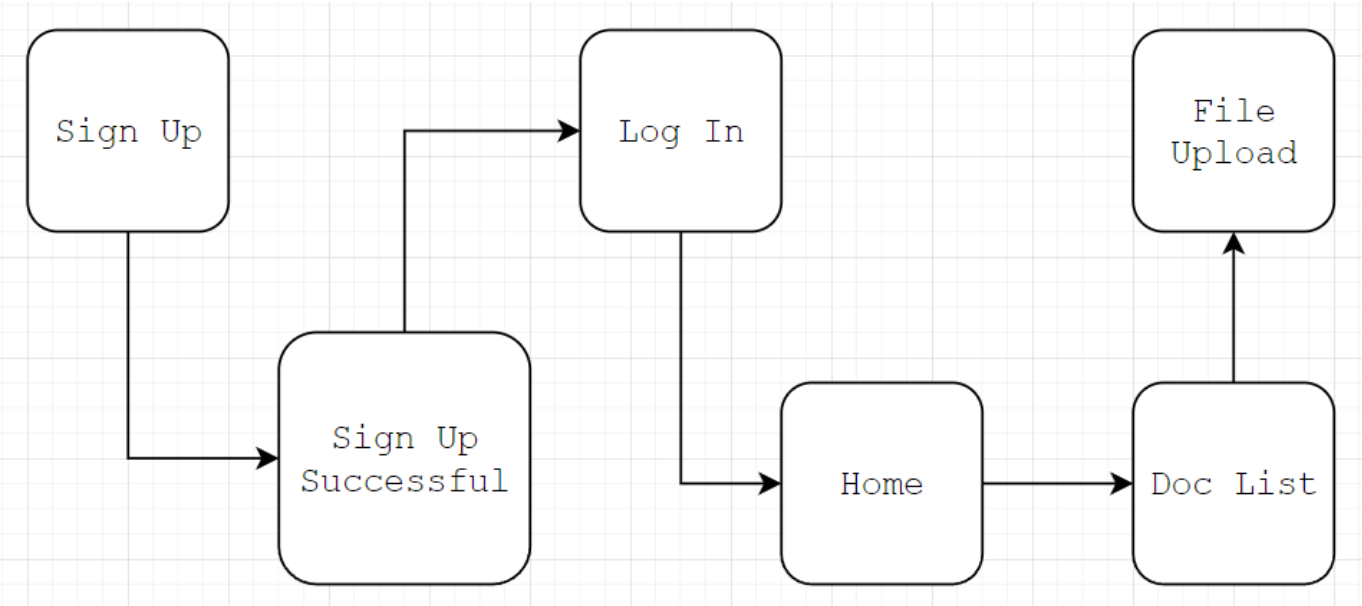
# Cloud Vault

We'll be creating a simple file storage system with **Django** with following features -

- **Metadata** or note could be added with files
- Files are **user specific** i.e. no user can see the files of another user
- Sign up and log in with **phone number** rather than email
- Phone number **verification** on sign up via OTP

## Screen Flow




## Why Django?

Django has excellent database support out of the box and are very easy to work with forms and models. As most of our features have form like functionalities (Sign up, Log in, File upload etc.), we though why not the mighty Django itself?

## Custom User Model

The reason we'll be using a custom user model rather than the one provided by Django is that the built in onw requires username where we are interested in phone no. based user authentication. So we make these changes in `models.py` -

```
from django.db import models
from django.core.validators import RegexValidator
from django.contrib.auth.models import AbstractBaseUser
from django.utils.translation import gettext_lazy as _

class User(AbstractBaseUser):
    """
    Custom user model.
    """
    phone = models.CharField(
        _('phone number'),
        validators=[RegexValidator(regex=r'^0[0-9]{10}$')],
        max_length=11,
        help_text='e.g. 01712345678',
        unique=True
    )
    email = models.EmailField(blank=True)
    is_active = models.BooleanField(_('active'), default=True)
    created_at = models.DateTimeField(auto_now_add=True)

    objects = UserManager()

    USERNAME_FIELD = 'phone'
    EMAIL_FIELD = 'email'
    REQUIRED_FIELDS = []
```

As we are planning to use the `phone` numbers as our primary key so it needs be unique here. Also, `email` is not mandatory in our application of course. Another thing to notice that we are using a field `is_active` to track if the user created is activated or not. They will be activated once they are verified. Last but not the least, `created_at` and `modified_at` are two fields we should have in almost every models. They help us debug our system by such means we cannot even imagine. In this particular case, `modified_at` is not necessary as we are not allowing change in users after creation.

