

# Gold Team

Venti  
Spring 2018

[venti Github](#)

## Team Overview

Team Member Name	Github Usernames
Ethan Johnson	predhelt
Terry Peters	terrancejpeters
Hunter Breen	hunnabreen
Natalie Slabczynski	nslabczynski
Tom Hubbard	thubbard0

## Venti Overview

Our application Venti combines multiple kinds of social media to create a daily forum where users can respond in 256 characters to the prompt of the day. Users can also vote for their favorite posts, and the most popular posts' posters will suggest topics for the next day. The whole community will get to vote on the next day's topic.

## User Interface

Venti's user interface focuses on simplicity as a means of error prevention.

compsci326 New Post Logout

Your account doesn't have access to this page. To proceed, please login with an account that has access.

## Log in

Log in and tell us what's on your mind!

Username:

Password:

[Lost password?](#)

---

© 2015-2017 compsci326. All rights reserved.

[About](#)

Figure 1 (Login page)

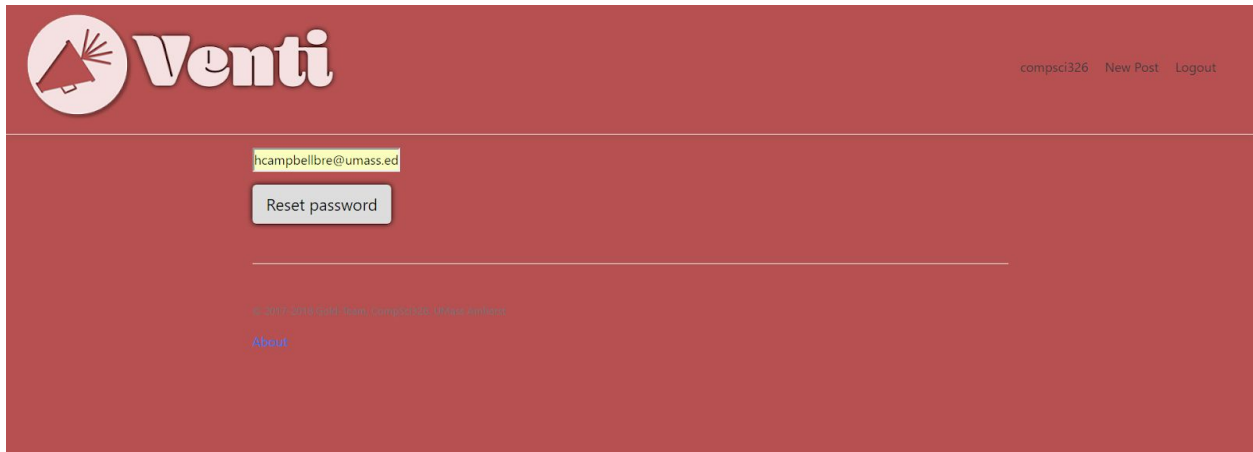


Figure 2 (Password Reset)

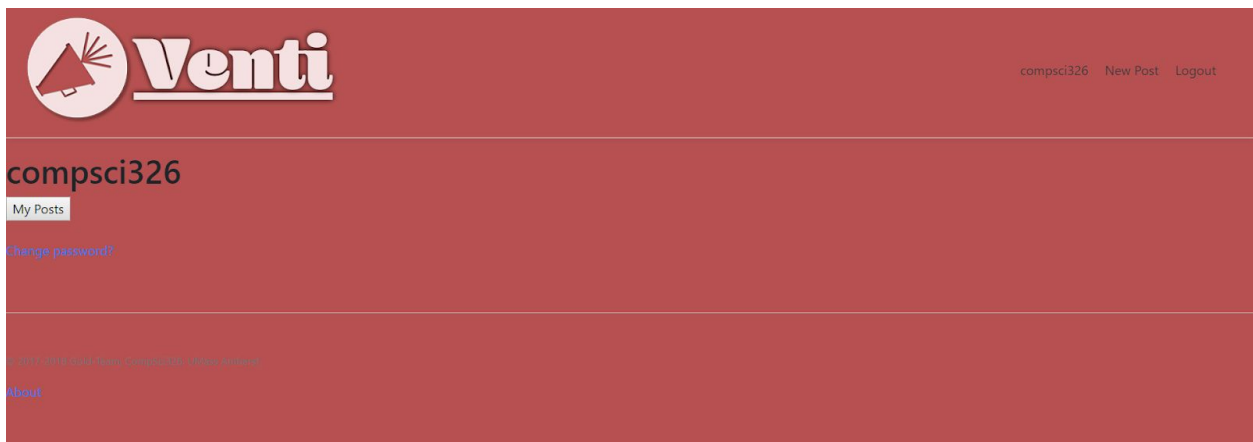


Figure 3 (Profile Page)

