GSoC 2017 Application
Cadasta - Additional Login options

About Me

Name and Contact Info

Name: Pranjal Tale

University: Indian Institute of Technology, Roorkee

Major: Applied Mathematics

Email: pranjaltale16@gmail.com

Github: github.com/pranjaltale16

Personal Website: pranjaltale16.github.io

Time Zone: IST (UTC +5:30)

Resume: Link

Personal Background

- Hello, This is Pranjal Tale, a Sophomore at Indian Institute of Technology Roorkee, India. I am pursuing a degree in Applied Mathematics.
- I got introduced to programming 4 years ago in my junior year at high school and it has sparked my interest since then. I started with C++ and developed small games, meanwhile, learning the fundamentals of HTML/CSS, SQL and XML.
- At college, I joined Information Management Group (IMG), IIT Roorkee - a student group which is responsible for development and maintenance of internet and intranet applications in the institute which are hosted on a central platform called Channel I, including the Website of IIT Roorkee. It was here at IMG that my hobby developed into a passion.
- I now have a strong knowledge of web development frameworks, and the MVC and MTV architectures. I've used Django, Python, Node.js, AngularJS quite extensively. While some of these may not be relevant to Cadasta, it has helped me gain an intuitive and superior understanding of structured web-development. I work on Elementary OS based on Ubuntu 16.04 with Vim as my primary text editor. I also have some experience in UI/UX design and follow proper guidelines in designing the same so that every component is intuitive and well-placed.

Programming Experience

- Information Management Group (IMG)
  At IMG, we strive to lead in the invention, development and maintenance of applications for IIT Roorkee like Placement portal, Notice Board, Study Portal (Lectures and Tutorials) and many more applications. And to glue all these intranet applications together is a platform named Channel I, which is developed in its entirety from scratch by us and is developed on Django itself.
  All these apps with a plethora of features make it very difficult to manage and maintain our code. Thus, we use various tools for it: gitlab to manage code privately, sentry to catch and report exceptions(configured with git to solve them via pull requests) and many more.
- Projects
  Since our work is mainly on the intranet, I can't share their links directly with you. If asked, I can show you a demo via teamviewer.
  - **IMG Website:**
    Tech Stack: Django, MySQL, apache2, Materialize, jQuery
    I am currently working on IMG Website as a full stack developer, hosting its backend on Django, frontend on Materialize and jQuery. It is a fully functional website for IMG which, when launched, will allow all the members to login, maintain their profiles, write and share blogs both privately and publicly along with and admin panel to manage information on the website.
  - **Nobel:**
    Tech Stack: Nodejs, Express, Socket.io, Redis, Semantic-UI
    Nobel is an internal Chat Room for the members of IMG, Built on NodeJs, socket.io and Semantic-UI. This was developed to allow communications in case of an urgent matter. It keeps track of the members present in the lab at any instant of time and allows us to interact with them from outside the lab.
  - **Open Source and Hackathons:**
    Tech Stack: PHP, python, Django, SymPy, Microsoft-Cloud
    I actively contribute to open source organisations and participate in various hackathons. I have implemented complete login mechanisms on both Django and PHP and hosted them on my own apache server. I have regularly been contributing to SymPy(Symbolic manipulation library written in python).
    I have also successfully competed in Microsoft's Code.Fun.Do where we have to make an app from scratch in 24 hours. I made an app on Food Waste Management for which I used Microsoft Azure cloud services and backend was built on Django.

Contributions to open source

My contributions as listed here have helped me gain experience in understanding the flow of any pre-written code at a rapid pace and enabled me to edit/add new code and features with ease:

Cadasta

1. Add user profile images #110 **(In Progress**). Here is the Pull Request.

Sympy

1. Correct the diophantine function in Sympy module.(**Merged**). Here is the Pull Request.

Other Contributions

1. Registered Django-admin in Amplio(a feedback submission and discussion application, **Merged**). Here is the Pull Request.