One thing to remember is that Django has it's own ORM out of the box. So when we make some change in `models.py`, we just need to execute `./manage.py migrate`.

An ORM gives us the flexibility of working only with objects and JSONs rather than writing repeatative, long and boring SQLs.

Here we are using **regex** to validate phone number and **gettext_lazy** method to add a string name to the model field. The fun part is this concize piece of code is capable of creating the whole form, with the help of `forms.py`, run the validators and show appropriate messages upon those validations. It will also pack the whole model in a single dictionary for us to send to the database. How easy our lives are with Django, right?

One last nail in the coffin, to change `settings.py` and adding `AUTH_USER_MODEL = 'code.User'`

## Sign Up Form

To create the form with our specific fields, we need make these changes in the `forms.py` file -

```
class SignupForm(UserCreationForm):
    """
    Create a user with phone and password.
    """
    class Meta:
        model = User
        fields = ('phone', 'email')

    def save(self, commit=True):
        if commit:
            code = gen_verify_code()
            with transaction.atomic():
                user = super().save(True)
                verification = Verification(user=user, code=code)
                verification.save()
            logger.info('set verification for %s with code: %s', user.phone, code)

            # send the verification code through SMS
            start = time.time()
            send_task.delay(user.phone, f'Cloud Vault Verification Code: {code}')
            end = time.time()

            logger.info('sent SMS in %dms', (end - start) * 1000)

            return user

        return super().save(False)
```

## Sign Up View

Let's make some changes in the `views.py` to get our custom form to show -

```
class SignupView(CreateView):
    form_class = SignupForm
    template_name = 'core/signup.html'
    success_url = reverse_lazy('core:verify')

    def form_valid(self, form):
        user = form.save(commit=False)
        logger.info('submitted signup form with phone: %s', user.phone)
        user.is_active = False
        return super().form_valid(form)

    def get_success_url(self):
        return reverse('core:verify', args=[self.object.phone])
```

When our Sign Up URL gets a **GET** request, it is going to render the template and when it gets a **POST** request it will validate the form and then save it to the database.

The reason we differentiate these two is because most of the time we have the same URL for our GET and POST requests. When we hit reload after submitting the form once, it tries to send the POST request again so the browser often warns us of this. That is why we should redirect our user to some other page after submitting the form. Here we'll redirect them to the Sign Up Successful screen.

```
def signup_success(request):
    context = {
        'login_url': reverse('core:login')
    }

    return render(request, 'core/signup_success.html', context)
```

In our `signup.html`, we should add some not so html lines to show our form. Django itself handles these lines and render plain html, css to the client.

```
{% extends 'core/base.html' %}
{% block content %}
    <h1>Signup to Cloud Vault</h1>

    <form method="post">
        {% csrf_token %}

        {{ form.as_div }}

        <input type="submit" value="Submit">
    </form>
{% endblock content %}
```

