Python GSoC: Build a multi-user Blogging Platform with additional features to manage GSoC@PSF

About Me

1. *Name:* Sounak Pradhan
2. *Nickname:* sounak98 (on irc, github, and most other platforms)
3. *Timezone:* India Standard Time (GMT +0530)

A development enthusiast, I am highly interested in the workings of different systems. Systems fascinate me and I like building robust systems from scratch.

I have been working with django and other web frameworks since the past two years and have used it as a part of numerous class assignments, course projects and different internship tasks as well.

Code Contribution

I have been working on the project from the start of February and have made substantial contribution to it by now. I have made a total of 14 merged pull requests. All of them can be found on this link. Some major ones are also listed below out of which the first three were priority issues.

- #104 Create a page for listing all the current blogs and also archived blogs of previous years
- #79 Permission setup for student blogs
- #45 Extended the toolbar of ckeditor in `djangocms-text-ckeditor` to include plugins that embed YouTube videos and insert, drag and resize images
- #32 Added a management command `runcron` for running scheduled tasks (like sending emails, irc messages, etc.)
- #20 Added an IRC Bot which can PM commands to different other bots
- #18 Added a filter which selects the migrations according to the set database, thus only applying the correct migrations to a database

Project Information

Sub-Org Name: Python GSoC
Mentors: Botanic, dotgourav, Meflin, Warthog9

Project Abstract

Every year more than 100 students apply for GSoC under the umbrella organization PSF. Currently there is a multi-user blogging website using WordPress CMS hosted for the students to publish their weekly blogs and a static landing page for reaching out to people for more information about this program.

This project aims to build a platform which allows smooth management of the GSoC program at PSF every year and also ties everyone associated with it to PSF, so that their work or they themselves can help out others in future.

Project Goals

- Admins should be able to add the GSoC timeline for a current year, accept/reject Sub-Orgs or request for changes and have superuser permissions
- Sub-Org admins should be able to apply for participation in Python-GSoC and submit list of mentors
- Mentors should receive an invite to join, should be able to accept the community guidelines
- Students should be able to publish and edit their blogs, submit their accepted proposals
- Landing pages should be hosted for reaching out to students and Sub-Orgs with various information about the application and the whole GSoC process
- Reminders should be given to all users for upcoming deadlines for different tasks

Implementation

The plan is to build a Content Management System (CMS) which will allow students to write and publish their own blog posts. We plan on using django to build this app, which is a high-level Python framework. We are using django CMS for building the CMS, and for the blogging platform we are using a plugin on top of django CMS called aldryn-newsblog. Currently, the app uses SQLite for managing the database. Administration is managed by django CMS admin, and a minimal front-end is built using django templates and a lightweight CSS framework known as pure.css.

We are planning the development of this application with the timeline of the GSoC program for this year. In this way the GSoC students of this year working under the umbrella org PSF will be able to use it simultaneously while we are developing it. This will allow us to get active feedback from the students using it, helping us make the application more robust.

A user can have multiple UserProfiles. One UserProfile ties a user to a particular year's PSF GSoC program and a Sub-Org. The different roles through which a user can be tied to a PSF GSoC are Sub-Org Admin, Mentor or Student. There will also be admins who have superuser access to the whole platform. According to the different roles, a user will have different tasks to perform. For example, a user who is tied to this year's PSF GSoC as a student under the Sub-Org PSF GSoC Team will have to upload reports and write blogs in a timely fashion.

```
class UserProfile(models.Model):
    ROLES = (
        (0, 'Others'),
        (1, 'Suborg Admin'),
        (2, 'Mentor'),
        (3, 'Student')
    )

    user = models.ForeignKey(User, on_delete=models.CASCADE)
    role = models.IntegerField(name='role', choices=ROLES, default=0)
    gsoc_year = models.ForeignKey(GsocYear, on_delete=models.CASCADE, null=True,
blank=False)
    suborg_full_name = models.ForeignKey(SubOrg, on_delete=models.CASCADE,
null=True, blank=False)
    accepted_proposal_pdf = models.FileField(blank=True, null=True)
```

They will also be notified about upcoming deadlines for submission of reports and blogs. Sending these notifications through mails and IRC will be taken care of by a Scheduler which will be used to schedule non-time sensitive or async tasks (ex: sending mails, irc messages, among other things). Along with the notifications being sent on mails and over IRC, users would also be notified on their profile page. This will be taken care of by the `Notification` model.