The Project

The Problem and Motivation

Project Description

It would be awesome to have different and more compatible methods to signup and login to Cadasta-Platform as opposed to the email only login at present. The project I would like to work on as a part of GSoC-2017 is titled Additional Login Options.
I would like to add additional Login Support for Cadasta. Currently users can only login with their email address. Providing alternate methods of login provide them with more flexibility and increases the usability of the application. Primarily I will focus on implementing:

1. Different Methods for user to login and signup.
2. Login using social accounts and phone no.
3. Multifactor authentication for user.
4. Reviewing current api's for accounts app.

Current Structure (in Brief)

Currently in Cadasta authentication system, django-allauth is used as its main component. User model is imported from auth_base.AbstractBaseUser along with all_auth. Views are generic class based views with Mixins and all_auth views inherited.

Ideas And Resources

The complete project can be divided in 5 parts:

- Adding registration and login support with the help of phone numbers along with email address.
- Adding the OTP support for phone numbers, registration verification and multifactor authentication.
- Reviewing of possible APIs for phone support
- Adding a feature of Multi-Factor Authentication for users
- Adding OAuth login support in addition with phone number support

Along with this each feature consist of its tests and corresponding CSS / Frontend changes in templates with them. Following is the detailed overview of each part with a brief description of the resources that I would use/require for the implementation of this project. As for the implementation, it is discussed in detail after the overview.

1. Registration by phone numbers

● Idea

Currently user can only login with their email ids. There is no other option for them to login. It is not necessary for all users to have and use an email and there should be other ways at their disposal using which they can register or login. One such way is their phone number. So, for this feature, we need to add phone number support which would provide the users with an additional option to access their account.
In order to achieve this, an additional field named phone_no will be added to the User model in accounts. Apart from that, email field will now become optional and so will the phone_no field, but it would be necessary for the user to have at least one of them. In case a user has both, he can simply use any one of email id and phone number to login.

● Resources and api

In order to add 'Phone Number' field in User model we will use [django-phonenumber-field] which is a django model and form field for normalised phone numbers using python-phonenumbers.

2. The OTP - package

● Idea

If we are to provide the users with access to their accounts using their phone numbers, it is of utmost importance that we ensure that the contact number entered is valid and more importantly, correct. OTP is an optimal way to achieve that. So, in order to authenticate and validate users' phone numbers, we will need to develop an OTP

module. The use of this package will be to send OTP to the number passed as argument and verify it.
I sifted through the documentations of many django-packages available like django-otp but they are mainly based on django.contrib.auth while the current structure of Cadasta uses django-allauth which might pose problems during integrations and overlapping the views and Mixins.
Also, the main work which would constitute the otp-package will be of two functions one will be to generate otp and the other one to verify and validate it. So, in my opinion, it would be a better idea to write our own code and inherit the current django-otp structure as and when required in both the views. Along with the two functions I will code I will inherit Middlewares and forms also for better exception handling.
The main work of this package is to send otp and verify maximum of the error handling will be done before.

- ### Resources and API

I am going to use Twilio api in order to send messages. There is a blog that explains the complete process in order to integrate twilio with django. Here is the reference. Here is the reference for django-otp package which is based on the django.contrib.auth. Also in order to generate and verify otp we can create forms using django-forms-wizard these are the forms in continuation. Also in order to modify any Class based View Here is the reference that I am going to follow class-based-views and mixins django official tutorials.

3. Review of Current API's for phone support

- ### Idea

Currently there are api for user details, login and register. They are implemented using django_rest serializers. We will update the current serializers and integrate them with phone and OTP support such that user can perform said actions with their Phone numbers as well.

- ### Resources and API

I will follow the django_rest documentation in order to update serializers and according to the generic views of django we can easily update views. We can refer django-serializers from here.

4. Multifactor Authentication

- ### Idea

  After we have completed implementation of phone support for both default views and the api and adding otp support, I will start working on Multi-factor Authentication to provide our users with a more secure way to access their account. The user will have a choice to enable TFA which will be implemented by adding a boolean field named two_fa_enabled to store whether the user has enabled TFA. In order to achieve this, I will integrate the OTP feature along with the current login. After completion, this will establish our Two Factor Authentication(TFA) in which the user will require to enter an OTP along with the correct password in order to login. The OTP sending and verification will be handled by the package that we would have developed in Part-3.