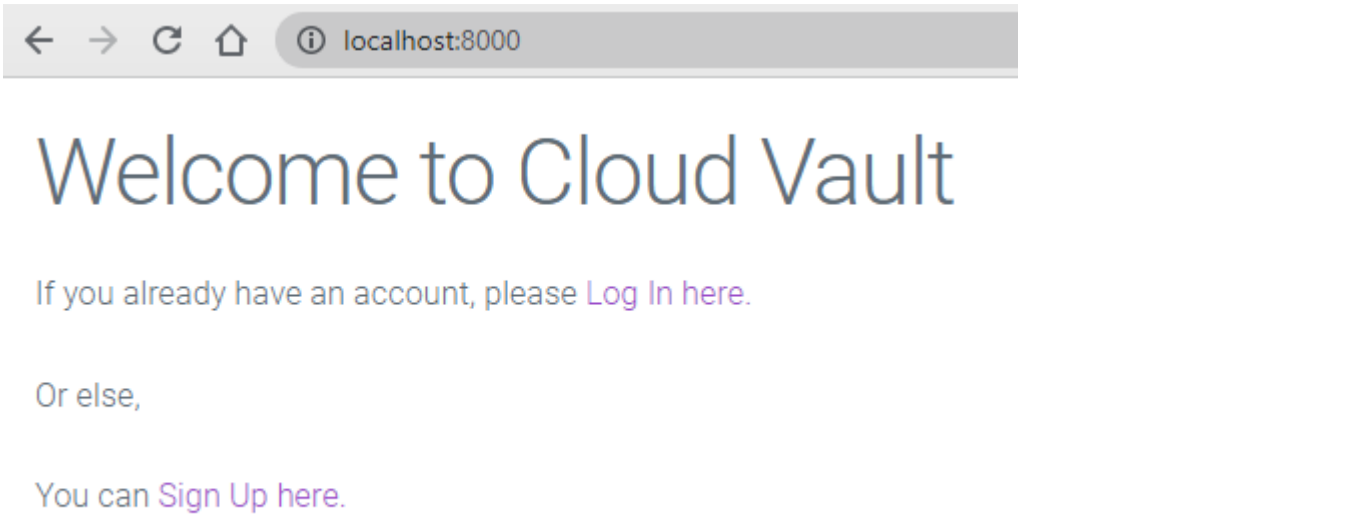
Here `csrf` means Cross Site Reference Forgery and this token is used to resist any other tab open in our browser to submit the form other than us. To understand more about csrf tokens, go to this link.

The `form.as_div` line here basically shows us the form we created. As simple as that.

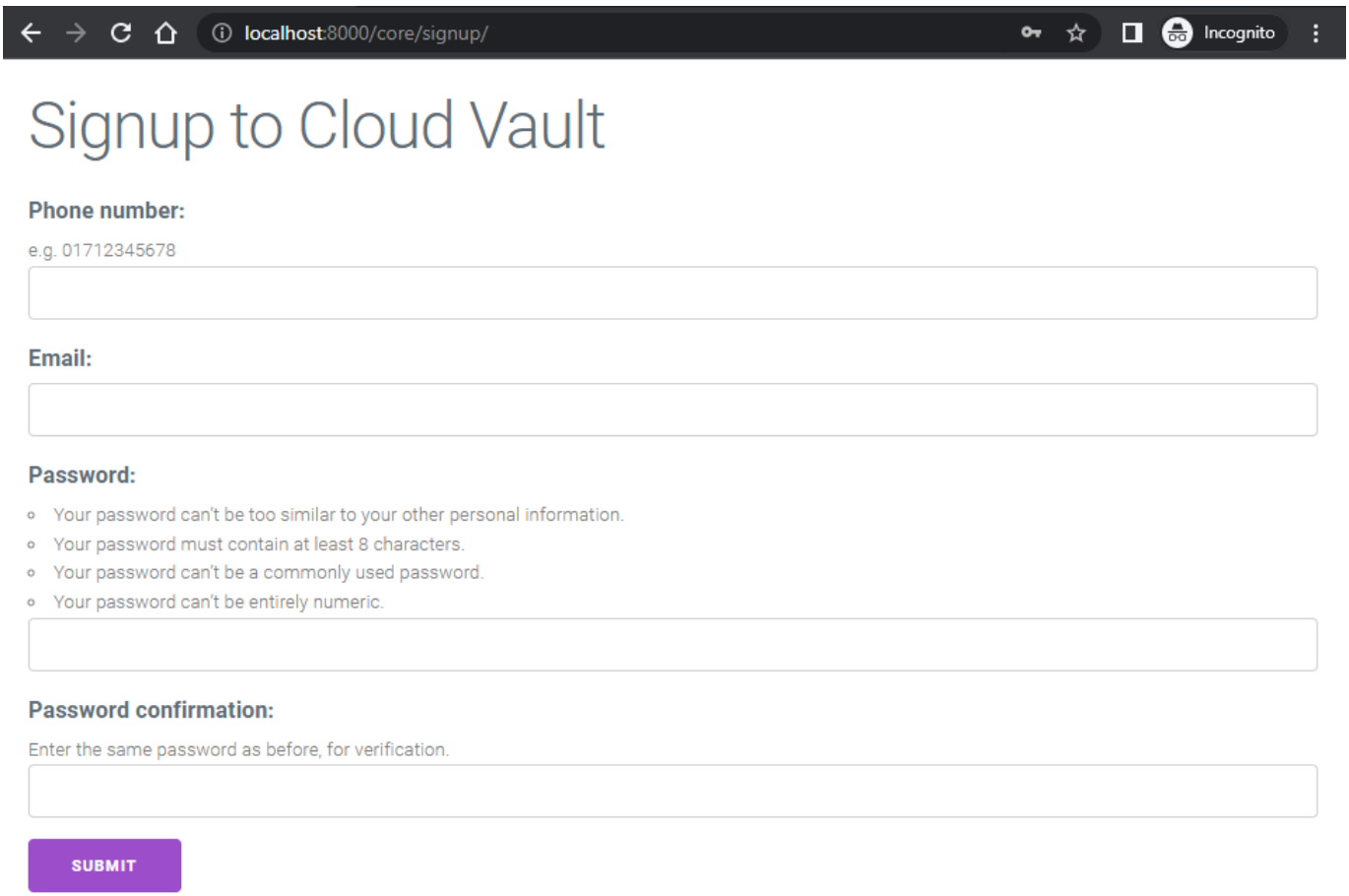
## How does it look?

