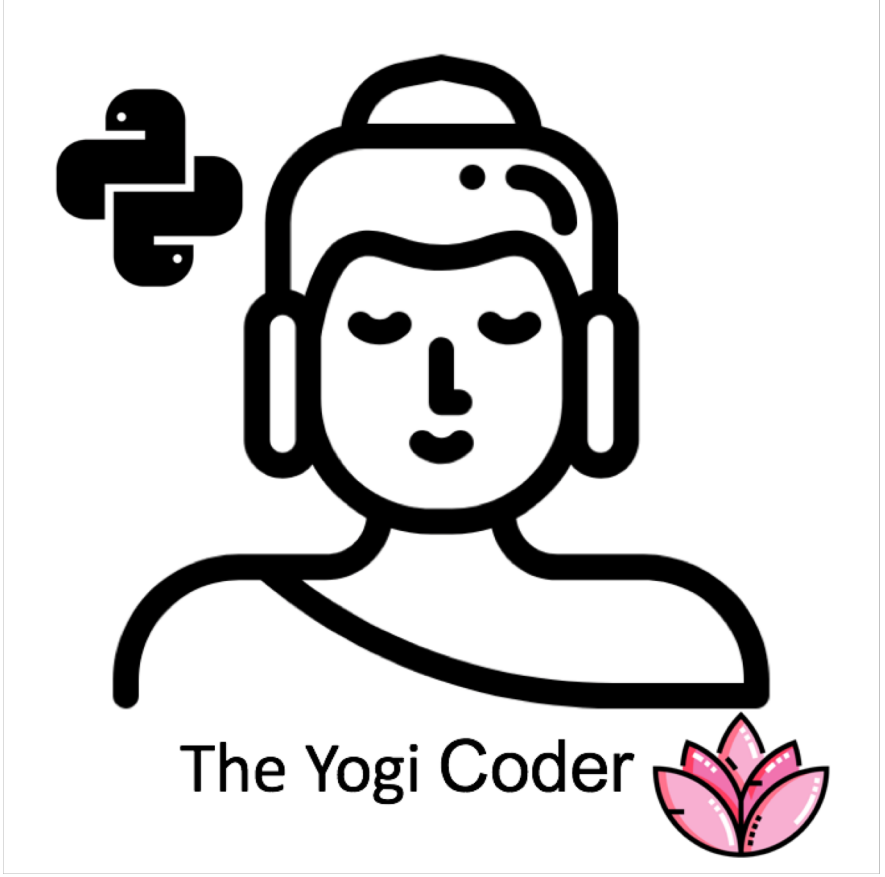
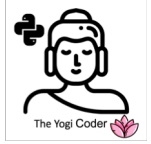


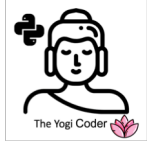
Using allauth & Twitter Login With Django

Inch by Inch Series 2019



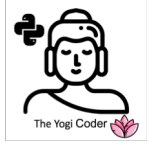
What are the objectives of this class?

- To setup the authentication of our Django Project using a popular package called **django-allauth**
- To understand basic authentication concepts
- To know how to create a **custom user model** through the Django framework
- Configure a **postgres** Database backend
- And configure a social authentication third-party service for our Development Environment, in our case it will be **Twitter**



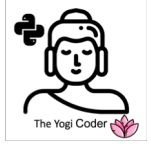
What is authentication?

- Authentication is the activity of determining whether someone or something is, indeed, what it claims to be.
- Generally, the way an authentication system works is it checks to see that the user credentials supplied at login (*username and password for example*) match what is stored for those values in the applications records (for example, a database) for the user trying to log in to your application.
- The password is known as an authentication factor (this is something that only the user logging in ***should*** know).
- There are other types of authentication like two-factor or Multifactor authentication (MFA) in which case the user may provide "something you have" like a special RSA key in addition to their password or even or fingerprint or iris (eye) as "something you are"



How does django perform authentication?

- Django comes with a user authentication system which allows for administration and management of user and group permissions and cookie-based sessions.
- This system addresses both authentication and authorization
  - Authorization permits an authenticated user to perform certain tasks and access certain data within your application
- Through Django authentication, parameters are automatically setup when you create your Django project using the "django-admin startproject" command
- The *user* object is the heart of the Django authentication system. It's through this object that access is managed to your site. This user object has *attributes* such as: "username", "password", "email", "is_active", "last_login" and *methods* such as: get_username(), check_password(), has_perm()
- Anytime you make permission changes in Django the python3 manage.py migrate command must be run to propagate those changes to the database.
- The basic syntax to see if a user has permissions to the data of a given application would be:
  - user.has_perm("my_app.view_my_model")
- Users gets logged in through the login() function and logged out through the logout() function
- Django supplies mechanisms to require a user be logged in such as the @login_required decorator.
- For passwords management , by default, Django uses the PBKDF2 (cryptographic key derivation functions) algorithm with a SHA256 hash, a password stretching mechanism recommended by NIST. This should be sufficient for most users: it's quite secure, requiring massive amounts of computing time to break. You can also use *bcyrpt* and *Argon2* with Django by installing their libraries using pip install
  - You may also use Django password validators to ensure sufficiently strong passwords are used (these settings are made in the settings.py file in the AUTH_PASSWORD_VALIDATORS section)



Customizing Authentication in Django

- The native Django authentication system may be customized through either extending the existing User model or substituting a totally new model
- We will be substituting a new model today

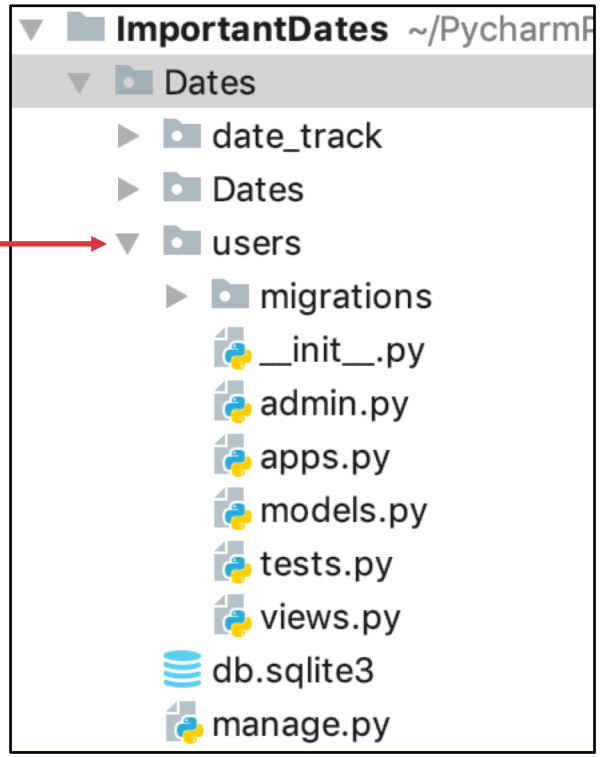
Creating a Custom User Model - 1

We create the custom user model to be able to add fields to the native Django **user** model.

1. Create a new application in our project called "**users**" (from within our virtual environment)

```
$ python3 manage.py startapp users
$
```

Creates new application called "users"



Creating a Custom User Model - 2

2. Modify the settings.py file:

2a - add the new users application to the INSTALLAED_APPS:

2b – and a new setting called AUTH_USER_MODEL to point to our new ***users*** application and a db model we will create called CustomUser

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'date_track.apps.DateTrackConfig',
    'users',
]

AUTH_USER_MODEL = 'users.CustomUser'
```

Django allows us to override the default user model by providing a value for the AUTH_USER_MODEL setting that references a custom model

Creating a Custom User Model - 3

3. Create our new model in users/models.py

```
from django.contrib.auth.models import AbstractUser
from django.contrib.auth.models import UserManager


class CustomUserManager(UserManager):
    pass

class CustomUser(AbstractUser):
    objects = CustomUserManager()
```

Customer Manager-

A Manager is the interface through which database query operations are provided to Django models. At least one Manager exists for every model in a Django application.

- ✓ If we want to include fields that are beyond what are in the default user, then we need to write a custom manager

Creating a Custom User Model - 4

4. **Update Built-in forms-** we will update the UserCreationForm and UserChangeForm so it points to our new model (CustomUser)

The UserCreationForm and the UserChangeForm are tied to the User object from Django's model class and need to be re-written or extended to work with a custom user model

The UserCreationForm is a ModelForm (that is a helper class form from your django model) for creating a new user- natively it comes with 3 fields:

- **Username**
- **Password1**
- **Password2**
  - Using the validate password() method it makes sure the 2 passwords match

The UserChangeForm is a form used in the admin interface to change a user's information and permissions.

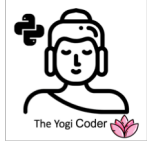
Add this code in users/forms.py

```
from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth.forms import UserChangeForm
from . models import CustomUser # imports our custom model

class CustomUserCreationForm(UserCreationForm):

    class Meta(UserCreationForm):
        model = CustomUser
        fields = ('username', 'email')

class CustomUserChangeForm(UserChangeForm):

    class Meta:
        model = CustomUser
        fields = UserChangeForm.Meta.fields
```

Creating a Custom User Model - 5

5. Register the new model in the user applications admin.py file