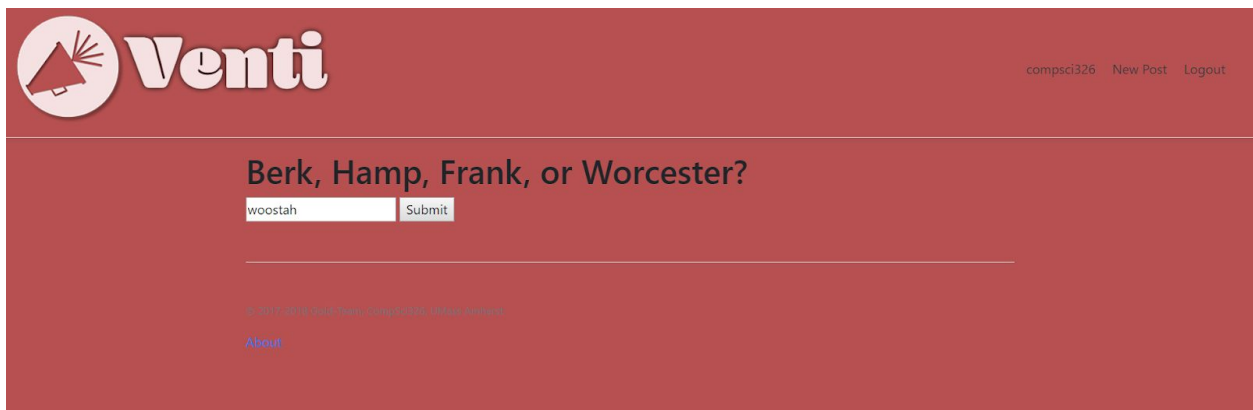
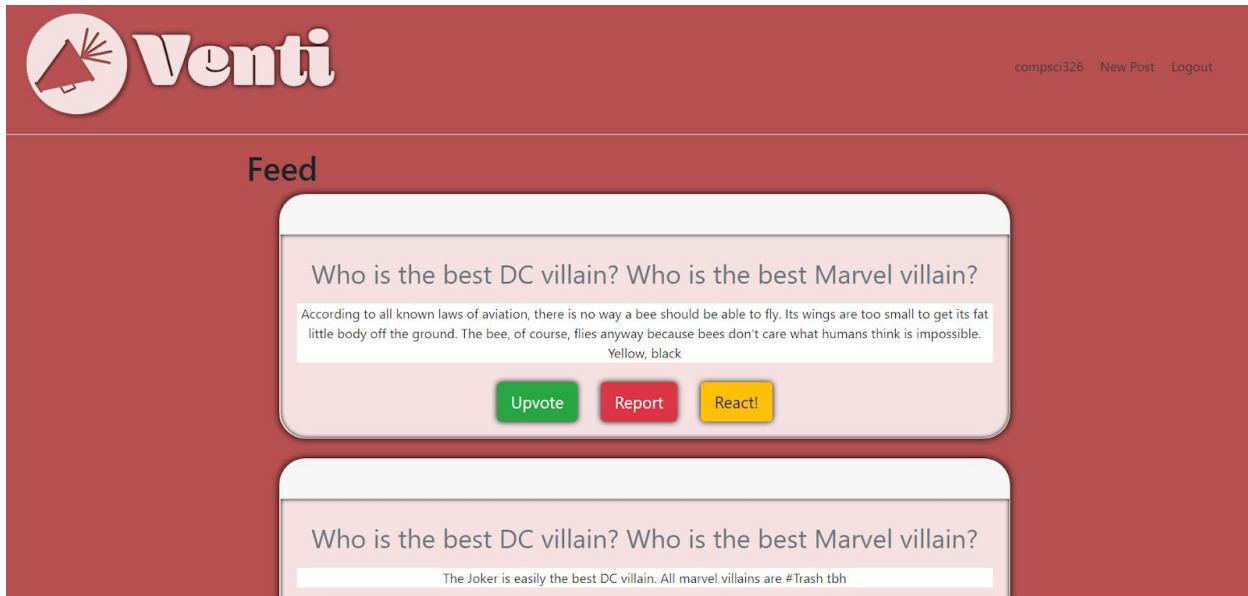