Aldryn-newsblog doesn't have a lot of features that we would ideally need. For example, one of them is handling permissions for a multi-user blogging site. We are planning to implement these features by

overriding the necessary functions like `user_has_add_permission` and `populate`.

```
def user_has_add_permission(self, user, **kwargs):
    """
    Return True if the current user has permission to add an article.
    :param user: The current user
    :param kwargs: Ignored here
    :return: True if user has add permission, else False
    """
```

```
def populate(self):
    """
    Populates the different buttons in the toolbar according
    to permissions
    """
```

Django provides base classes for writing unit tests and integration tests (ex: `SimpleTestCase` for writing unit tests which gives us options to isolate apps while performing tests, manages db setup, teardown and more). We will be using these classes for writing unit tests. We will also be using `Selenium WebDriver` for multi-browser intergration testing.

Timeline

- **Community Bonding Period**
    - Finishing up the pending issues from the application period
    - Updating to Django CMS 3.7 & Django 2.2
    - Add test frameworks such as `Travis`, `pylint` to test PRs before commiting
    - Refactoring and cleaning up the code written by different applicants
- **Week 1** (May 27)
    - Creating the `Notification` model which will handle notifications for the users (ex: Reminders for submitting reports or writing blogs to students)
        - The model will have a many-to-many link with the `UserProfile` model, a `notification_message` field and an `expiry` field. The implementation will be somewhat as mentioned below:

          ```
          class Notification(models.Model):
              user = models.ManyToManyField(UserProfile, null=False)
              message = models.TextField(null=False)
              expiry = models.DateTimeField(null=False)
          ```

    - Creating a panel in the *My Profile* section which will display all the notifications according to their priorities
        - This page will display objects from the `Notification` with the filter `Notification.objects.filter(user=userprofile).filter(expiry__gte=today)` where `userprofile` is the current year's `UserProfile` object for a user and `today` is the current server date and time
        - This page will have basic modals to change password and edit basic details of a user (Name, Profile Picture, etc)
        - It will also allow a user to view details of the past `UserProfile` s (ex: display link to submitted proposals in case he/she was a student in the past)
    - Creating a form for the admin to fill out different dates of the current GSoC program for

sending out notifications to various users

- This will add some fields to the `GsocYear` model as shown below:

```
class GsocYear(models.Model):
    gsoc_year = models.IntegerField(name='gsoc_year')
    suborg_application_open = models.DateTimeField(null=False)
    suborg_application_close = models.DateTimeField(null=False)
    suborg_acceptance_announce = models.DateTimeField(null=False)
    student_application_open = models.DateTimeField(null=False)
    student_application_close = models.DateTimeField(null=False)
    student_acceptace_announce = models.DateTimeField(null=False)
    phase_1_eval_open = models.DateTimeField(null=False)
    phase_1_eval_close = models.DateTimeField(null=False)
    ...
```

- These fields will be used to add notifications and schedule other tasks using the scheduler, and also auto-generate the sharable Google Calendar

- **Week 2** (Jun 3)
  - Creating the Sub-Org admin form where they fill out the Sub-Org details, with mentors (email) associated with the Sub-Org
    - This form will be made using `forms.Form`, and will have fields like `name`, `link`, `details`, `mentors`, `irc_channel`
    - The form will save to the updated `SubOrg` model. This model will also have a `BooleanField` `accepted` which will be set to `True` when the admin accepts the SubOrg. The unaccepted ones can be deleted using the scheduler model
  - Sending out invites to all the mentors and letting them use their registration links to fill out personal details and accept the community guidelines
    - The registration slugs will be generated using JWT to have the email of the mentor which will be prefilled and uneditable in the mentor registration form
    - The mentor will be required to accept the community guidelines to register himself/herself
  - Adding comments and reactions to blogs using `Comment`, `Reaction` models
    - `parent` field of `Comment` will handle replies to a comment

```
class Comment(models.Model):
    user = models.ForeignKey(User, null=True)
    blog = models.ForeignKey(Article)
    message = models.TextField(null=False)
    parent = models.ForeignKey(null=True)
    created_at = models.DateTimeField(null=False)
    updated_at = models.DateTimeField(null=True, blank=True)
```