```
from django.contrib import admin
from django.contrib.auth.admin import UserAdmin
from . forms import CustomUserCreationForm, CustomUserChangeForm
from . models import CustomUser

class CustomUserAdmin(UserAdmin):
    model = CustomUser
    add_form = CustomUserCreationForm
    form = CustomUserChangeForm

admin.site.register(CustomUser, CustomUserAdmin)
```

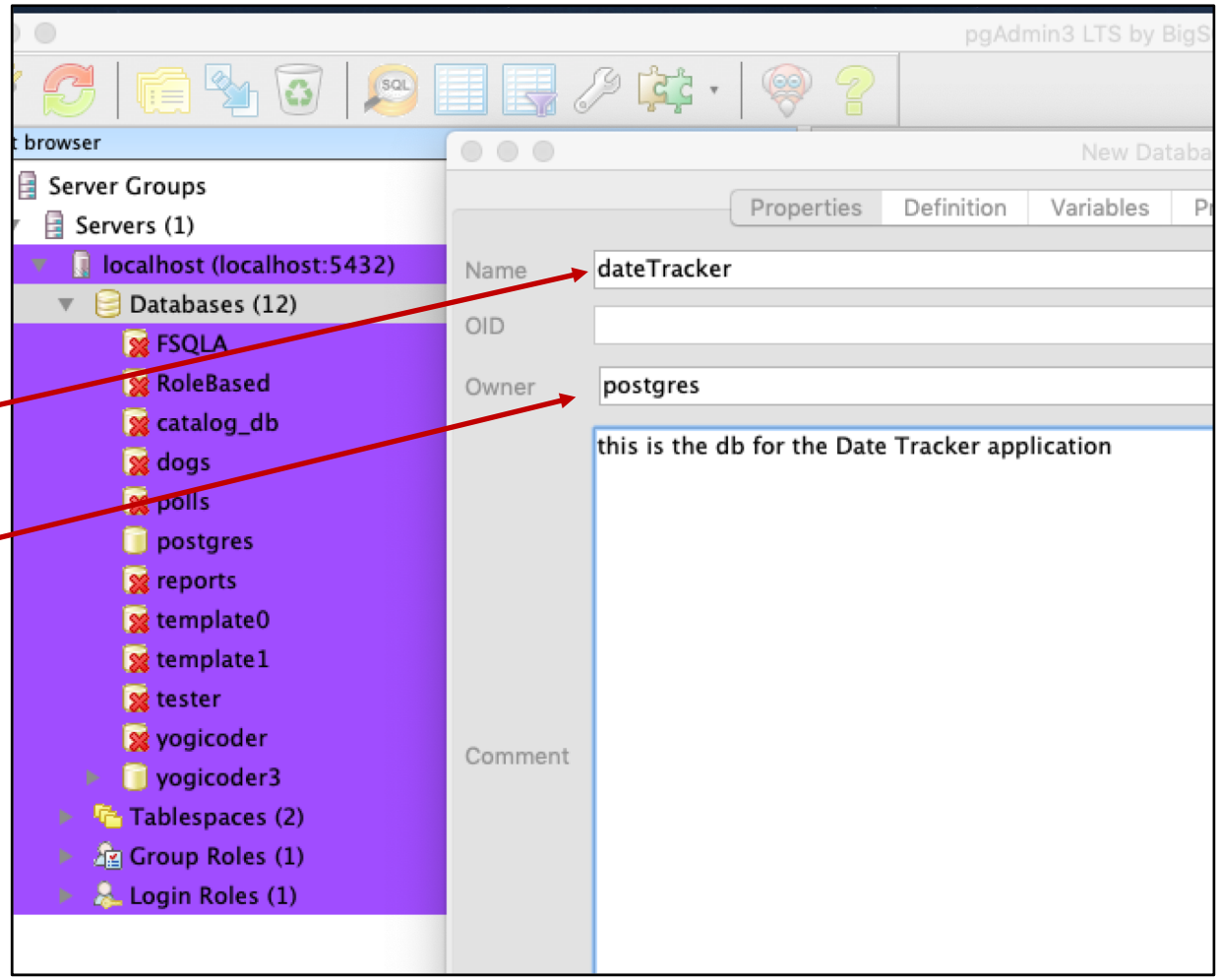


Let's Change Our DB Backend to Postgres

Install Postgres & Create a New DB using the Postgres pgAdmin3 tool

1. Follow directions to download Postgres and its Gui (pgAdmin3 tool) for your operating system:

2. Once Postgres is up and running, you can right click on the "Databases" icon and create a new db. Mine is called "dateTracker", with owner selected as "postgres"



Django models will create the schema(tables), but the DB must be created first

Make the modification to the settings.py file for your project

In Dates/Dates/settings.py, modify your DATATBASES settings to look like this, but use your own specifics. Username will be postgres however.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'dateTracker',
        'USER': 'postgres',
        'PASSWORD': 't361dhebfr9gh23rh4g29r2f',
        'HOST': '127.0.0.1',
        'PORT': '5432',
    }
}
```

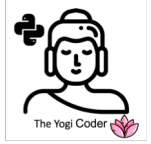
Fake password shown here

Create new db and create your superuser

From within the virtual environment in the project directory run the makemigrations and the migrate users command as you see them below to create our new database and then, create your superuser

```
(venv) Claudias-iMac:Dates claudiaacerra$ python3 manage.py makemigrations users
Migrations for 'users':
  users/migrations/0001_initial.py
    - Create model CustomUser
(venv) Claudias-iMac:Dates claudiaacerra$ python3 manage.py migrate users
Operations to perform:
  Apply all migrations: users
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0001_initial... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying users.0001_initial... OK
(venv) Claudias-iMac:Dates claudiaacerra$ python3 manage.py createsuperuser
```

This will ask you to input name, email and password twice and will check for some basic password validation



Create a t users/urls.py file

To ensure the admin console can be reached , I had to create the admin URL pattern in the Dates/urls.py file

```
from django.contrib import admin
from django.urls import path, include


urlpatterns = [
    path('admin/', admin.site.urls),
]
```

Fire up our application!

1.

```
$ python3 manage.py runserver
```

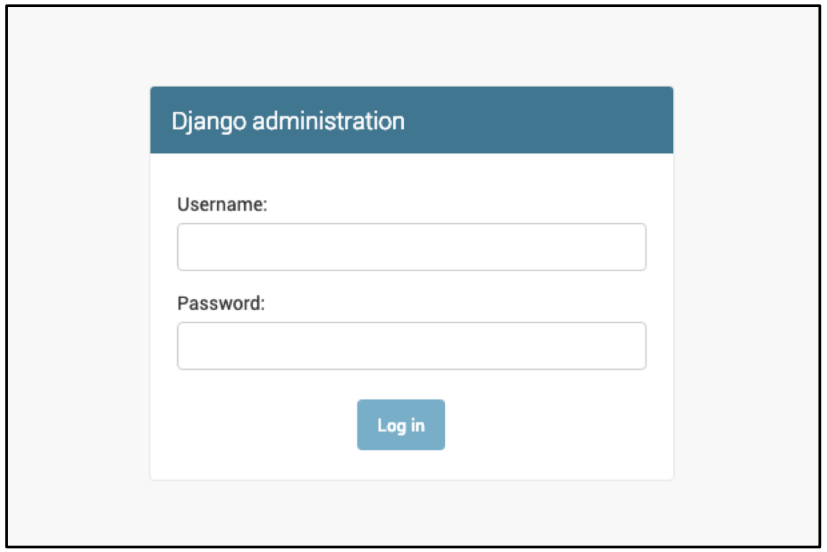
Start the server!

2.

```
Django version 2.1.5, using settings 'Dates.settings'
Starting development server at http://127.0.0.1:8000/
```

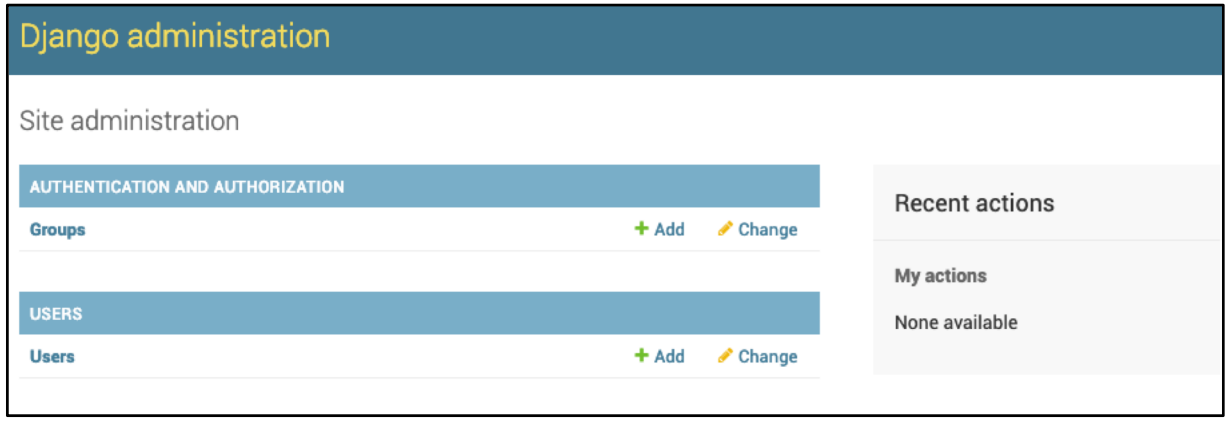
Access the server URL!

3.



Visit the Django admin console at :
http://127.0.0.1:8000

4.