Let's get a demo of what we did so far.

> python manage.py runserver

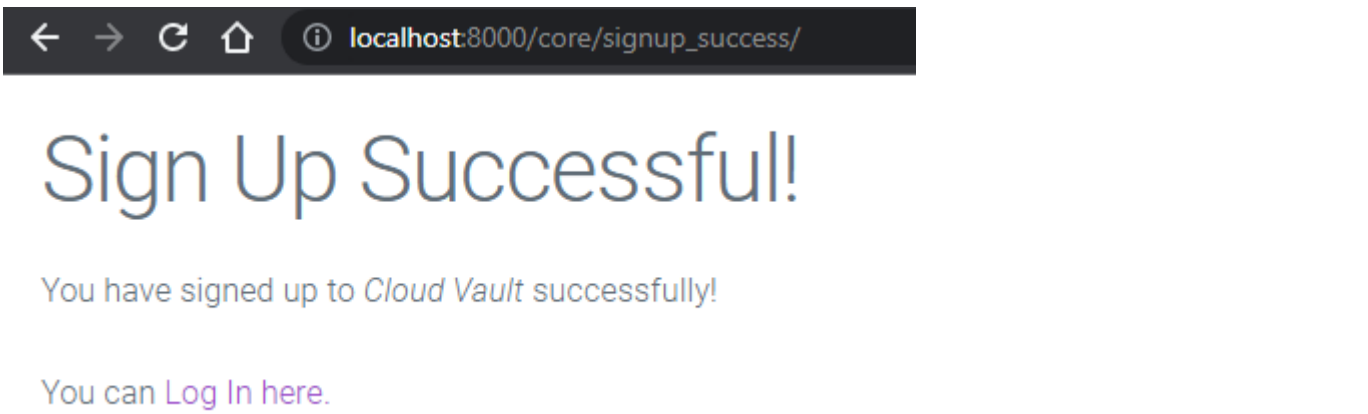



Upon running the server, we get served with this landing page. From here, let's signup first.