    - The `user` can be set to `null` to make an anonymous comment
    - The `parent` field will be used to handle replies to a comment
    - The `Reaction` model will have a `user`, `blog`, and `type` fields. The `type` field will have choices from multiple types of reactions (like happy, sad, thankful, etc)
  - Adding UI elements for rendering the comments and reactions below each blog
  - Adding preventive measures to detect spams in comments
    - Will be using reCAPTCHA v3 and keep a threshold to allow users to comment
    - Also will have basic content filtering to check for abusive language in comments

- **Week 3** (Jun 10)
  - Adding Sub-Org specific links to the Sub-Org model and auto-generating a Sub-Orgs page

with the links and a brief description of the Sub-Org
        - Auto-generating the Sub-Org page by fetching Sub-Org specific details from the `Sub-Org` model
        - Each Sub-Org will have a link to the Sub-Org's website and also a brief description about the Sub-Org
    - Moving these pages to archive when the current GSoC ends
        - When a new year starts, changing the urls to move them into archive
        - The `Sub-Org` model will also have a `gsoc_year` field. A new django app will be created, which will list out the current year's suborgs on `/` and will list out older years' suborgs on `/archive/<year>`
        - This app can be hooked to the `Sub-Org` page using CMS Apphook
    - Adding a custom form in the admin page for adding selected students and assigning their mentors to them
        - The form will be made using `forms.Form` with fields `student_email`, `mentor`, `project_name`
    - Sending out registration links to the invited students to join and fill out their details
        - The students will receive an email with the registration link which will contain a slug
        - The slug will be generated using JWT with the email, project name and mentor primary key as the payload
        - When the student tries to register using the link, the data in the payload will be pre-filled in the form and will be uneditable

- **Week 4** (Jun 17)
    - Adding a plugin to CKEditor for dragging and dropping images (currently only accepts URLs which is pretty inconvenient)
        - Can use the plugin Image Uploader and Browser, although this seems to have a lot of bugs and is probably not compatible with the current version of CKEditor
        - As an alternative, a plugin can be created which would convert an image into base64 and embed it using html img tag
    - Adding a plugin to CKEditor for attaching files to the blog posts
        - Will use the plugin Attachments
    - Adding a plugin to CKEditor for adding code snippets
        - Will use the plugin Code Snippet GeSHi
    - Adding a page specific notification block
        - This will be used so that students will be able to give some information (ex: links to their personal blogs) over their blogs
        - This will be implemented by adding the following field to the `Notification` model

          ```
          page = models.ForeignKey(Page, null=True)
          ```

    - Letting only those users who have the permissions on that page add/edit those notifications
        - This will be handled using the page permissions that CMS provides
        - `CMS_PERMISSIONS` needs to be set `True` in `settings.py` and each student should have permissions on his/her blog page

- **Week 5** (Jun 24)
    - Creating the `build_items` page for creating conditional triggers which will create scheduled items based on a certain condition (ex: if user has not uploaded a blog for more than 7 days)
        - One kind of scheduling will be date and time based. This will build a scheduler item at

the specific date and time
        - Other kind of scheduling will be trigger based. There will be multiple triggers (conditions), which when satisfied will build an scheduler item. An example trigger is as follows:

```
foreach user where group == student if last_blog_post > 7 days then
send_email for user
```