Pass in your superuser credentials and you will be able to see the this above

Update URLs

Dates/urls.py (our project URLConf)

date_track/urls.py | users/urls.py | Dates/urls.py

```python
from django.contrib import admin
from django.urls import path, include

urlpatterns = [

    path('', include('date_track.urls')),
    path('admin/', admin.site.urls),
]
```

date_track/urls.py (our main application URL)

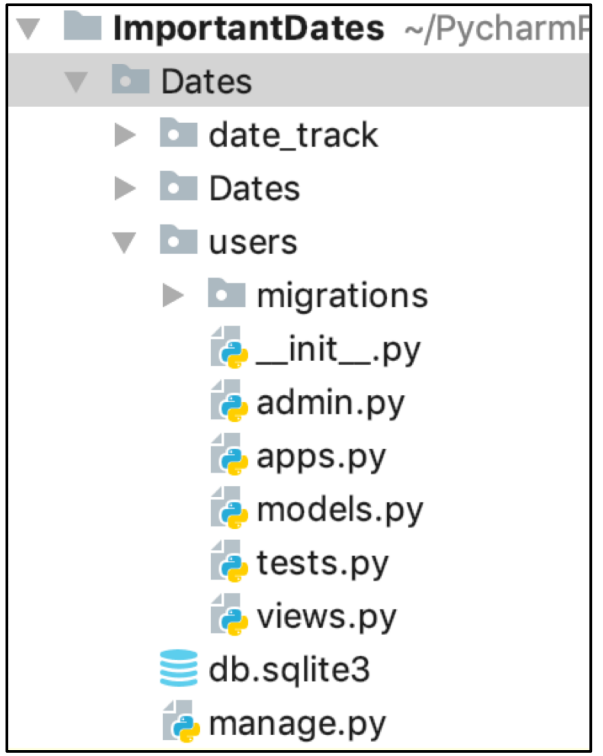
date_track/urls.py | users/urls.py | Dates/urls.py | login.htm

```python
from django.urls import path, include
from . import views

urlpatterns = [
    path("", views.home, name="home"),
    path('users/', include('users.urls')),
    path('users/', include('django.contrib.auth.urls')),

]
```

users/urls.py gets created new

```python
from django.urls import path
from . import views
from django.contrib import admin


urlpatterns = [
    path('signup/', views.SignUp.as_view(), name='signup'),
    path('admin/', admin.site.urls), # so we can get here from home page
]
```

Reminder of what our directory structure looks like →



Routing People to the Home Page- Modifying *settings.py*

You remember that our home page's view is called "home"

date_track/urls.py (our main application URLConf)

date_track/urls.py | users/urls.py | Dates/urls.py | login.htm

```
1  from django.urls import path, include
2  from . import views
3
4  urlpatterns = [
5      path("", views.home, name="home"),
6      path('users/', include('users.urls')),
7      path('users/', include('django.contrib.auth.urls')),
8
9  ]
```

date_track/views.py (our main application's view)

```
from django.shortcuts import render
from datetime import datetime


def home(request):
    current_user = "Mable Marbles"
    return render(request, 'home.html',
        {'date': datetime.now().year,'login' : current_user})
```

This specifies where to send a user when either they log in to our application or they log out of our application: We make these changes in our *settings.py* file by adding these two statements. So either way they are being redirected back to our home page

```
LOGIN_REDIRECT_URL = 'home'
LOGOUT_REDIRECT_URL = 'home'
```

Creating the templates to support Login and Signup

We need to create a folder called ***registration*** in our templates folder because that is where django will look for the login.html by default

date_track/templates/registration/login.html

```
{% extends "base.html" %}
{%  block content %}

<h2>Login</h2>
<form method="post">
  {% csrf_token %}
  {{ form.as_p }}
  <button type="submit">Login</button>
</form>
{% endblock %}
```

The **csrf_token** protects against *cross site request forgery* attacks. This is where an attacking site tricks a user's browser into logging into a site with someone else's credentials. This should be used whenever the **post** method is used.

All the form's fields and their attributes will be unpacked into the **{{ form }} variable** .
form.as_p renders the form fields in **<P>** HTML paragraph tags

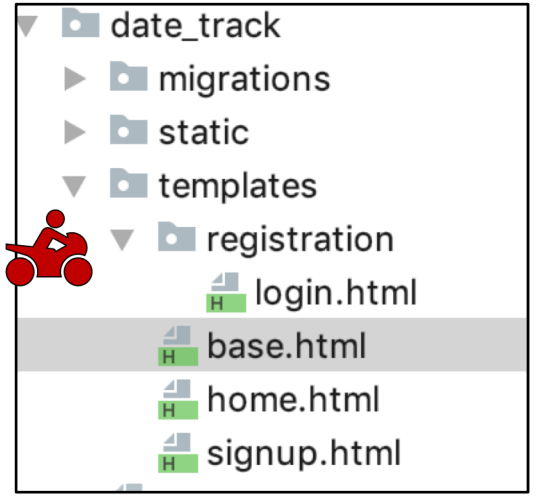
date_track/templates/signup.html (same as where home.html is)

```
{% extends "base.html" %}
{%  block content %}

<h2>Sign up</h2>
<form method="post">
  {% csrf_token %}
  {{ form.as_p }}
  <button type="submit">Sign up</button>
</form>
{% endblock %}
```

date_track/templates/base.html (same as where home.html is)

This is the template from which all other templates inherent

```
{% if user.is_authenticated %}
      <li class="nav-item" class="col px-md-5">
          <a class="nav-link"><font color =#ff00ff><i> Hi {{ user.username }}  </i></font></a>
      </li>
      <li class="nav-item">
          <a class="nav-link" href=" {% url 'logout' %}">Log out</a>
      </li>
{% else %}
      <li class="nav-item">
          <a class="nav-link" href="{% url 'signup' %}"><font size = "-1">Sign Up</font></a>
      </li>
      <li class="nav-item">
           <a class="nav-link" href="{% url 'login' %}"><font size = "-1">Login</font></a>
      </li>
{% endif %}
```

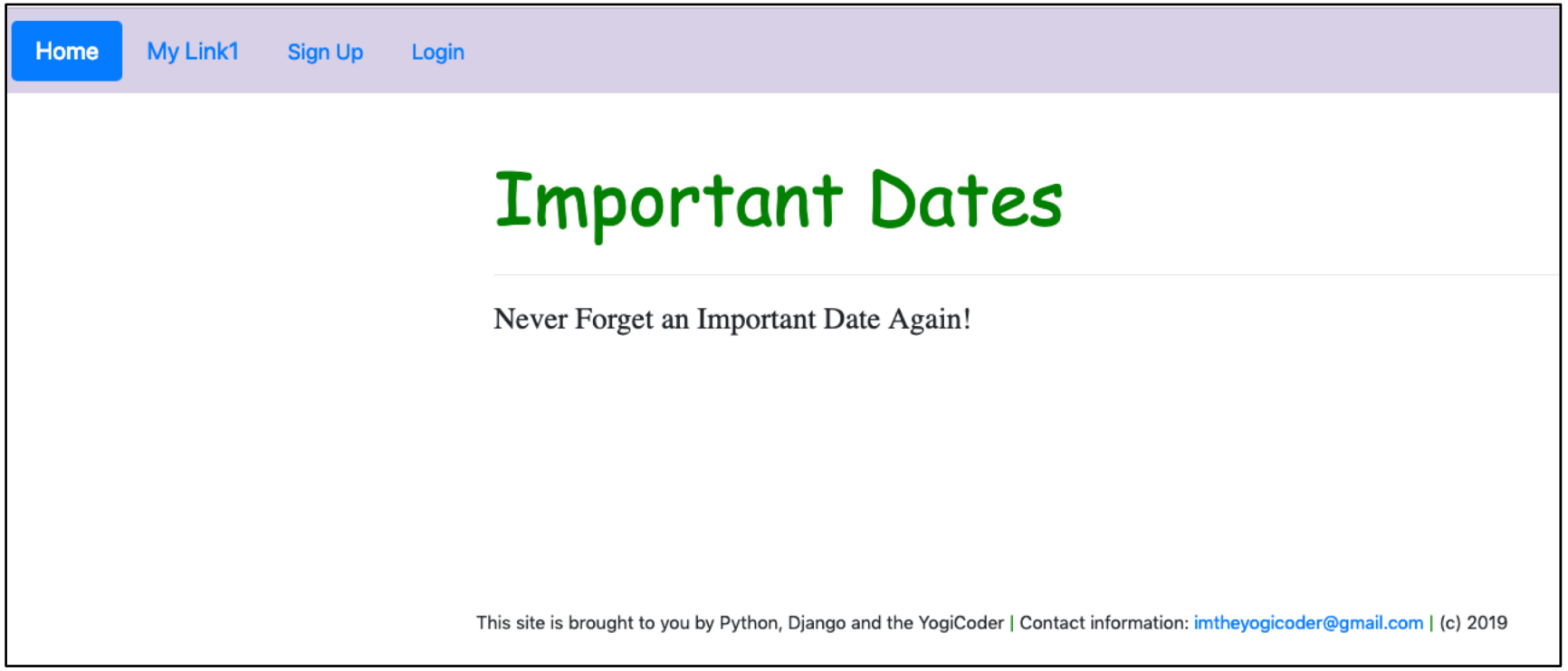


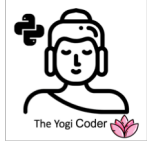
Creating the Signup view

users/views.py

```
1
2   from django.urls import reverse_lazy
3   from django.views import generic
4
5   from .forms import CustomUserCreationForm
6
7   class SignUp(generic.CreateView):
8       form_class = CustomUserCreationForm
9       success_url = reverse_lazy('login')
0       template_name = 'signup.html'
1
2
```

This is what the home page looks like now





Installing and Configuring Django Allauth

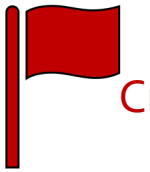
Adding Django allauth

What is django allauth?

- It is a Integrated set of Django applications addressing authentication, registration, account management as well as 3rd party (social) account authentication (like Gmail, Facebook, etc.)