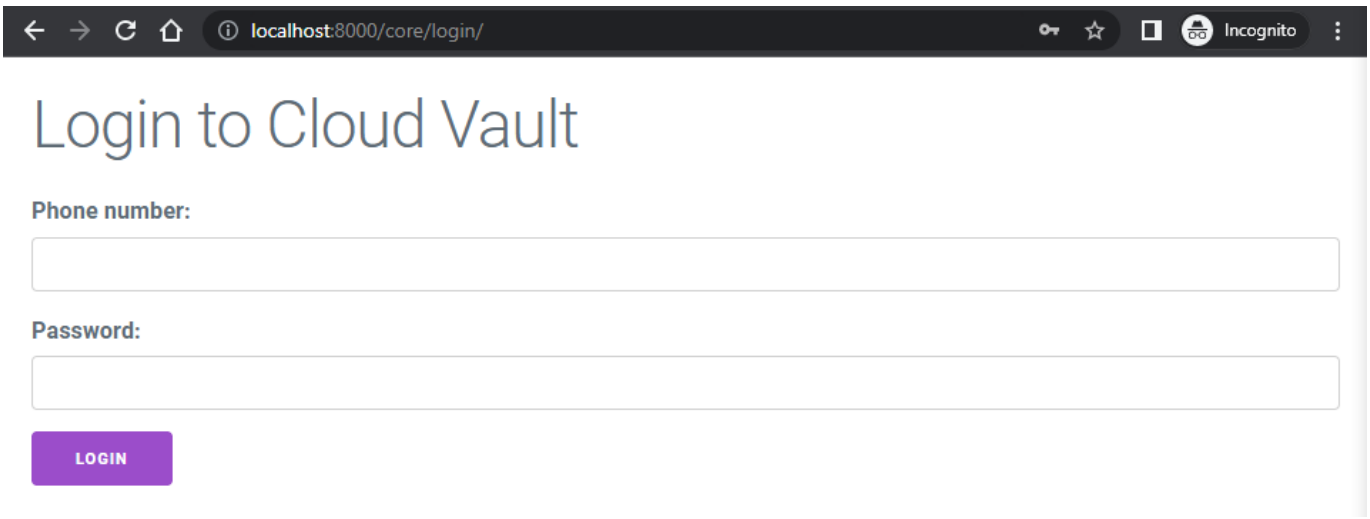



After we submitted our form, we were taken to the verification page. Here we have to add the verification number we got via SMS.

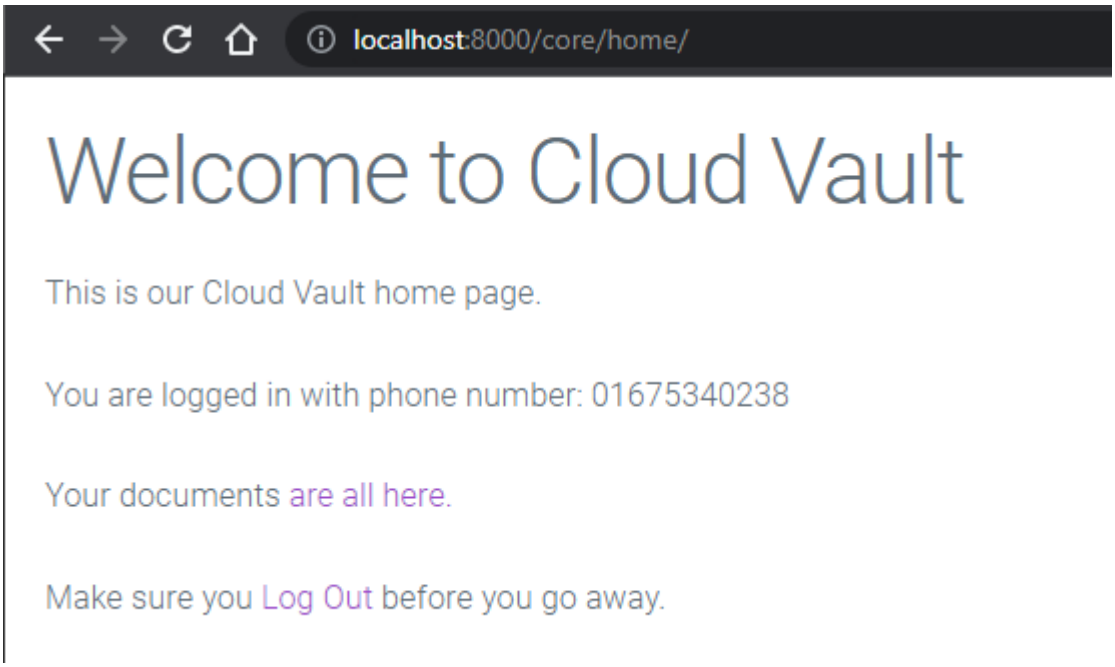
Now this is our signup successful page -




Now we can log in using our previously entered phone no. and password.




After we are logged in, we can see our home page.




## Tools

We have used **Redis** for sending OTP through SMS.

For our basic CSS purpose, we used **Milligram**.