- ### Resources and API's

  Here is the great reference that I am going to follow, where they integrated the django-allauth with Two factor authentication using django_otp package.

5. OAuth Authentication

- ### Idea

  Once we are done with Phone support, OTP support and Multifactor Authentication, we will have an improved user interface which would allow users to login via any one of their email or phone.
  The next step will be to provide the user with even more options to access his/her account. Giving the user to login/signup directly from their Facebook, Twitter or Google+ ids would simplify the user experience to a great extent and would ease the user registration process and would draw more users towards our platform. This can be readily achieved via the django-allauth package which is already being used in the current code. Right now, the User model is built over it.
  During the course of my project, I will be adding OAuth support for basic OAuth providers like Facebook and Google+ and then if time permits, I will add support for Twitter and LinkedIn as well.

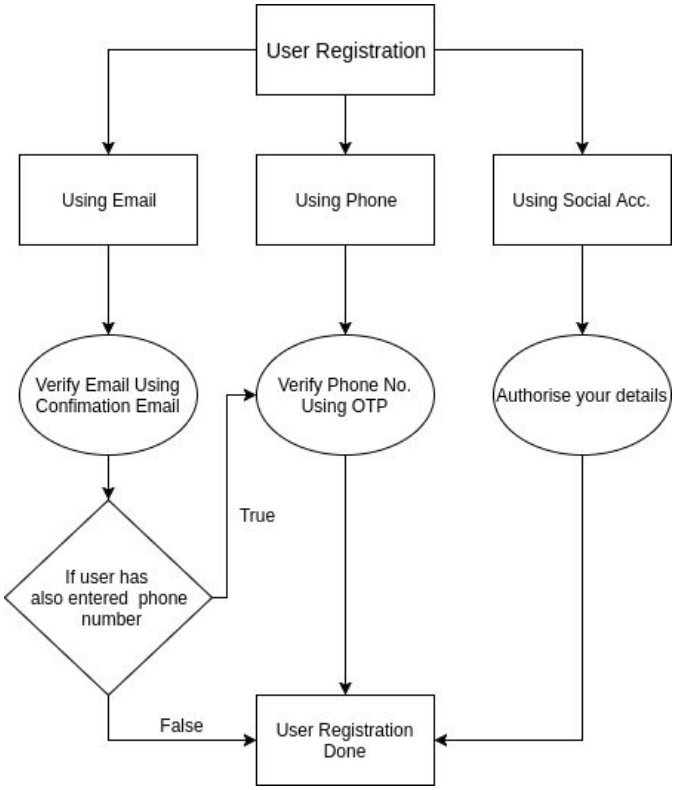
- ### Resources and API

  We can extend the use of the django-allauth package that is already being used for user authentication for this. Initially I plan to add support for Facebook and Google+ and if time permits I will add support for Twitter and LinkedIn as well.
  You can also check out this and this, where the complete process for adding social auth(facebook and google) using django-allauth is given in detail.
  To understand the basic concepts and working of OAuth I took a reference from here. In order to add tests for social accounts here is the great answer that I will follow.