Install django allauth?

```
(venv) Claudias-iMac:ImportantDates claudiaacerra$ pip3 install django-allauth
```



Critically important to use pip3 and not pip to install this

Changes to the settings.py

1. This is already set, just wanted to point out the dependency

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',#allauth needs this
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
```

2. We have to add this entire section new

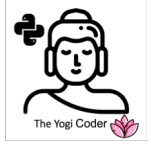
```
AUTHENTICATION_BACKENDS = (
    # Needed to login by username in Django admin, regardless of `allauth`
    'django.contrib.auth.backends.ModelBackend',

    # `allauth` specific authentication methods, such as login by e-mail
    'allauth.account.auth_backends.AuthenticationBackend',
)
```

3. In INSTALLED_APPS we add these

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django.contrib.sites',
    'date_track.apps.DateTrackConfig',
    'users',
    'allauth',
    'allauth.account',
    'allauth.socialaccount',
    'allauth.socialaccount.providers.twitter',


]
```

4. 3 other settings

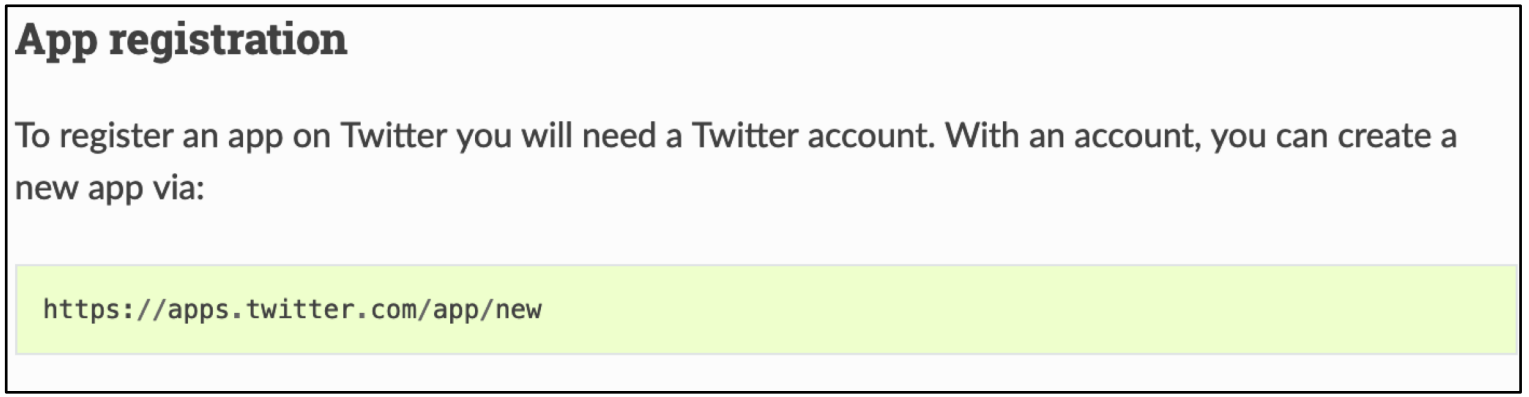
```
#for 3rd party provider setup in admin console
SITE_ID = 1

# so they can provide email instead of user name
ACCOUNT_EMAIL_REQUIRED = False
ACCOUNT_USERNAME_REQUIRED = False
```



Configuring Twitter to Use As a Signup and Login Method

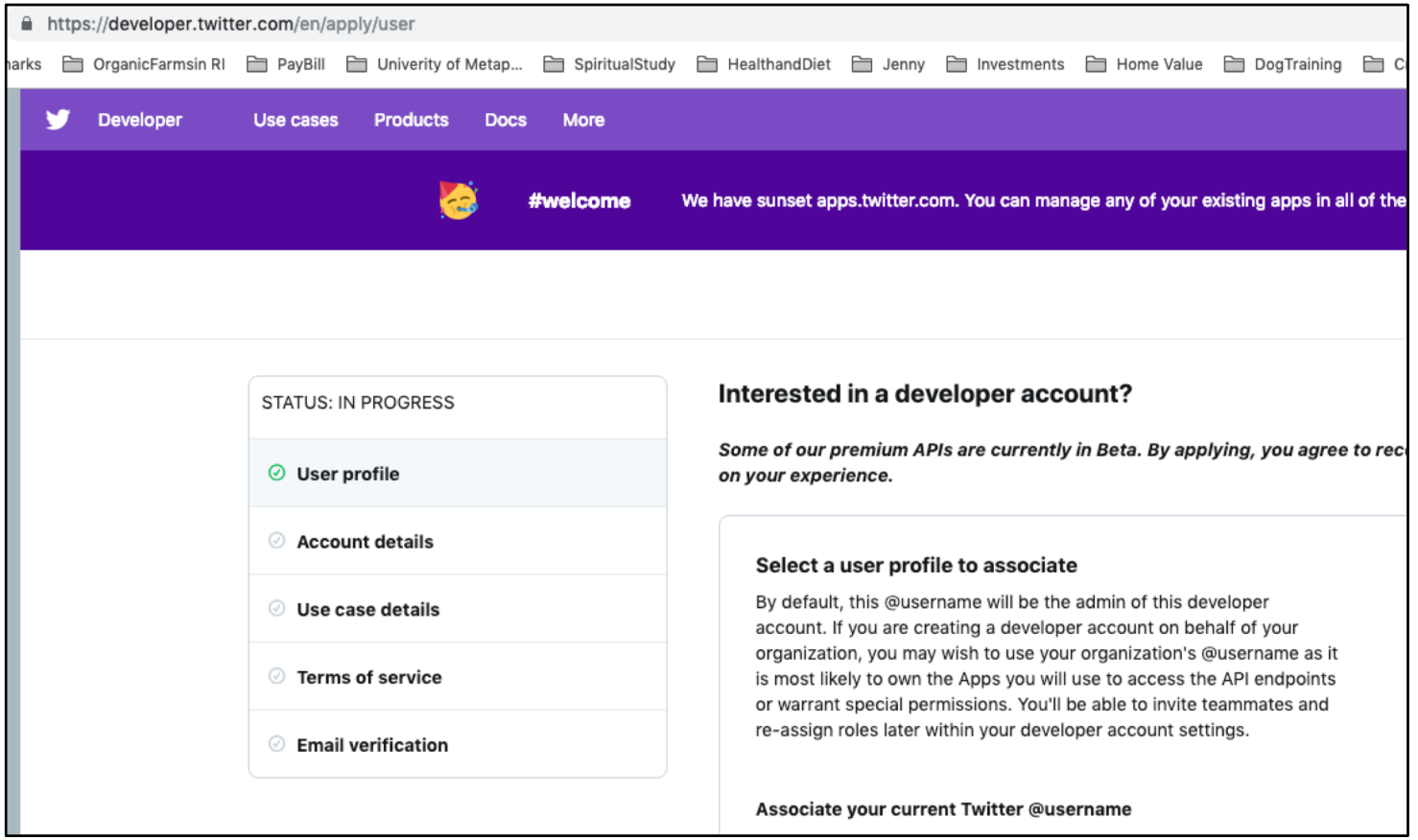
Configure a Twitter Account

If you first try to register an application on twitter, it will ask to create a ***developer account***. So you will go through those required steps first, before you create your "app"



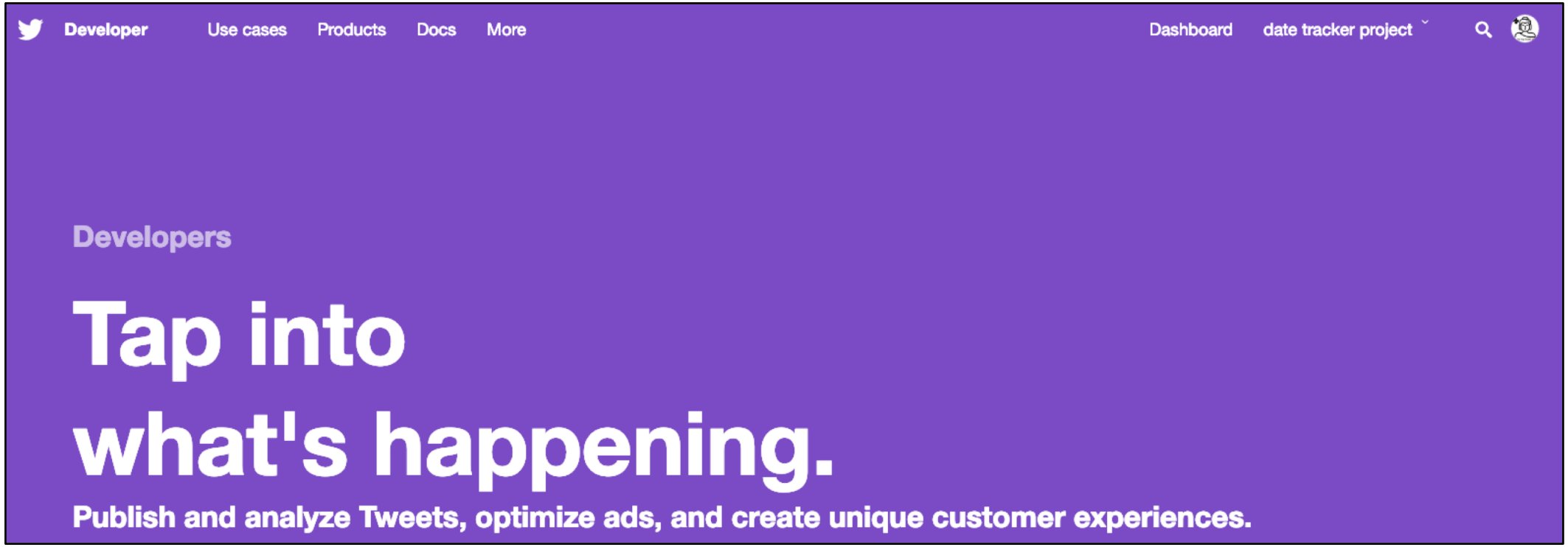
Associated your twitter account and hit ***continue***

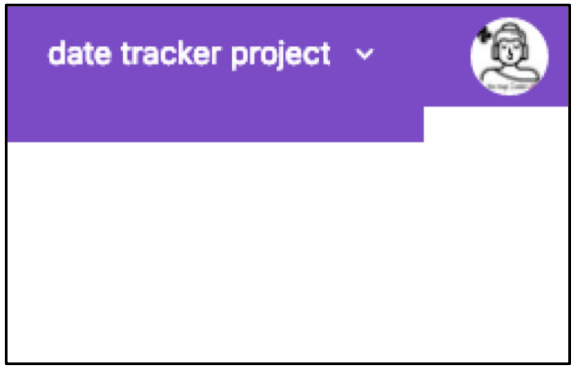
Add you ***project name***, for me it is ***date tracker*** project



