SLUG meetingifier Documentation

Release 1.5

SLUG team

May 29, 2012

CONTENTS

Contents:

CHAPTER

ONE

README - OVERVIEW FOR DEVELOPERS

1.1 More documentation

We're starting to add more documentation into the `doc` folder.

If you would prefer to read formatted HTML (highly recommended), simply `make doc` and then look under `doc/_build/html`

This `README` is automatically generated and placed at the top level as part of that documentation build. If you're editing this at the top level, your changes will shortly be clobbered. Edit `doc/README.rst` instead.

You can browse the latest checked-in version of the docs at http://slug-usergroup.rtfd.org

You can check the status of the latest checking thanks to our free CI service from Travis

1.2 Initial Configuration

To get the code and dependencies:

```
git clone git@github.com:sydney-linux-user-group/slug.git
cd slug
make install
```

1.3 Running a test server

Simply `make serve`; this will configure a virtualenv, download and install dependencies (inside the virtualenv; your system will not be touched); and a test server will be started.

If this is your first time running `make serve` you'll be prompted to provide a username and password for an admin account.

1.4 Running tests

Simply `make test`

1.5 Coding standards

In general, we follow PEP-8. `make lint` will tell you in detail about all the things we need to fix.

1.6 Production Deployment

1. To account for differences between the dev and prod infrastructure, we have a `private` repo which needs to be checked out. Exactly where the `private` repo comes from will be specific to your deployment.

   To see where this is used and get an idea of what you can override, search for `private` in settings.py

2. The SLUG deployment uses one user to deploy the code, and another user to run the code:

   ```
   zhasper@tridge:~$ grep slug /etc/passwd /etc/group
   /etc/passwd:slug:x:1001:1001::/home/slug:/bin/bash
   /etc/passwd:slug-run:x:1005:1005::/home/slug-run:/bin/sh
   /etc/group:slug:x:1001:
   /etc/group:slug-run:x:1005:slug
   ```

3. Usually, the run user should only need read access to the files you've checked out. If there are specific files or directories that the run user needs to write to [1], simply use `chgrp slug-run $FILE; chmod g+w $FILE` to make them accessible to both the deploy and run users. If this needs to be persisted across deployments, you may have to take care of this in your deploy script.

4. For deployment, I simply `ssh slug@localhost -A`; this turns on agent forwarding so that my usual SSH keys are used to pull the code from bitbucket. For automated deployments, you can create a passwordless key stored for the `slug` user to use and upload it to Github as a deploy key

5. To do the deployment, I use this simple script, which would benefit from a ton more checking:

   ```
   #!/bin/bash

   TIMESTAMP=$(date +%Y%m%d-%H%M)
   cd ~/django
   git clone -b django git@github.com:sydney-linux-user-group/slug.git ${TIMESTAMP}
   cd ${TIMESTAMP}
   make install
   make private
   make prepare-serve
   ```

   This makes it relatively easy to revert to an earlier version of the code.

   Note: Depending on your database setup you may have to do some extra work here. If using sqlite you will want to point the new checkout at your existing sqlite database somewhow. If you're using `mysql` or `postgres` (configured in `private/settings.py`) you'll probably want to take a backup of the existing database before running `make prepare-serve` in case you need to roll back the database changes.

   ---

6. We've chosen to run the app inside Green Unicorn, and have it started by `upstart`:

   ```
   slug@tridge:~/django/current$ cat /etc/init/slug.conf
   description "SLUG Django instance"
   start on runlevel [2345]
   stop on runlevel [06]
   respawn
   ```

[1] For instance, if you're using `sqlite` as the database, the run user will need permission to write to the `sqlite` file

```
respawn limit 10 5
script
  cd /home/slug/django/current
  bin/gunicorn_django -u slug-run -g slug-run
end script
slug@tridge:~/django/current$ ls -l /etc/init.d/ | grep slug
lrwxrwxrwx 1 root root    21 2012-02-04 23:24 slug -> /lib/init/upstart-job
```

This solution is not perfect. `upstart` doesn't kill `gunicorn` properly, so a restart involves killing a few processes before using `sudo service slug start`. I need to find time to figure out how to improve this.