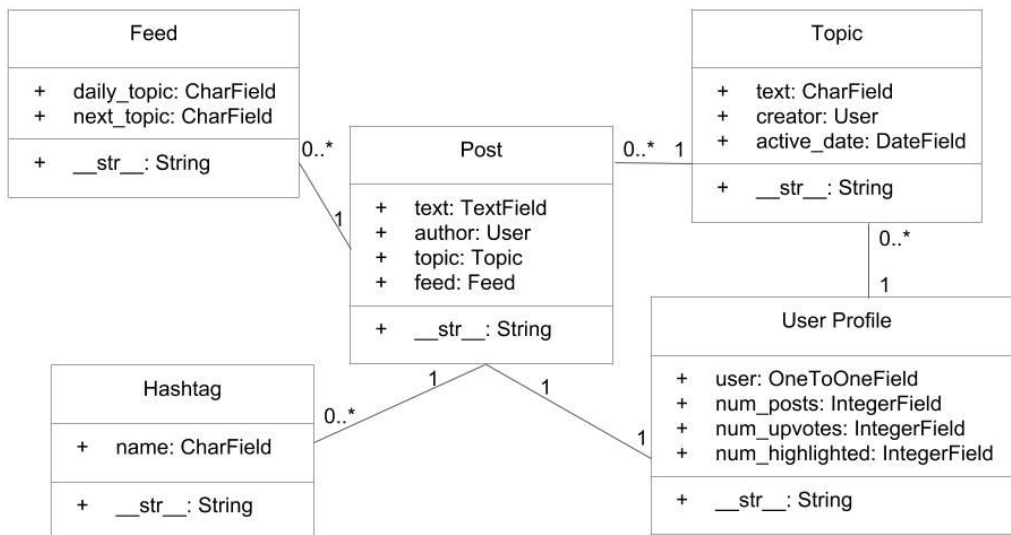