Configure a Twitter Account To Authenticate By

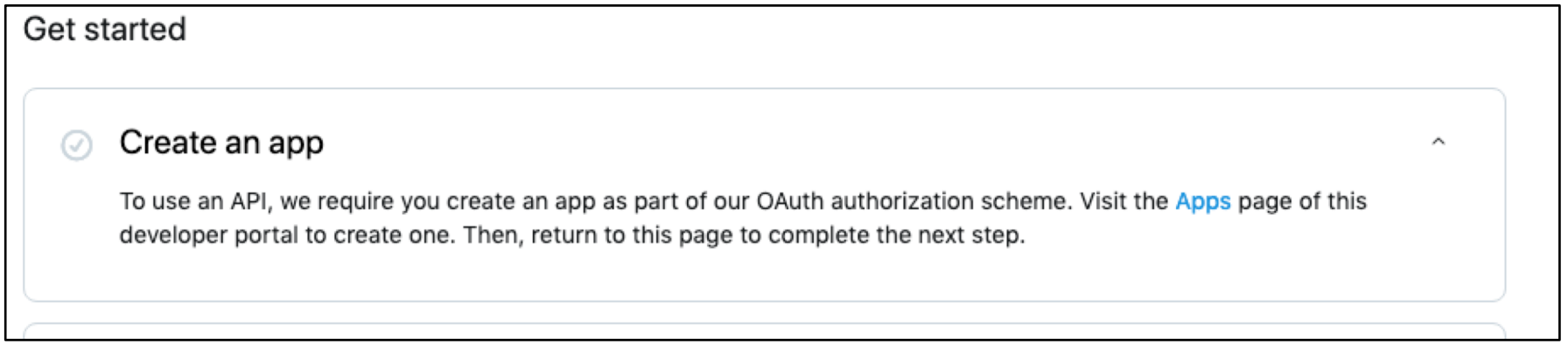
Will ask you a bunch of questions, answer the questions
and then get an email to confirm in order to proceed





Use the pull down to get to the "***get started***" option

Then you'll see the ***Create an app*** option, choose *this* option

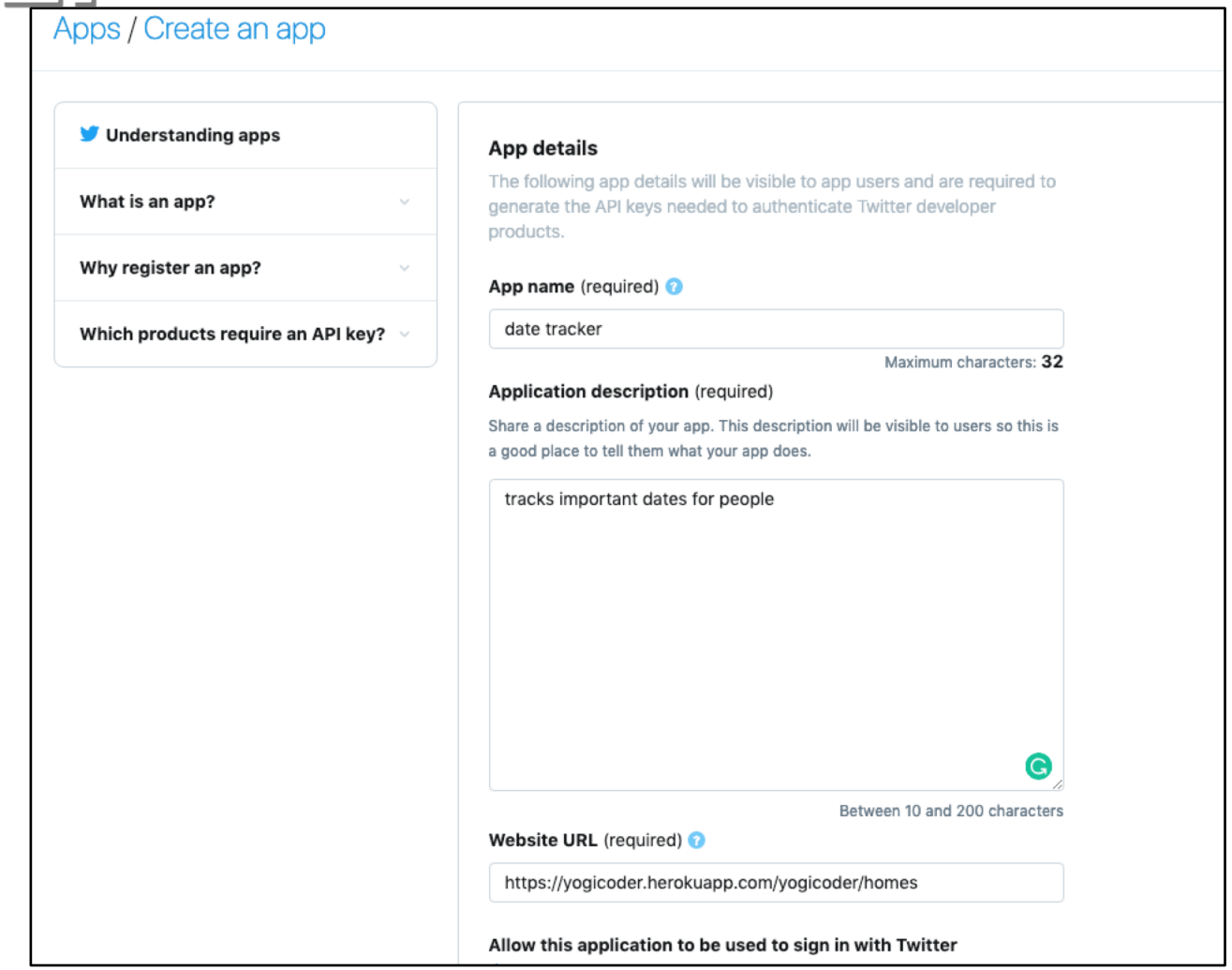


And finally here you're able to create an application

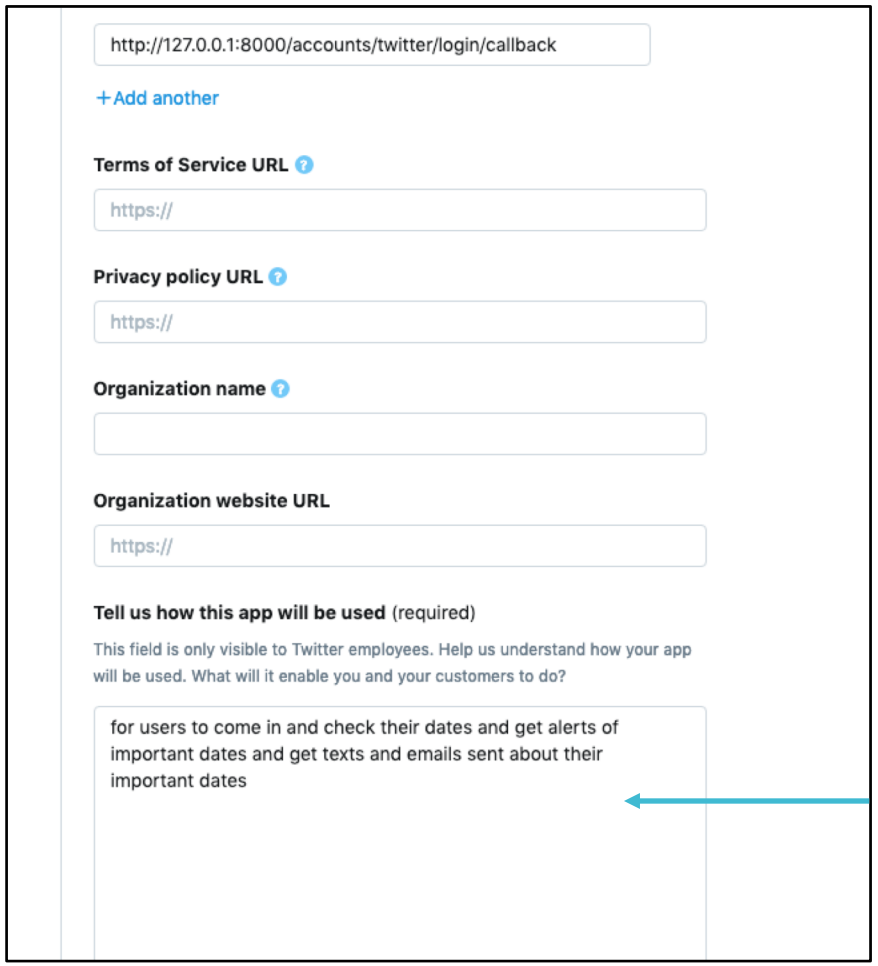
Filling out the details of the App creation

VERY IMPORTANT: enter this in the "**callback URL field**"

1.