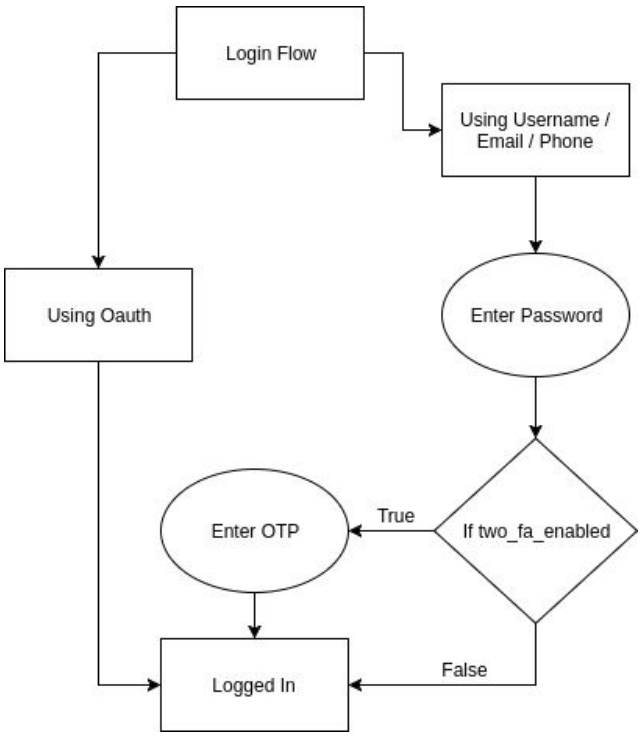
SignUp - Login Flow

Here is a basic flow for login and signup.

SignUp:



Login:



Implementation Details

1. Registration by phone numbers

- ## Approach

In order to implement this feature I will add Phone number field in the user model, which will not be a required field and similarly email field will be modified to be an optional field.However, we will require at least one of email and phone number to register the user, so I will check it before saving the instance. This can be easily achieved by making modifications in forms.py to check whether both the fields are empty or not.

In order to add phone no. field to the *accounts user ya jo bhi kuch naam tha* we can user django-phonenumber-field package as discussed earlier. Here is the sample code snippet of how to add phone field in any Model.

```
from phonenumber_field.modelfields import PhoneNumberField

class MyModel(models.Model):
    name = models.CharField(max_length=255)
    phone_number = PhoneNumberField()
    fax_number = PhoneNumberField(blank=True)
```

Internally, PhoneNumberField is based upon CharField and by default represents the number as a string of an international phone number in the database (e.g '+41524204242').

Correspondingly we will have to modify forms.py. Forms will be modified in two ways first by adding field in it and second by writing a function to check whether both the fields (Phone no and email) are blank or not. This can be achieved by adding custom check in the save function on that formClass.

Also we will have to display phone no. field in both register and Update profile views that can be done by modifying corresponding templates. And for the css part we can extend the current css of forms used easily. Here is the probable form instance of RegisterForm

```python
@parsleyfy
class RegisterForm(forms.ModelForm):
    email = forms.EmailField(required=False)
    password1 = forms.CharField(widget=forms.PasswordInput())
    password2 = forms.CharField(widget=forms.PasswordInput())
    phone = PhoneNumberFormField(required = False)

    class Meta:
        model = User
        fields = ['username', 'email', 'phone', 'password1', 'password2', 'full_name']

    def clean_username(self):
        #Same as the Current Structure
    def clean_password1(self):
        #Same as the Current Structure
    def clean_email(self):
        #Same as the Current Stucture

    #Addition
    def clean_phone(self):
       email = self.data.get('phone')
        if User.objects.filter(phone=phone).exists():
            raise forms.ValidationError(
                _("Another user with this phone already exists"))
        return phone

    def save(self, *args, **kwargs):
        user = super(RegisterForm, self).save(*args, **kwargs)
        #FUNCTION TO CHECK FOR BOTH THE FIELDS ARE BLANK OR NOT
        if one_ofthem_is_non_empty:
            user.set_password(self.cleaned_data['password1'])
            user.save()
            return user
        else:
            raise forms.ValidationError(
                _("Another user with this phone already exists"))
```

After this User can login with the help of their phone numbers along with their emails. But now, it would pose a security threat as user need not to verify their phone numbers, they can directly login and access all the details and so, in order to correctly authenticate the user, OTP module is required which would verify their Numbers which is continued in the next part.