Figure 4 (Posting Page)



**Figure 5 (Feed)**

## Data Models



**Figure 6 (Data Models)**

As seen in Figure 1, we have a total of five different data models we used within our application. Our most centric model in the application is “Post.” This model contains the actual text that comprises the post, and also has to contain the User who wrote it, the Topic that it is talking about, and the Feed which it is appearing on. All the models branch off of Post. “Hashtag” can be contained within a post, as it is just a string of characters (ex. #YOLO), however a Post doesn’t have to contain a Hashtag. A topic is what Posts are written about, and they contain the text that actually comprises the topic, as well as the User that created it, and the date it was created. A Post has to have a topic, and a Topic has to have a User that created it. A User Profile contains the user (who actually created it), and then counts for the

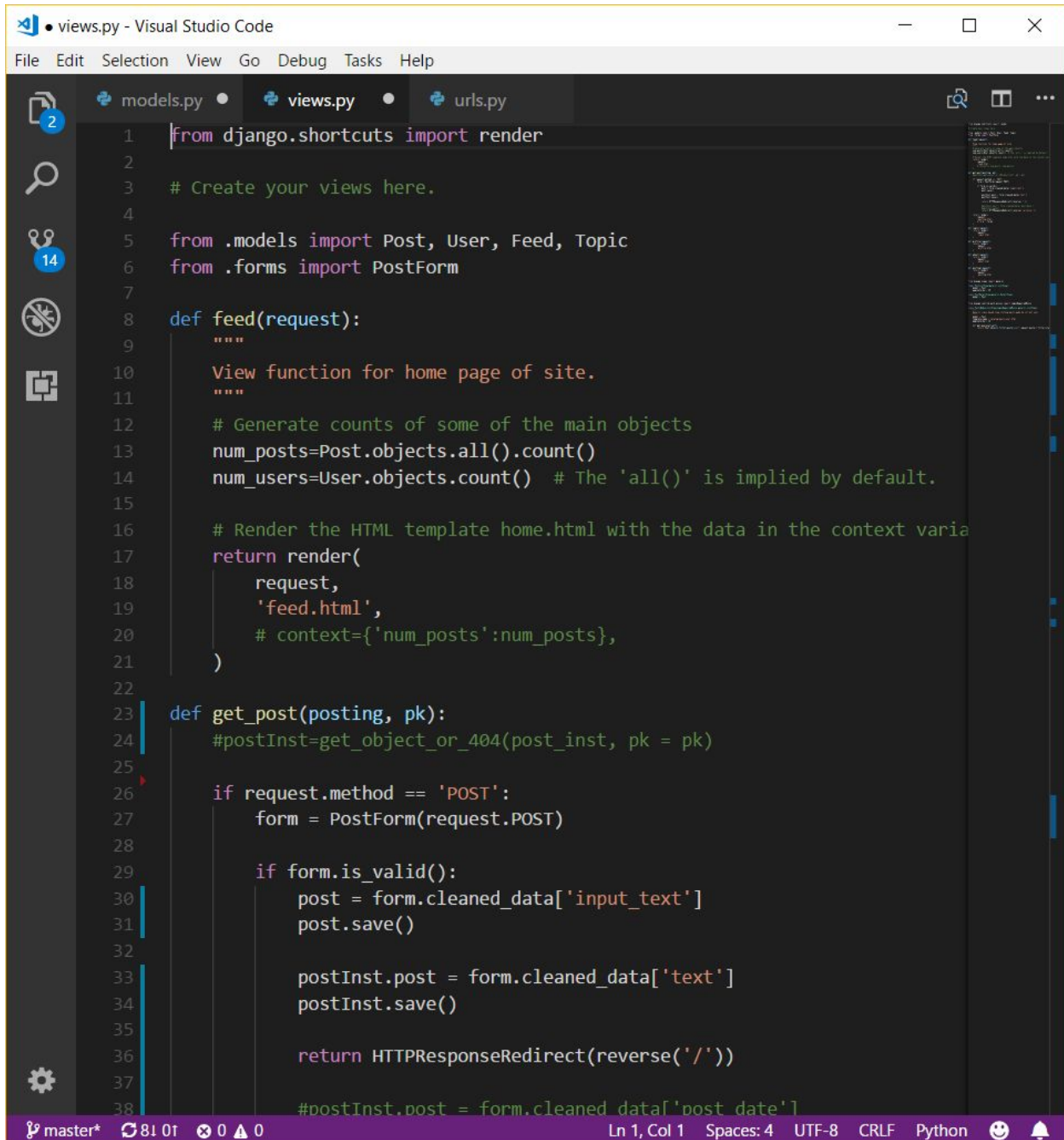
amount of posts, upvotes, and highlighted topics/posts that they have. Finally, there is a feed that contains the current topic as well as the next topic to be displayed.

### URL Routes/Mappings

URL	Description	Permissions
''	Path to the home page	
'login/'	Path to login	Must not be logged in
'profile/'	Path to view own profile	Must be logged in; can only see your own
'about/'	Path to information about Venti	
'newpost'/'	Path to make a post	Must be logged in as a user
'poll/'	Path to vote on daily topic	Must be logged in as a user
'poll/results''	Path to view the results of the poll	
'myposts/'	Path for users to view their own posts	Must be logged in; can only view your own
'accounts/'	For site authentication urls for login, logout, password management	