7. We've chosen to use nginx as a frontend, and to serve static files. Only a few changes from the default config are needed to accomplish this:

```
# path for static files
root /home/slug/django/current/usergroup/;

location /static/ {
        alias /tmp/slug-static/;
}

location /admin/media/ {
        root /home/slug/django/current/lib/python2.6/site-packages/django/contrib;
}

location / {
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header Host $http_host;
    proxy_redirect off;

    proxy_pass   http://localhost:8000/;
```

`/tmp/slug-static` is stipulated as the STATIC_ROOT in `settings.py`. We should really get around to fixing this - it just needs to be a location that the deploy user can write to and the user running nginx can read from.

CHAPTER

TWO

HITLIST - THINGS WE NEED TO TEST

2.1 Event creation and publication

2.1.1 Create an event

- Fixtures:
  - Single admin user
- Login as an administrator
- Click “add event”
- Choose a date and template
- Add a title
- Submit
- Verify that we’ve been redirected appropriately.

Implemented in `TestCreateEvent.test_create_event()`

2.1.2 Newly created events are ready to publish

- Fixtures:
  - Single admin user
  - One unpublished event
- As an admin, browse to the Events List page
- Find the submit button associated with the event
- Verify that the button says “Publish Event”

Implemented in `TestPublishEvent.test_ready_to_publish()`

2.1.3 Publish an event

- Fixtures:
  - Two unpublished events
  - One admin user

- Install a fixture with two unpublished events
- Load the events page as an admin
- Publish one event
- Verify that events page shows that event as being published (ie, ready to announce)
- Verify that events page does not show the other event as being published

Implemented in `TestPublishSomeEvents.test_unpublished_events_show_as_unpublished()`

2.1.4 Normal users cannot see unpublished events

- Install a fixture with at least one published and one unpublished event
- Load the front page as a normal user
- Verify that all published events are visible
- Verify that all unpublished events are not visible

Implemented in `TestEventVisibility.testVisibilityAsOrdinaryUser()`

2.1.5 Anonymous users cannot see unpublished events

- Install a fixture with at least one published and one unpublished event
- Load the front page as a an anonymous
- Verify that all published events are visible
- Verify that all unpublished events are not visible

Implemented in `TestEventVisibility.testVisibilityAsAnonymousUser()`

2.1.6 Announce an event

- Install a fixture with at least 1 published unannounced event
- Announce the event
- Django outbox should contain one email
  - Validate from address
  - Validate to address
  - Check that the subject is the meeting name
  - Check that if the meeting is re-announced, the second email has "Updated: " plus the meeting name.
  - Check that body has nothing that looks like a template tag
  - Check that title is present in body
  - Check that date and time are in body

Implemented in `TestEventEmail`

2.1.7 Edit an announced event

- Install a fixture with at least one 1 published announced event
- Load the event detail page as an admin
- Edit the details of the event and submit changes
- Load the event list page as an admin. Verify that the event now shows ready to be re-published
- Load the event details page as an admin. Verify that the HTML and Plaintext views ahve been updated.
- Re-publish the event, and verify that the updated information is used to generate the email.

Implemented in `TestEventEditing`

2.1.8 Re-publish an event

- Install a fixture with one published announced edited event ready for re-publication
- Load the event list page as an admin
- Verify that the event shows as being ready for re-publication

Implemented in `TestEventEditing.test_event_ready_for_republish()`

- Load the event list as an anonymous user
- Verify that the event list shows the old issue details

Done in `TestEventEditing.test_event_shows_old_details_for_anonymous_user()`

- Republish the event
- Load the event list page as an admin
- Verify that the event list page shows the event being ready for re-announcement

Done in `TestEventEditing.test_republished_event_shows_as_ready_for_reannouncement()`

- Load the event list as an anonymous user
- Verify that the event list shows the new issue details

Done in `TestEventEditing.test_republished_event_displays_for_anonymous_user()`

2.1.9 Re-announce a re-published event

- Install a fixture with one published announced edited re-published event ready for re-announcement
- Load the event list page as an admin
- Verify that the list shows the event as ready for re-announcement
- Re-announce the event
- Verify that the event list page shows as having been re-announced
- Django outbox should contain one email
  - Validate from address
  - Validate to address
  - Check that body has nothing that looks like a template tag