- ## Tests

The main changes and addition of tests will be done in test_models and test_forms. In test_models there will be two tests one for email and one for phone no. currently there is only one corresponding to email. For test_forms we can modify or extend `test_valid_data to (a) test_valid_data_with_email, (b) test_valid_data_with_phone, (c) test_valid_data_without_both (d) test_valid_data_with_both and one more test for test_password_contains_phone`

2. The otp-package

- ## Approach

After giving user access to register with the help of their phone numbers it is necessary to authenticate user and in case of Phone numbers that can be done with the help of OTP-package. The main work associated with otp-package is to generate otp and verify it.
We have to make two functions first is generate_otp and the other one is to verify it. I will generate a model name UserOTP.
The structure of This UserOTP model will be somewhat like this:

```
Class UserOTP(models.Model):
user = models.OneToOneField('User', on_delete = Models.CASCADE)
        otp = models.IntegerField()
valid_till = models.DateTimeField(default = 'timedelta(5)')
no_of_attempts = models.IntegerField(default = 0)
```

There might be modifications in this but this is the basic structure than can be used. Here user is mapped with User models so that there is only one record corresponding to each user then each otp generated is valid for a period of time only. Every time the otp instance is invoked it will check for the valid_till option, if expired it will set no_of_attempts to zero and reset the valid_till field. One form(VerifyOTPForm) will be registered in order to verify generated otp.
Main use of OTP-package is in login view (when two factor auth is enabled), register view and update user profile view. The basic difference here with this otp package is here I actually wrap up the complete opt mechanism into a single unit and it is called after all the checks and authentication, for ex in case of login with Two factor auth enabled there password validation is done with the help of current structure only once all the checks are passed then only this package is called to generate and verify otp.
The Generate OTP view: OTP will be a four character numeric string which can be generated as a combination of 4 randomly generated integers(lying between 0 to 9). After the key is generated a twilio instance will be called in order to send otp and entry in UserOTP model will be updated correspondingly. After this Next View will be called in order to verify the OTP having its form named VerifyOTPForm.
Before Generating the otp in otp view first we will check whether the user is logged in or not or directly that view has been called, this will help us in stopping user to authenticate without logging in and redirect such users to login view.
Verify OTP view: This will be a form instance in order to check the otp this can be attached with the generate otp view with the help of DjangoWizardForms(Showing forms in parts) so that none to them can enter the otp without calling the generate otp function

- ## Tests

  Mainly we have to check for the forms and models, I will create different wrong instances of the model and proceed with it like record with **blank otp**, entered **otp of invalid length** etc and one with the all valid values. Then submitting **blank forms** and similar tests for different form instances.

3. Review of Current API's for phone support

- ## Approach

  Currently for accounts we have 3 urls Profile, login and register. We will modify mainly two serializers namely UserSerializer and RegistrationSerializer by adding phone no support in them such that user can see their details along with phone no when they visit '/api/vi/account' and can register as user with their phone numbers when they visit '/api/v1/account/register'. API's are based on django_rest framework serializers.

  In Registrationserializer I will add validate_blank_field function to check whether both the fields(email and phone) are blank or not also the same can be checked before saving or updating instance. For reference you can check here .

- ## Tests

  Mainly tests will be added in test_serializers.py, a few of them are `test_create_with_phone_without_email`, `test_create_with_email_without_phone`, `test_create_with_both`, `test_create_with_existing_phone`, `test_password_contain_phone`. In order to write tests I will refer this. For example test for registering user with existing phone can be written as:

```
def test_create_with_existing_phone(self):
    """Serialiser should be invalid when another user with the same phone
       no is already registered."""

    UserFactory.create(phone='+919876543456')

    data = {
        'username': 'imagine71',
        'phone': '+919876543456',
        'password': 'iloveyoko79!',
        'password_repeat': 'iloveyoko79!',
        'full_name': 'John Lennon',
    }

    serializer = serializers.RegistrationSerializer(data=data)
    assert not serializer.is_valid()
    assert (_("Another user is already registered with this Phone Number")
            in serializer._errors['phone'])
```

4. Multifactor Authentication

- ### Approach

In order to add multi factor authentication I will start it by adding a BooleanField named two_fa_enable in User model. Multifactor auth is mainly going to be attached with the login view. Hence LoginForm and views associated with login are going to be affected. In login view I will check the value of two_fa_enable, if it is true then the request will be forwarded to generate otp and verify otp view and then it will be redirected accordingly. Also I will add a MiddlewareMixin of the form shown below

```
class AllauthTwoFactorMiddleware(MiddlewareMixin):
    """
    Reset the login flow if another page is loaded halfway through the login.
    (I.e. if the user has logged in with a username/password, but not yet
    entered their two-factor credentials.) This makes sure a user does not stay
    half logged in by mistake.

    """
```

