1500px` preferred
  - height: >= `200px`, `1500px` preferred

You may need to edit the images to meet the requirements. Use `brew cask install gimp` to install gimp or use Pixlr.

Optimitize the images with Optimizilla or a similar service. See the Google Image Optimization Guide

Adding an entry in the config

In `config/environment.js`, there will be a `PREPRINTS` object and a `providers` array. You will need to add another object to that `providers` array.

If adding a provider domain, you'll need to run `sudo ./scripts/add-domains.js`

```
{
    id: 'provider_id', // This must match the ID in the OSF Repo
    domain: 'provider.org', // If using a domain. Otherwise, delete this line
    logoSharing: {
        path: '/assets/img/provider_logos/provider_id-sharing.png', // The path to␣
↪the provider's sharing logo
        type: 'image/png', // The mime type of the image
        width: 1500, // minimum 200, 1500 preferred (this is the width of the image,␣
↪in pixels)
        height: 1500 // minimum 200, 1500 preferred (this is the height of the image,␣
↪in pixels)
    },
    permissionLanguage: 'provider_permission_language'
}
```

Adding permission language to the footer text

The branded preprint partners need to show permissions to use content/titles from the owner institutions/organizations. For example, engrXiv, SocArXiv, and PsyArXiv are using the -rXiv extension with permission from Cornell and there is a need to have a language on their pages stating such.

Adding an entry in the translation

In `translation.js`, there will be a permission-language entry where you will need to add the provider permission language.

```
'permission-language': {
    arxiv_trademark_license,
    arxiv_non_endorsement: `${arxiv_trademark_license} This license should not be␣
↪understood to indicate endorsement of content on {{provider}} by Cornell University␣
↪or arXiv.`
}
```

Note that if the permission language is expecting to be used fully or partially by other providers then it is preferable to be defined as a constant at the beginning of the `translation.js` file. The const can be later re-used within the permission-language entry.

```
const arxiv_trademark_license = 'arXiv is a trademark of Cornell University, used␣
↪under license.';
```

Creating the stylesheet

The basic stylesheet must be named `app/styles/brands/provider_id.scss` and contain the following:

```
@import 'brand';

@include brand(
    #ABCDEF,                                  // Color, theme color #1 (header␣
↪backgrounds, hover backgrounds)
    white,                                    // Color, theme color #2 (text color␣
↪mostly, usually white or black)
    #012345,                                  // Color, theme color #3 (navbar color,␣
↪preferably a dark color)
    #6789AB,                                  // Color, theme color #4 (used in link␣
↪colors)
    black,                                    // Color, theme color #5 (text color that␣
↪contrasts with #2, usually black or white)
    $logo-dir + 'engrxiv-small.png',          // String, path to the rectangular␣
↪provider logo
    $logo-dir + 'engrxiv-square-small.png', // String, path to the square provider␣
↪logo
    true,                                     // Boolean, whether to use the white␣
↪share logo or not
    false,                                    // Boolean, whether to use theme color 4␣
↪or theme color 2 for the navbar link color
    true                                      // Boolean, whether to use the contracts␣
↪link color (theme color 4)
);

// Add any custom styles or overrides here
```

You may need to manipulate the colors and options to get them to look good. Avoid overriding styles, if at all possible.

Open Pull Requests

Open pull requests against osf.io and ember-preprints with your changes. Be sure to cross-reference in the PR description that it requires the other PR. If the PR includes adding a domain, Add a "Notes for Reviewers" section with a reminder to run the add-domains script. Add notes for QA that include screenshots of the newly added provider

In your PR against osf.io, add a section called "Notes for Deployment" with a reminder to request an API key from SHARE. This is necessary, because the provider's preprints will not be indexed by SHARE without the API Key.

3.10.3 CAS Support for Login and Sign Up

Create a ticket in CAS Board with "Login and Sign Up Support: <the name of the preprint provider>" as the summary. Basic features are guaranteed and extra ones are welcome. Make this ticket block the OSF or EOSF tickets for this provider.

Basic Features

1. Register the preprint provider to CAS Registered Service.
2. Whitelist the provider's external domain to OSF Authentication Logic.
3. Customize the login page (CAS) and the sign up page (OSF).

Extra Features

Please add other requirements in the description.

Resources To Provide

1. Preferred display name: e.g. `PsyArxiv`.
2. The default, black and colored logo images (if available).
3. Preferred CSS background color: the main background color of the home page.
4. OSF domain and external domain: e.g. `osf.io/preprints/psyarxiv/` and `preprints.psyarxiv.org/`

3.10.4 DevOps

If the provider is using a domain, create a DevOps ticket to update the proxy redirects and for them to contact the domain admin to get the domain pointed at the correct IP address. Include the contact information from the product team.

CHAPTER 4

Getting Help

4.1 Resources

4.1.1 Flowdock

We use Flowdock for group and 1-and-1 online chat. It's a great place to ask questions!

CHAPTER 5

Contributing To These Docs

5.1 Getting Started

Contributions to this documentation are welcome. Documentation is written in reStructured Text (rST). A quick rST reference can be found here. Builds are powered by Sphinx and are hosted by ReadTheDocs.

5.1.1 Fork, clone, install

First, fork the COSDev repo, clone it, and install the requirements (requires Python 2.7 or 3.4 with pip):

```
# After forking CenterForOpenScience/COSDev
$ git clone https://github.com/<your_github_username>/COSDev.git
$ cd COSDev
$ pip install -r requirements.txt
```

Be sure to replace `<your_github_username>` with your Github username.

5.1.2 Build the docs

To build docs:

```
$ invoke docs -b
```

The `-b` (for "browse") automatically opens up the docs in your browser after building. Alternatively, you can open up the `docs/_build/index.html` file manually.

Autobuilding on File Changes

You can use `sphinx-autobuild` to automatically build the docs when you change a file in the `docs` directory.

To install `sphinx-autobuild`:

```
$ pip install sphinx-autobuild
```

You can now start the livereload server with:

```
$ invoke watch
```

Point your browser to http://localhost:8000 to see your docs.

5.1.3 Send a PR!

Once you are done making your edits, send a pull request on Github to the COSDev repo.

5.1.4 Header Conventions

Use the following underlining conventions for heading levels:

- = for h1
- * for h2
- – for h3
- ^ for h4

5.2 Todo List

Below is a list of all the TODO items in this documentation. Please *help out* and send a pull request!

Todo: Add detail.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/cosdev/checkouts/latest/docs/osf/addons.rst, line 281.)

Todo: Document how to use mako variables in JS modules (`contextVars`)

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/cosdev/checkouts/latest/docs/osf/js_modules.rst, line 163.)

Todo: This is incomplete. Add final steps and clean this up.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/cosdev/checkouts/latest/docs/osf/ops.rst, line 56.)