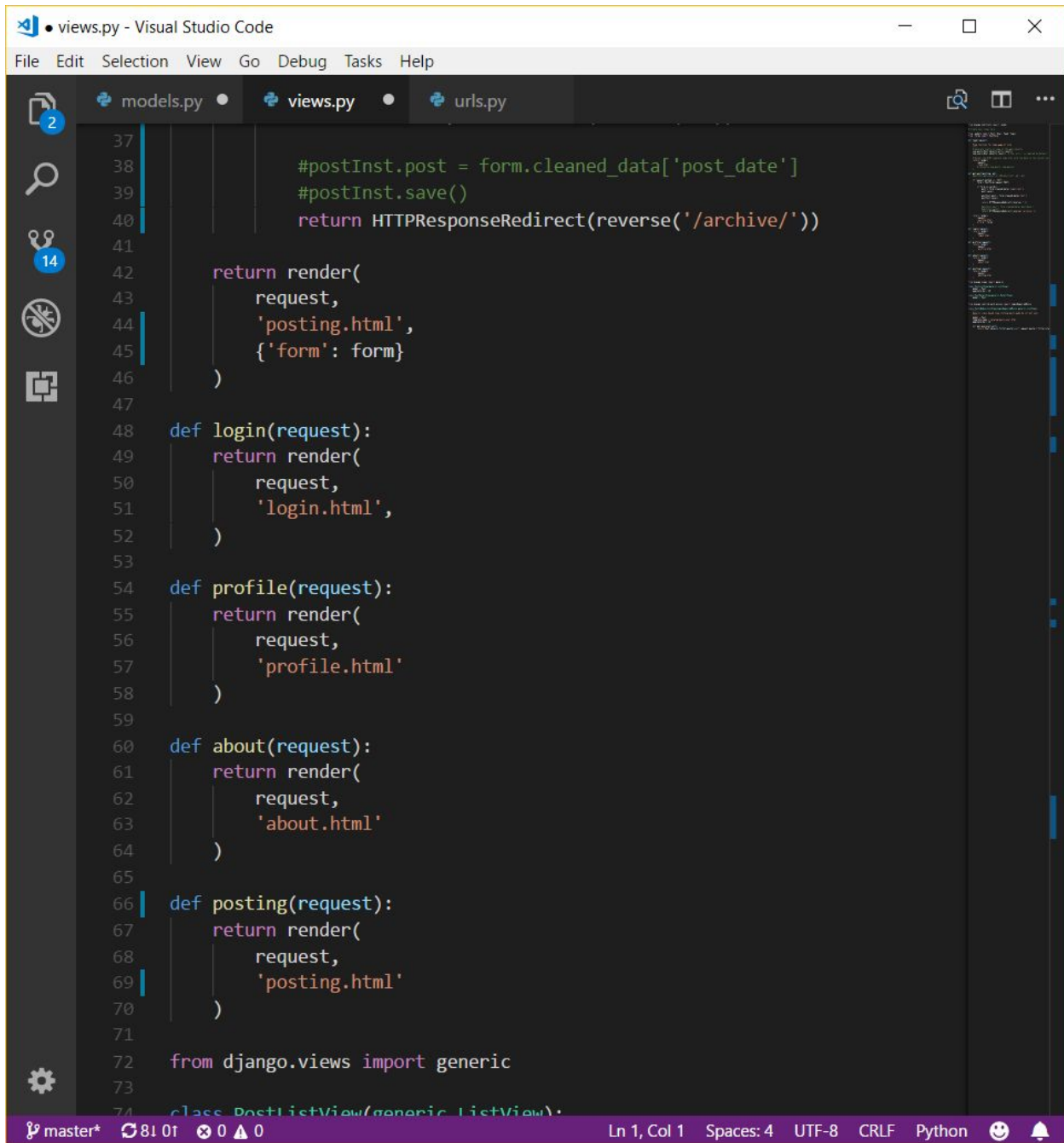
**Figure 7 (URL Paths)**

## Views



```
1 from django.shortcuts import render
2
3 # Create your views here.
4
5 from .models import Post, User, Feed, Topic
6 from .forms import PostForm
7
8 def feed(request):
9     """
10     View function for home page of site.
11     """
12     # Generate counts of some of the main objects
13     num_posts=Post.objects.all().count()
14     num_users=User.objects.count() # The 'all()' is implied by default.
15
16     # Render the HTML template home.html with the data in the context variable
17     return render(
18         request,
19         'feed.html',
20         # context={'num_posts':num_posts},
21     )
22
23 def get_post(posting, pk):
24     #postInst=get_object_or_404(post_inst, pk = pk)
25
26     if request.method == 'POST':
27         form = PostForm(request.POST)
28
29         if form.is_valid():
30             post = form.cleaned_data['input_text']
31             post.save()
32
33             postInst.post = form.cleaned_data['text']
34             postInst.save()
35
36             return HttpResponseRedirect(reverse('/'))
37
38     #postInst.post = form.cleaned_data['post_date']
```

Figure 8 (Views)



```
37
38     #postInst.post = form.cleaned_data['post_date']
39     #postInst.save()
40     return HttpResponseRedirect(reverse('/archive/'))
41
42     return render(
43         request,
44         'posting.html',
45         {'form': form}
46     )
47
48 def login(request):
49     return render(
50         request,
51         'login.html',
52     )
53
54 def profile(request):
55     return render(
56         request,
57         'profile.html'
58     )
59
60 def about(request):
61     return render(
62         request,
63         'about.html'
64     )
65
66 def posting(request):
67     return render(
68         request,
69         'posting.html'
70     )
71
72 from django.views import generic
73
74 class PostListView(generic.ListView):
```

Figure 9 (Views, cont.)