        - An admin page will be created which will let superusers add these triggers and dates so that `build_item` builds these scheduler items when the triggers are satisfied or the date and time is reached
    - Adding a page which will list out all the selected proposals for reference of students of coming years
        - This will be displayed by fetching the `accepted_proposal_pdf` file from the UserProfile and there will be links for each proposal
    - Integrating a hit counter for the articles (can use `django-visits`)
        - Will use the `CounterMiddleware` to get a count of the requested urls
        - Must make sure that only AJAX calls or css links gets counted, otherwise search engines might cause wrong counts. This can be made sure by setting `BOTS_USER_AGENTS` and `IGNORE_USER_AGENTS` lists in `settings.py`

- **Week 6** (Jul 1)
    - Using the Google Calendar API to create a downloadable calendar file specific to users
        - For admins, it will add all scheduled tasks and also the GSoC timeline
        - For other users, it will add only the notifications for that particular user and the GSoC timeline
    - Creating the handler for sending emails from a json template to notify about deadlines to various users
        - The json template will look like this:

```
{
  "to_email": "test@email.com",
  "template": "testemail.template",
  "from_email": "test@python-gsoc.org",
  "Subject": "test email!",
  "data": {
    "username": "test",
    "message": "this is a test message"
  }
}
```

        - The handler will take the necessary data from the json and send email using django's in-built function `django.core.mail.send_email()`

- **Week 7** (Jul 8)
    - Basic SEO for the blogs, mostly using Google Search Console
        - Making sure Google can access the blog contents
        - Making search results more visually engaging
        - Analyzing which queries caused the site to appear in search results
    - Keeping a few days to finish up tasks which might have not been completed

- **Week 8** (Jul 15)
    - The plan for this week is flexible, to be decided according to new requirements that come up

    - Freezing backend work by the end of this week

- **Week 9** (Jul 22)
    - Building a clean, responsive and uniform UI for the whole site using `pure.css`, as the whole site will be built by now
    - Making proper styling classes for each type of element in the website and use them
        - Notifications (different according to priority)
        - Posts
        - Comments
        - Replies
        - Reactions
        - Navbar

- **Week 10** (Jul 29)
    - Writing unit-tests using django's in-built testing platform to test all models, views and forms using the following template

```
class TestClass(TestCase):
    @classmethod
    def setUpTestData(cls):
        print("setUpTestData: Run once to set up non-modified data for all
class methods.")
        pass

    def setUp(self):
        print("setUp: Run once for every test method to setup clean data.")
        pass

    def test_one_plus_one_equals_two(self):
        print("Method: test_one_plus_one_equals_two.")
        self.assertEqual(1 + 1, 2)
```

        - This base class `TestCase` handles db setup and teardown by itself
    - Getting 100% code coverage

- **Week 11** (Aug 5)
    - Writing integration-tests using django's `StaticLiveServerTestCase` base class and `Selenium WebDriver`
    - Extending the Integration tests on multiple browsers
    - Performing `Cross Browser Testing`
    - Freezing the code

- **Week 12** (Aug 12)
    - Refactoring and cleaning up code
    - Fixing bugs
    - Updating documentation for using the application and also a styling guide for creating new landing pages

- **Final Week**
    - Deployment tasks
    - Maintenance

- Submitting project

Apart from this, the plan also includes fixing bugs on a regular basis as and when they are posted by users and mentors.

Expectations from the Project

I have always worked on projects as a part of a huge team, which did not let me have much control over the decision making and having an overview of the system as a whole. With control comes responsibility and I want to learn how to take responsibility of projects and build big systems from scratch. This project will also enable me to work better with a tight timeline and a given set of deliverables.

I have been working with django for quite a bit of time. However, contributing to this project has helped me fine-tune my django skills and know more about the intricacies of the framework. I hope that this project continues serving as means of furthering my skills.

I have always loved python as a programming language because of its utter simplicity, and it will be a great opportunity for contributing to the community. It really excites me that my project will be used by a number of people in the coming years.

Other Commitments

I have summer holidays in my university from May to July. So, I'll be able to devote most of my time to this project. Near the end of May, I'll be going home to meet my parents for a couple of weeks. That won't be a hindrance by any means, but I'll probably be working a little less than usual in these two weeks.