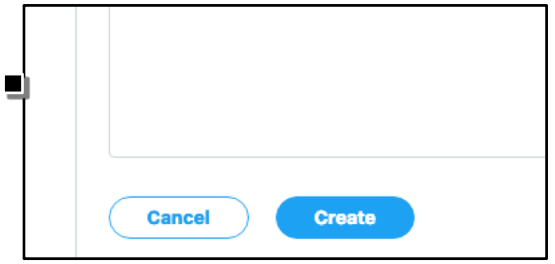


2. **http://127.0.0.1:8000/accounts/twitter/login/callback**



Add text here and Twitter has many minimum text amounts required

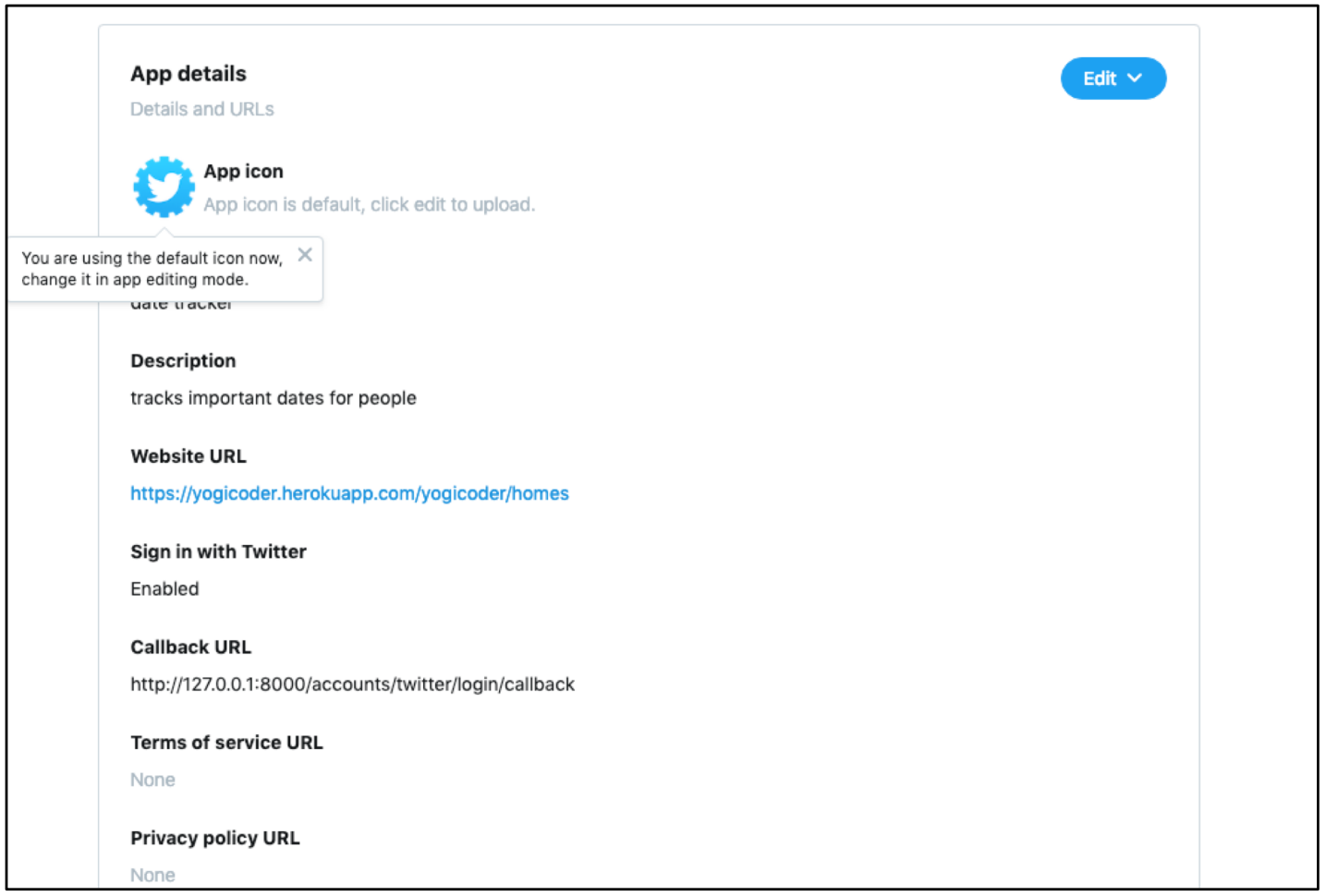
3.



Select "**create**"

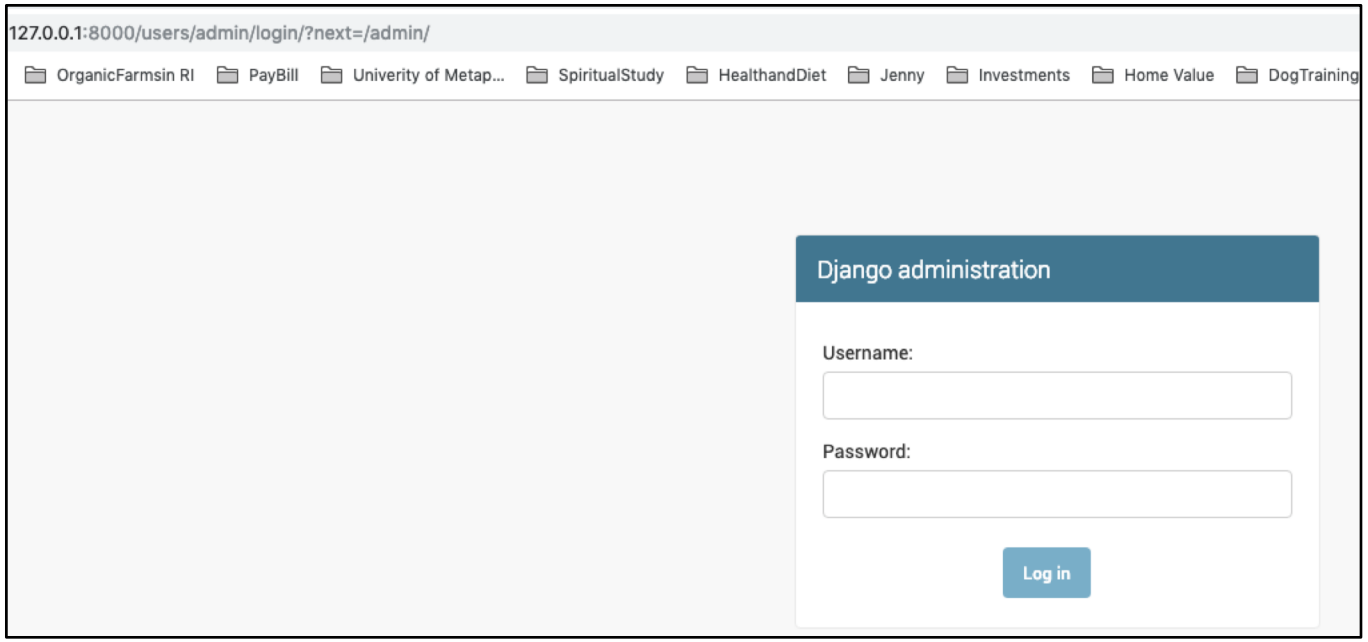
I was forced to give a real URL and I do not have one for this application because its in development, so I gave the yogicoder site. I can always change that later (Twitter says "the website will be used to source attribution for Tweets created by your app and will be used in user-facing authorization screens")

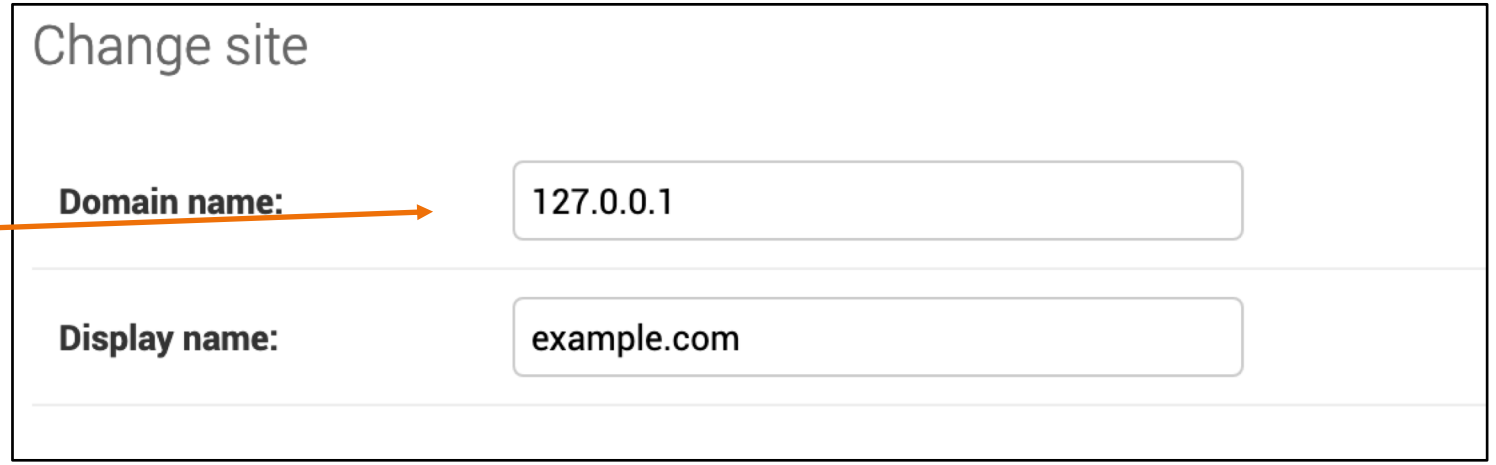
You'll now get confirmation that looks like this



Complete your twitter Configuration in Django admin

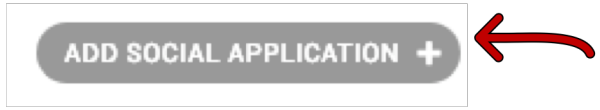
- Access, http://127.0.0.1:8000/admin
- And enter with your ***superuser credentials***
- Then go into "***SITES***"
- And click on the one website that is there called ***example.com***
- Change the domain name to your ***dev servers IP address*** as seen here