Also The view for login, i.e., Account_login view will be updated where we will check the value for two_fa_enabled and will proceed in accordingly: if it is true it will be redirected to the OTP-module part otherwise the current process will be executed. User can enable or disable TFA from update profile view, there will be a separate form for this the main purpose of this form is just to update the UserOTP -> ‘two_fa_enabled’ field.
If user has not entered the phone number then they can’t see the Enable Two factor auth option, it is visible only when the phone number is verified.

- ### Tests

For two factor authentication here is the link which I will follow to write tests.
Some of them are

- Test login behavior when 2FA is not configured.
- Test login behavior when 2FA is configured.
- Test login behavior when 2FA is configured and wrong code is given.
- Going to OTP page when not logged in.

5. OAuth Authentication

- ### Approach

I will add support for user to register directly using their social accounts like facebook, google this can be achieved easily with the help of django-allauth package as it is already used in the current code for authentication and is already integrated. We will start by

registering in our OAuth service providers in settings.py and making migrations for the same. Then we have to create our key, which is generated from the api console or on the api platforms. Now we have to register the Social Application in our code that can be done with the help of admin.
After that the templates can be modified with reference to django-allauth official documentation [django-allauth] Here I will use social account tags defined in it to update templates. I will use `provider_login_url` tag to generate provider specific login URLs.

- ## Tests

  In order to check for the social accounts here is the sample test with correct credentials now I will alter details in this test to validate related tests associated with it.

```
class SocialAccountTests(TestCase):

    @override_settings(
        SOCIALACCOUNT_AUTO_SIGNUP=True,
        ACCOUNT_SIGNUP_FORM_CLASS=None,
        ACCOUNT_EMAIL_VERIFICATION=account_settings.EmailVerificationMethod.NONE  # noqa
    )
    def test_email_address_created(self):
        factory = RequestFactory()
        request = factory.get('/accounts/login/callback/')
        request.user = AnonymousUser()
        SessionMiddleware().process_request(request)
        MessageMiddleware().process_request(request)

        User = get_user_model()
        user = User()
        setattr(user, account_settings.USER_MODEL_USERNAME_FIELD, 'test')
        setattr(user, account_settings.USER_MODEL_EMAIL_FIELD, 'test@test.com')

        account = SocialAccount(user=user, provider='openid', uid='123')
        sociallogin = SocialLogin(account)
        complete_social_login(request, sociallogin)

        user = User.objects.get(
            **{account_settings.USER_MODEL_USERNAME_FIELD: 'test'}
        )
        self.assertTrue(
            SocialAccount.objects.filter(user=user, uid=account.uid).exists()
        )
        self.assertTrue(
            EmailAddress.objects.filter(user=user,
                                        email=user_email(user)).exists()
        )
```

Proposed Timeline

Community Bonding Period (5 May - 30 May).

My vacations will start from 5th May. I will utilise this time to get more familiarized with the current code structure, read the documentation and get to know my mentor(s) and other fellow community members so that we can work together with an amplified efficiency. Some of the important things to study up on will be to learn more about the Core django-allauth, django-mixins, django-adapters and middlewares apart from planning and strategizing about the changes that would need to be done in models with the mentor. There are also other things to look upon and discuss with the mentor during this period (specifically for Multi-factor authentication).

Week 1 (May 30th - June 5th)

By this time, during the community period, I would have finalized and planned all the changes that need to be made to the model in order to add phone support and hence, I will add the phone_number field to the model as discussed above. Apart from that, I will also implement the required changes in UserRegisterForm and the templates corresponding to Login and signup. By June 5th, we would have compete phone support working.