  - Check that tile is present in body
  - Check that date and time are in body
  - Validate that the subject indicates that this is a re-announcement

2.2 Create and edit talk offers

2.2.1 Anonymous user clicks "offers a talk"

- Browse the main page as an anonymous user
- Click "Offer Talk"
- Get redirected to the login page

Done in `TestAnonymousUserClicksOffer.test_anonymous_user_clicks_offer_talk()`

2.2.2 Logged-in user offers a talk

- Browse the main page as a logged-in user
- Click "Offer Talk"
- A second window opens with the "Offer Talk" form

Done in `TestLoggedInUserClicksLogin.test_logged_in_user_clicks_offer_talk()`

- Enter values into all fields
- Submit the form
- Verify that the window has redirected to "/offer/add#prevoffers"
- Verify that the entered talk details show in the list of previous offers

2.2.3 Admin looks at list of talk offers

- Install a fixture with at least one offered talk
- As an admin, browse the list of offers
- Verify that the offered talks are listed

2.2.4 Admin edits agenda for a meeting

- Install a fixture containing at least one offered talk and one published event
- As an admin, load the detail page for an event
- Click on the "Agenda" tab
- Drag a talk from "All Offers" to "Agenda Items"
- Verify that the talk was dropped into Agenda Items; and has turned orange
- Click on the Source tab and submit the form
- Click on "Formatted Plaintext" and "HTML" and verify that the talk is shown in the agenda

2.3 Normal user interaction

Assume starts with "Able to"

2.3.1 Login with username/password

- Admin user can login with username/password
- Valid existing user can login with username/password
- Invalid user can't login username/password

Implemented in `TestValidAdminLogin` Implemented in `TestInvalidUserLogin` Implemented in `TestValidNonAdminLogin`

2.3.2 Login with OpenID

- Admin user can login
- Valid existing user can login
- Invalid OpenID goes to Create a new account option below.

2.3.3 Create a new account using OpenID

- Including checking of the email address from non-trusted providers.

2.3.4 Create a new account using username/password

- Including checking of the email address.
- testFailOnMissingField
- testFailOnNonMatchingPasswords
- testFailOnInvalidEmailAddress
- testFailOnExistingUsername
- testFailOnExistingEmail
- testRegistrationSuccess

Implemented in `TestRegister`

2.3.5 Change/add local password

2.3.6 Add an OpenID account to an existing account

2.3.7 Remove an OpenID account to an existing account

2.3.8 Merge two accounts (OpenID/OpenID, OpenID/login)

2.3.9 Say Yes to attending an event

2.3.10 Say No to attending an event

2.3.11 Bring friends to attending an event

2.3.12 get to via links

- map
- webirc on freenode
- mailing list
- calendar

2.3.13 “quick tweet” about attending an event

2.3.14 “quick facebook post” about attending an event

2.3.15 to sign up to the mailing list

CHAPTER

THREE

TESTS – HOW WE KNOW THAT STUFF AIN'T DONE BROKE.

3.1 Selenium tests – Test client-side behaviour

3.1.1 `base` - Base classes and common code for selenium tests

Tip: While it's very cool, testing with Selenium is slow due to the need to fire up a Firefox process and wait for page rendering. Selenium should only be used when testing client-side behaviours.

Warning: If your computer doesn't have a working internet connection, Firefox will switch to offline mode and not even try connecting to localhost. This breaks the tests. Good job, Firefox!

3.1.2 `create_and_manipulate_meetings_test`

3.1.3 `login_test` - Walking through the login process

3.1.4 `register_and_respond_test` - New user registration and handling

3.2 Django tests – Test views and backend functionality only

3.2.1 `email_test` – Test that emails get sent appropriately

Note: Currently very incomplete

3.2.2 `login_test` – Test that logins work as expected

3.2.3 `event_manipulation_test` – Create and manipulate events

CHAPTER

FOUR

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*