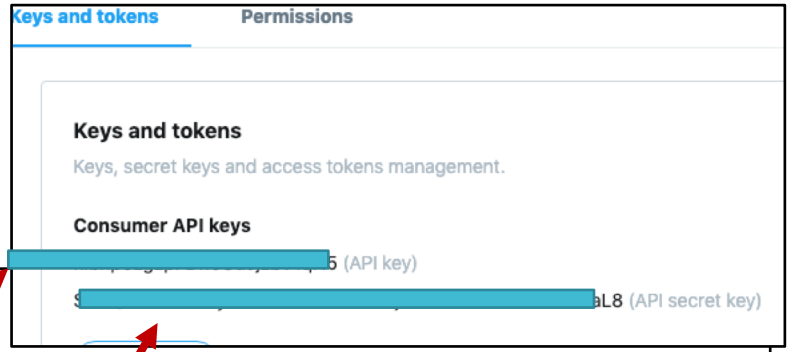




1. In Django Admins' home page…select the **Social applications** link

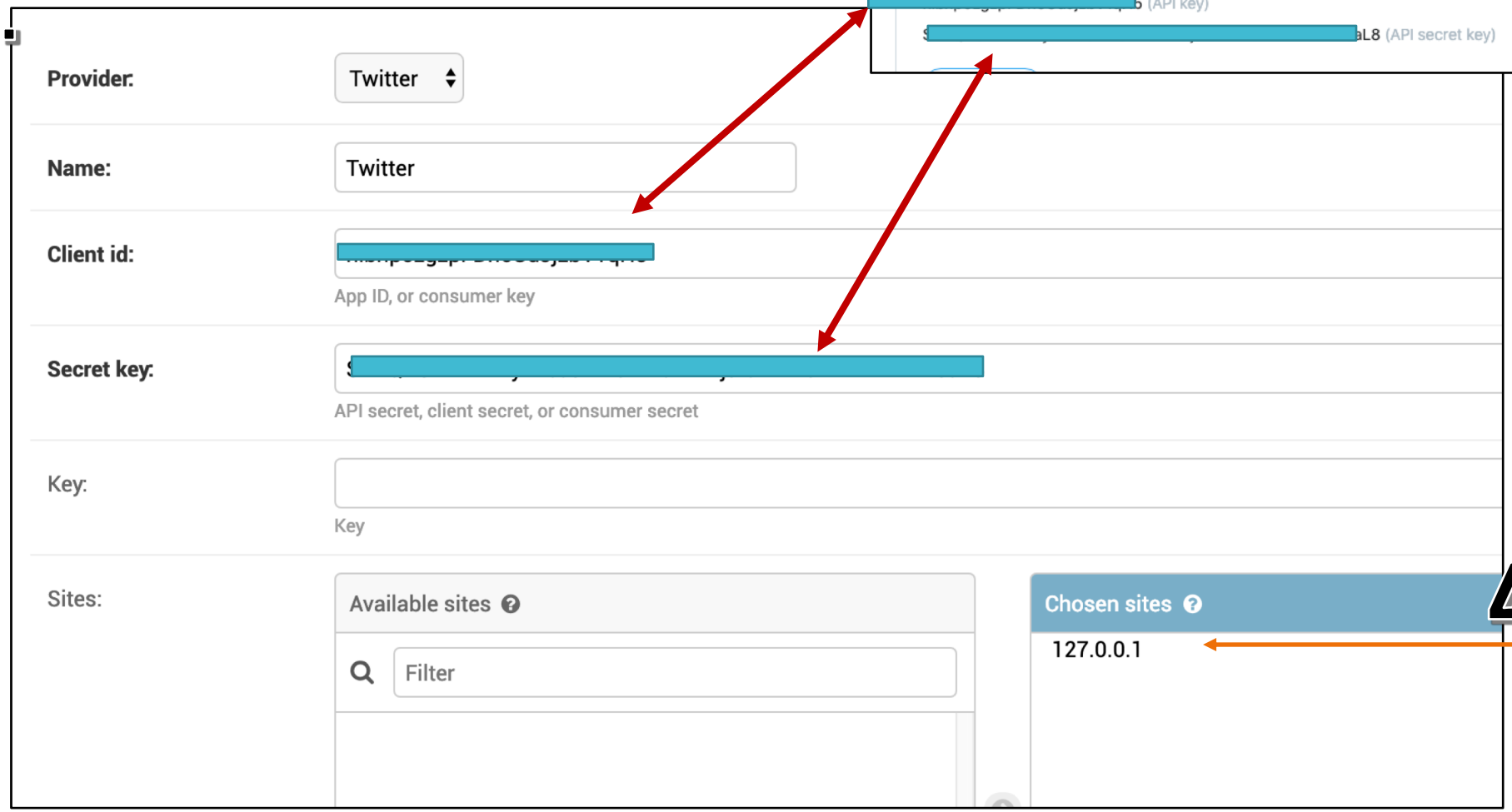
2. Select Add Social Application



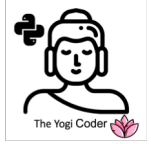


- Get ***Client ID*** from API Key con the keys and tokens tab in your Apps details on **Twitter**
- And ***Secret key*** use the API secret key on **Twitter**
- "***Key***" is not needed, leave it blank

3.

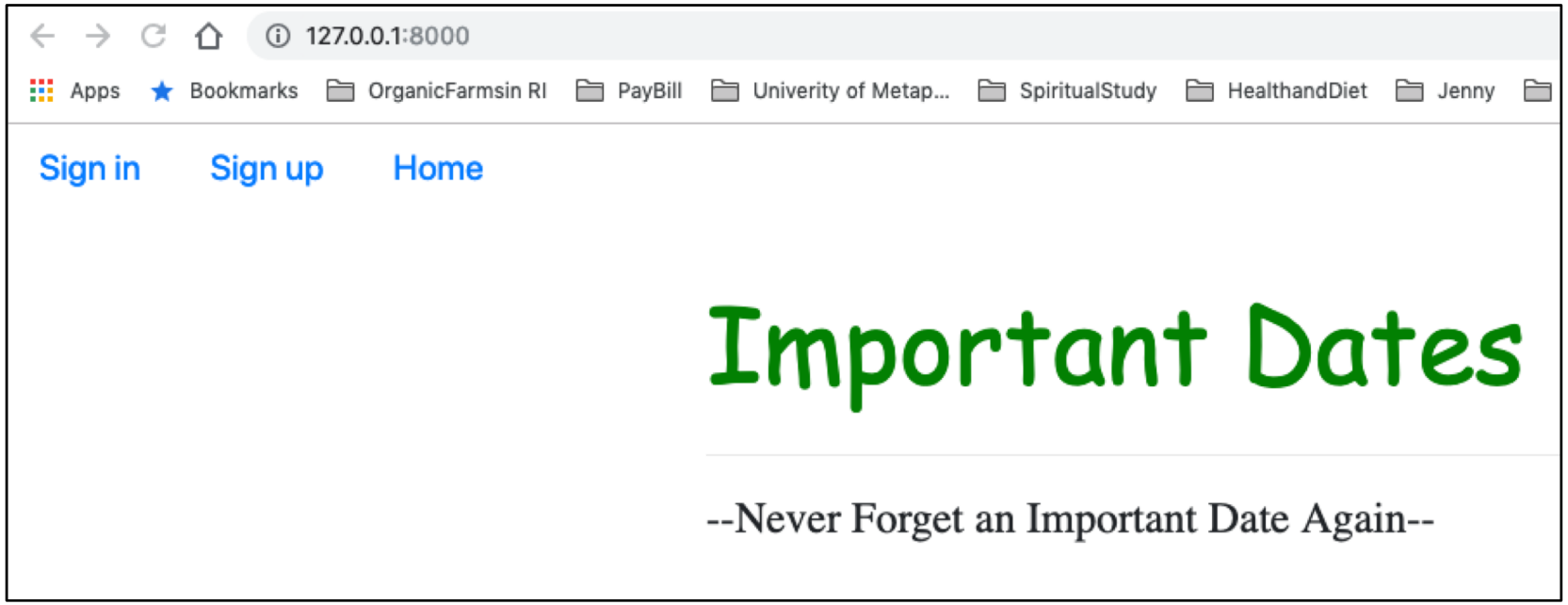


4. ***Make sure*** you choose this site from "available sites" (left hand side of page under sites)



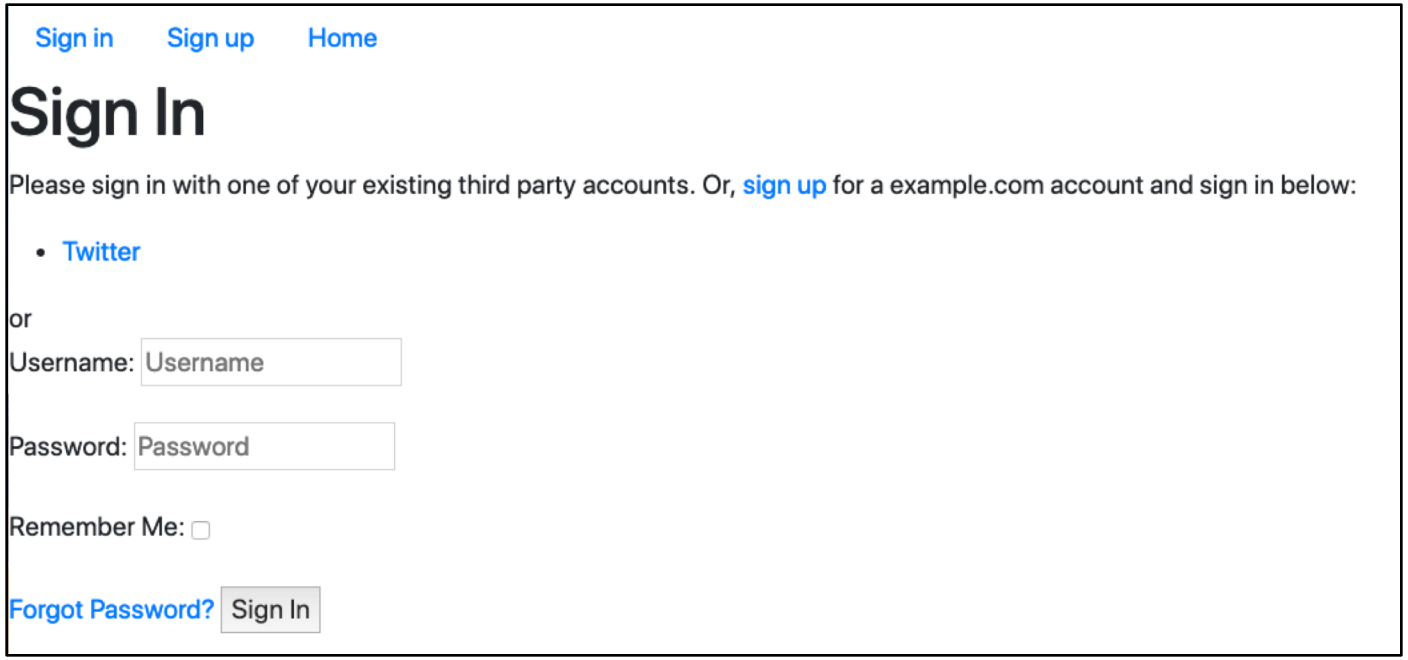
Fire up your development server and lets test it out