Week 2, 3 (June 6th - June 19th)

Once I have completed phone support, I will turn my attention towards implementing the OTP module. I believe it would take me a maximum of one week to set up the messaging client of choice which I would discuss with the mentor during community bonding period (most probably it would be Twilio). During the next week I would implement the two views: one to generate the OTP and another to verify and validate the OTP. This OTP module will be completed by June 19th and would then be used to fulfill multiple purposes.

Week 4 (June 20th - June 26th)

Once the OTP module is completed, I will integrate it with the already implemented phone support such that whenever the user tries to register or change phone number, an OTP will be generated and the number will have to be verified by the user. This shouldn't take too long to implement and for the time that I am left with in this week, I'll thoroughly test all the features implemented till now and fix any bugs that arise. By June 26th, we will be completely ready with our phone support along with OTP wherever it is required and we will submit it for our Phase-1 evaluation.

Phase One Evaluations

Week 5 (June 27th - July 3rd)

Now that I have implemented phone support and OTP, I will catch up on the tests for these features and complete the tests for phone support and OTP as I discussed in the implementation section above. This will be followed by yet another rigorous testing and bug-fixing session and by July 3rd, my first PR will be ready to be merged.

First Pull Request

Week 6 (July 4th - July 10th)

Once the above code is merged and the default views are integrated, I will start working on the api endpoints that are User, Login and Register. By July 10, OTP and phone support will be added to the api as well.

Week 7 (July 11th - July 17th)

I would dedicate this week for writing the tests for api integration of phone and OTP support followed by another rigorous and meticulous testing and debugging session so that my code is ready to be merged.

Week 8 (July 18-July 24th)

I would keep this week as a miscellaneous/filler week to account for any roadblocks/bottlenecks that may be encountered during the development of above code, or to catch up with writing code in case I am lagging. I am keeping this week to account for any unforeseen circumstances that we may encounter which take longer than we expected so that we have our phone support ready for both default views and all the api endpoints for Phase-2 Evaluations by July 24. In case everything goes as planned and everything is completed on time, I will use this week to plan for further features that need to be implemented for the completion of this project and start working on them.

Second Pull Request

Phase Two Evaluations

Week 9 (July 25th - July 31st)

In week 9, I will implement Multi-factor authentication. This will be implemented as discussed in the implementation section by adding a BooleanField named two_fa_enable to store if the user has enabled two factor authentication. I will complete its implementation including its integration with forms and templates by July 31.

Week 10 (August 1st - August 7th)

In week 10, I would implement OAuth using django-allauth which would be easily done since it is already being used which would save me the trouble of integrating it. I would just have to follow the references I stated above to implement it and it would be easily done by Aug 7.

Week 11 (August 8th - August 14st)

In this week I would write the tests for multi-factor authentication and oauth and back them up with yet another thorough testing and bug fixing session. This would mark the completion of our project including all the main important features that we sought to implement in the beginning. I will complete the tests by August 14, after which the code will be ready to be merged and ready for final evaluation well in time.

Week 12 (August 15th - August 21st)

This would be the final week. I will keep this week to finish things up and clean the code wherever and whenever required and sort things that were left out (if any) during the making of this project. In case everything goes according to plan and I am already done with the project, I will try to implement some features from the wishlist of features which are comparatively low on importance but would improve user experience and accessibility of our platform.

Third and Final Pull Request

Final Evaluation

Notes

I was busy reading the references mentioned above in this proposal for the past 8-10 days, and the PR I had been working on was also a pretty big one. So, I wasn't able to devote time to the PR and hence not a lot of work is done on it in recent times. Now that I've completed the proposal, I'll be concentrating again on the PR. Currently, it is almost complete, just some tests need to added further and I assure you that I'll complete this PR as soon as possible.

How do I fit in?

Although the PR that I have been working on was a big one, it was related to the User code which is a key element to this proposal as well and hence, I have gotten quite familiar with the user code and I think that's what would make me an ideal candidate for this project in GSoC 2017.

Apart from that, I have no other engagements for the whole of summer and can easily attribute in excess of 50 hours a week for the complete duration of this project.