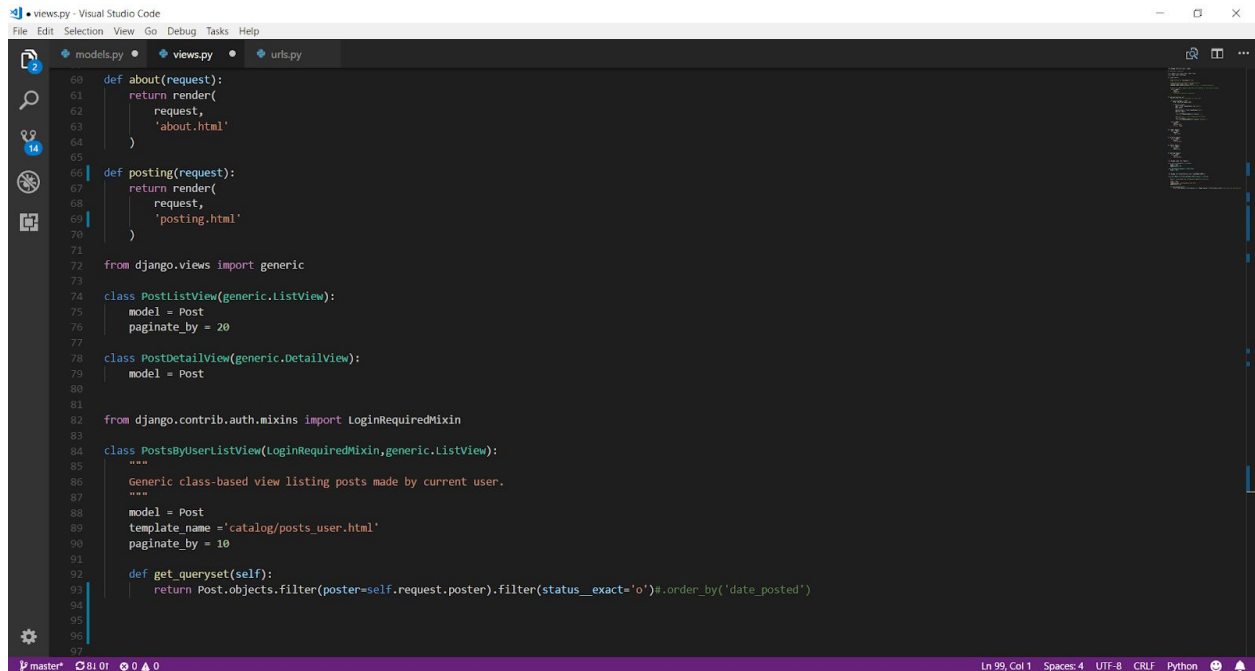
A screenshot of the Visual Studio Code editor interface. The top bar shows the file explorer with three files: models.py, views.py, and urls.py. The views.py file is open in the editor, showing Python code for Django views. The code includes functions for 'about' and 'posting', and class-based views for 'PostListView', 'PostDetailView', and 'PostsByUserListView'. The 'PostsByUserListView' class uses 'LoginRequiredMixin' and 'generic.ListView'. The 'get\_queryset' method filters posts by the current user and orders them by date posted. The status bar at the bottom indicates the file is on the master branch, at line 99, column 1, with 4 spaces, UTF-8 encoding, CRLF line endings, and Python syntax.

Figure 10 (Views, cont.)

## Authentication/Authorization

Users are authenticated through creating a sessions framework for users who have created accounts. We also allowed users to login and logout of Venti, removing the permissions to view their profile, vote, change passwords, and make posts. The profile UI, poll results, polling, and 'my posts' views are only available to the logged in users and custom to the specific user logged in.

## Team Choice

For our team choice, we decided to make a fully functional password reset. We implemented it by taking the example from local library further. We actually got emails to be sent. We used the 'accounts/' URL path for authentication for password management, and this can be seen in Figure as seen above. We used Django's default user system to store passwords.

## Conclusion

This project really forced us to get creative with our implementations. We needed to really think about our data models and how they influence how we implement our ideas. However, our biggest struggle was communication and version control with github. We had to ensure we were all on the same page to prevent overwriting someone else's work. We also had issues with underestimating what needed to be done or how complex our ideas were. However, we still came together in the end and made it happen.