1. Start your Development Server and Access Your Applications' Home Page



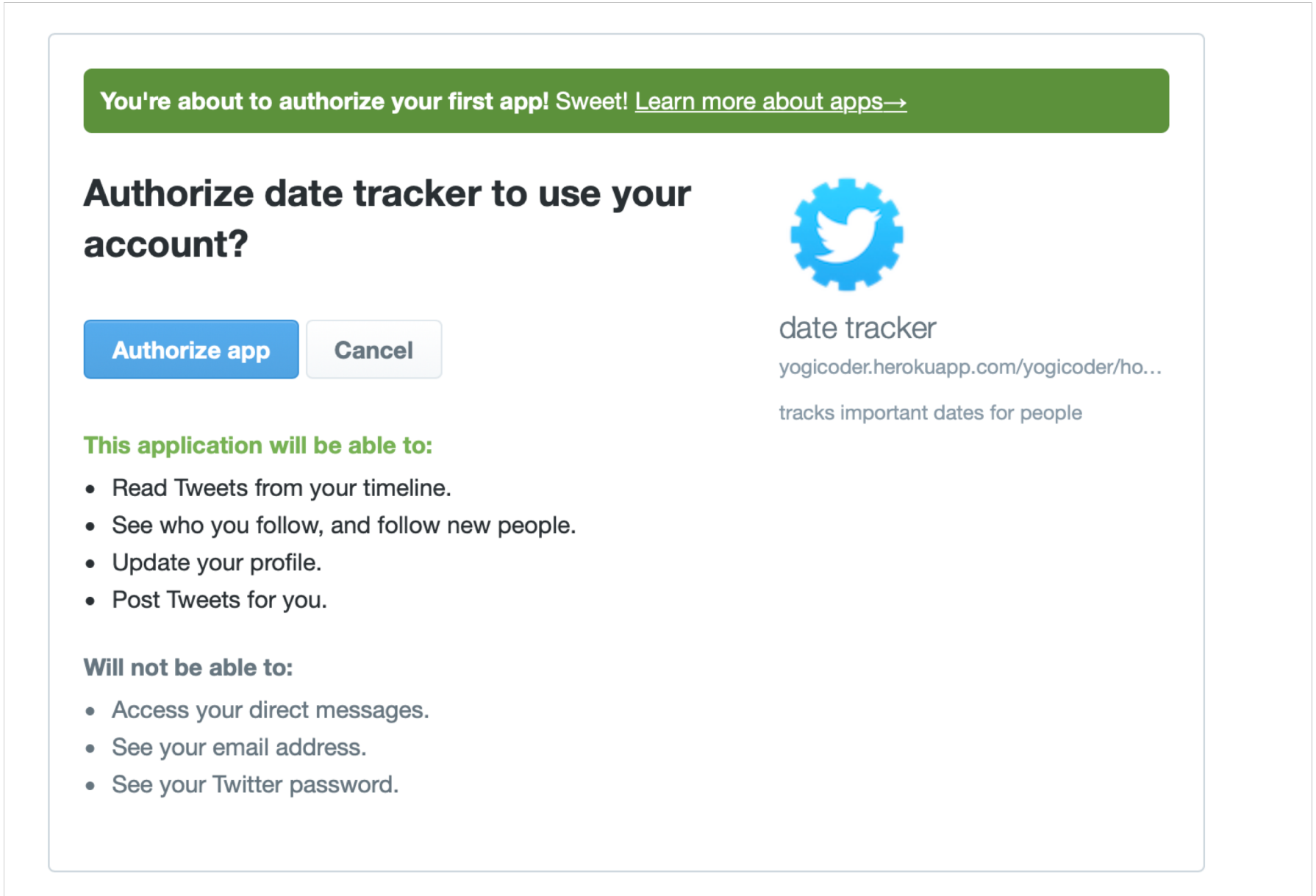
2. Choose ***Sign in*** and you will be able to sign up with your **Twitter** Account

Click the "Twitter" link



You will then see this Twitter Confirmation Page

Choose "***Authorize app***" to continue



1. Then supply the *email account* associated with your Twitter account

Menu:

- Sign In
- Sign Up

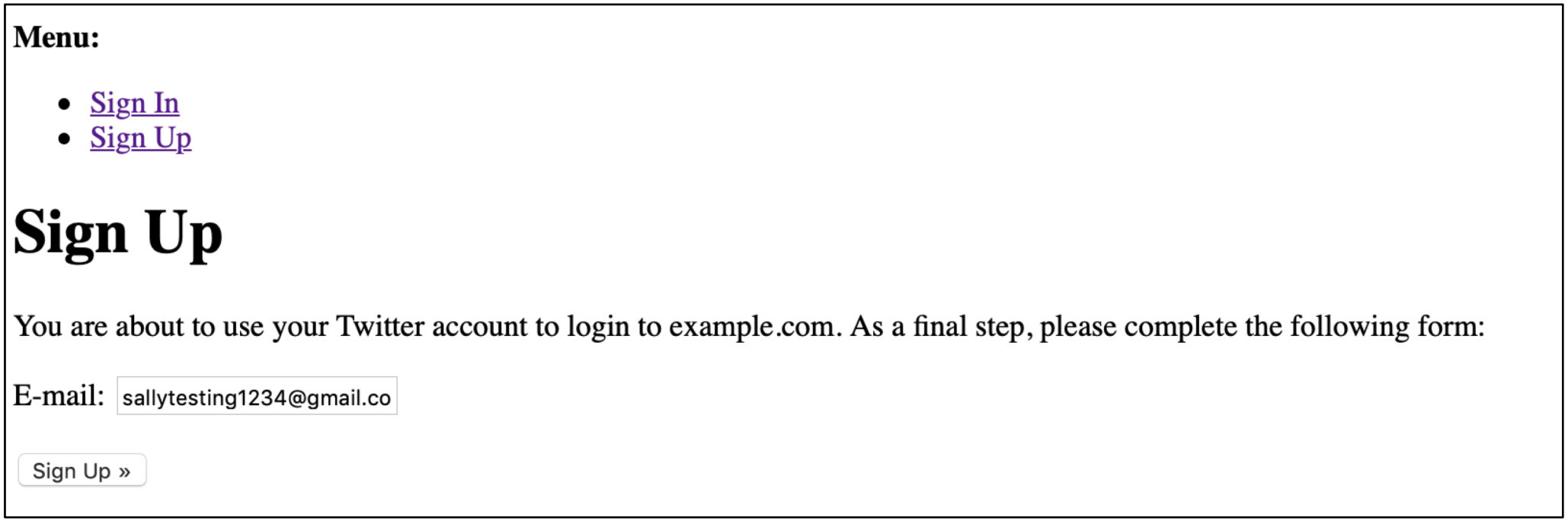
Sign Up

You are about to use your Twitter account to login to example.com. As a final step, please complete the following form:

E-mail: E-mail address

Sign Up »

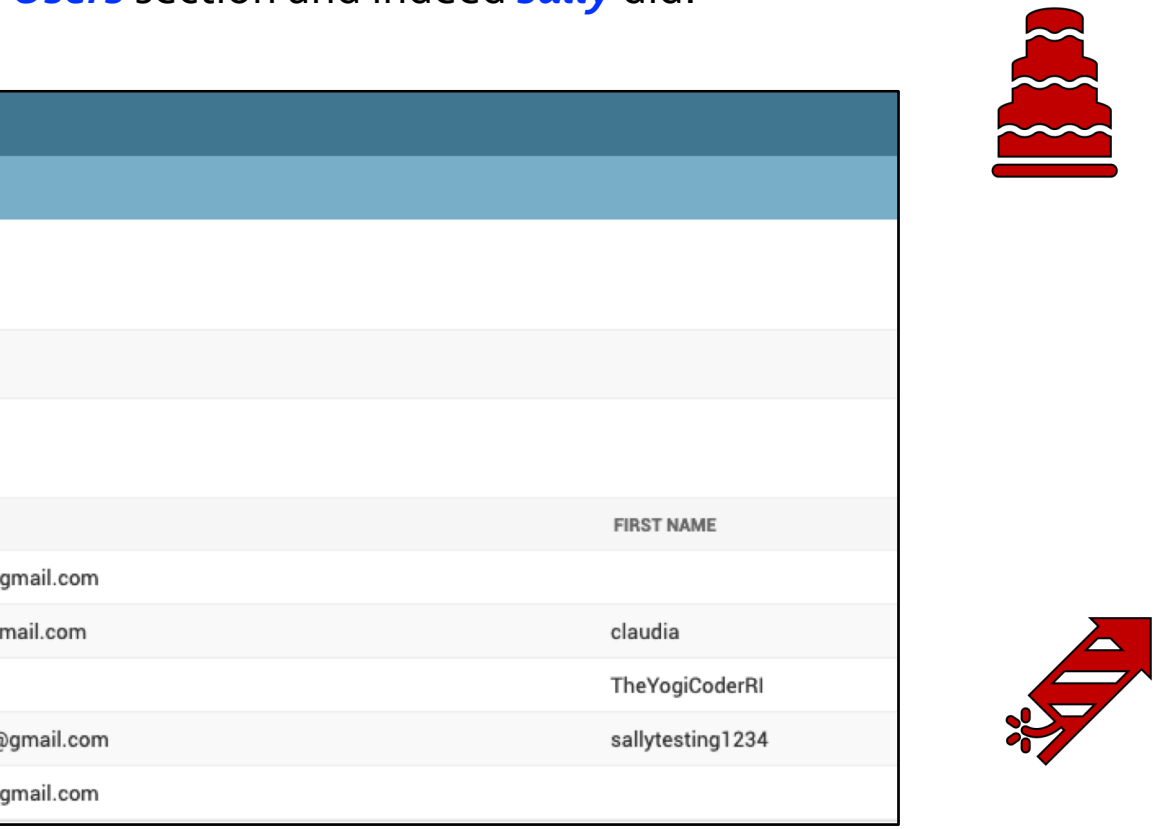
1. Enter the ***email account*** associated with your twitter account

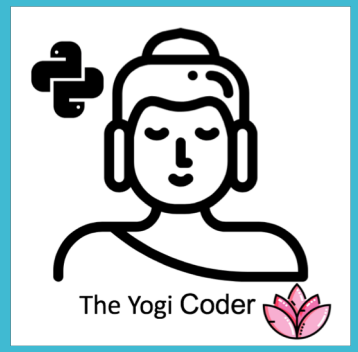


2. Select "**Sign Up**"

Access the Django Admin Application Again

Check that the user got created in the ***Users*** section and indeed ***sally*** did!




The Yogi Coder