

Topic: Bulk Email Tool 18: Handing the Task To Celery

Speaker: Udemy Instructor Rathan Kumar | Notebook: Django: Automating Common Tasks



Sending bulk emails will take awhile especially if its thousands or millions of email addresses. To resolve this, we use CELERY again just like how we used it in DATAENTRY to import or export large data.

1. In EMAILS folder, create a new file, TASKS.PY

A screenshot of a code editor interface. The left sidebar shows a file explorer with a tree view of a Django project. The 'emails' folder is expanded, showing files like __pycache__, migrations, __init__.py, admin.py, apps.py, forms.py, models.py, tasks.py, tests.py, urls.py, and views.py. The 'tasks.py' file is highlighted with a red box. The main editor area shows the code for 'send_email_task' in 'tasks.py', which imports 'app' from 'autocommontasks_main.celery' and 'send_email_notification' from 'dataentry.utils'. The function 'send_email_task' is decorated with '@app.task' and calls 'send_email_notification' with the provided arguments, returning a success message. The code is highlighted with a red box.

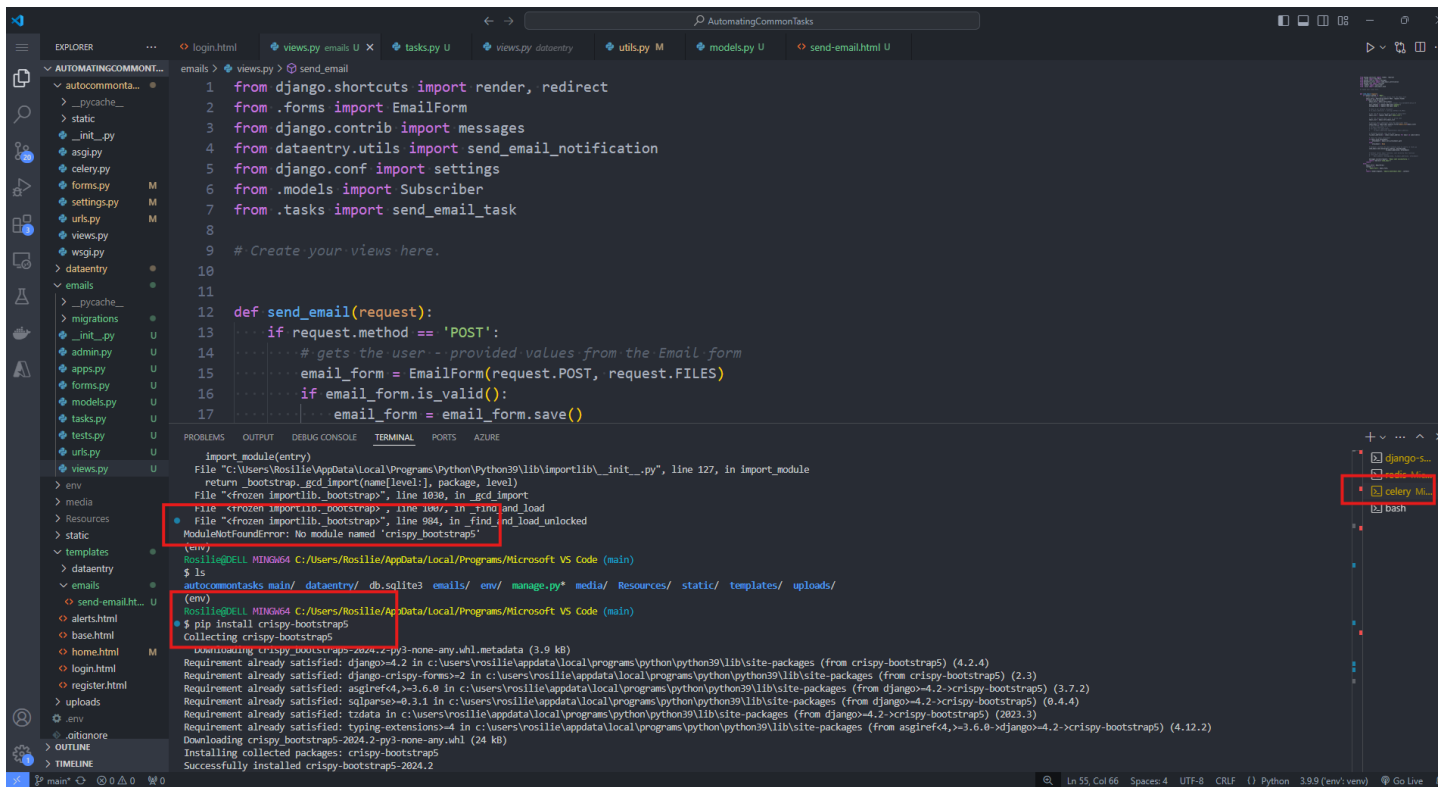
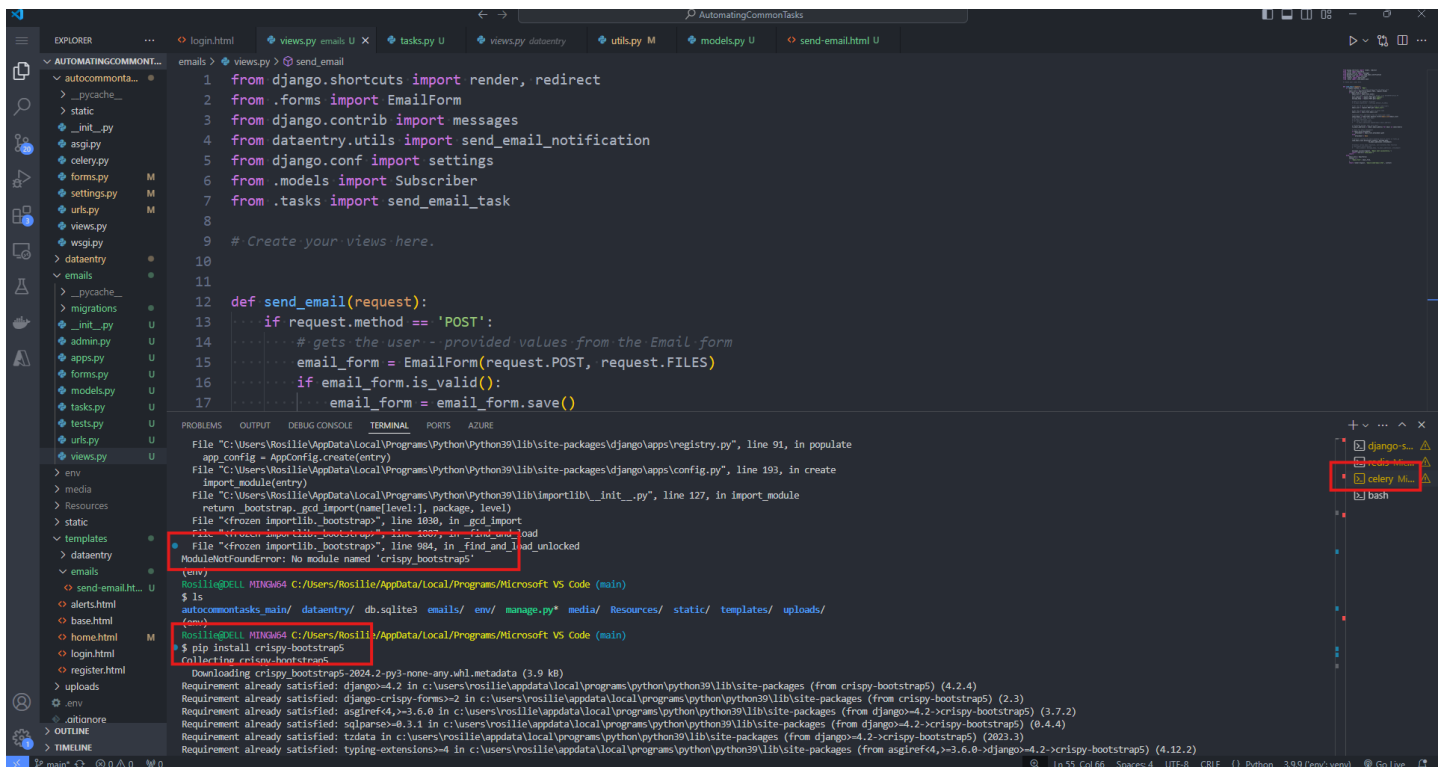
```
1 # from the main project folder within CELERY.PY, we load the variable APP
2 from autocommontasks_main.celery import app
3 from dataentry.utils import send_email_notification
4
5
6 @app.task
7 def send_email_task(mail_subject, message_body, to_email_addresses, attachment):
8     send_email_notification(mail_subject, message_body,
9                             to_email_addresses, attachment)
10     return 'Email sending task executed successfully.'
11
```

2. Update our EMAILSVIEWS.PY to call the celery function in EMAILSVIEWS.PY

```
1 from django.shortcuts import render, redirect
2 from .forms import EmailForm
3 from django.contrib import messages
4 from dataentry.utils import send_email_notification
5 from django.conf import settings
6 from .models import Subscriber
7 from .tasks import send_email_task
```

```
12 def send_email(request):
13     # for email in subscribers:
14     #     to_email_addresses.append(email.email_address)
15     # OPTIMIZED OPTION: adds each email
16     to_email_addresses = [email.email_address for email in subscribers]
17     # check for email attachment
18     if email_form.attachment:
19         attachment = email_form.attachment.path
20     else:
21         attachment = None
22     # Handover instead the email-sending task to celery in tasks.py
23     send_email_task.delay(mail_subject, message_body,
24                           to_email_addresses, attachment)
25     # Without celery email function, call directly this function
26     # send_email_notification(
27     #     mail_subject, message_body, to_email_addresses, attachment)
28     messages.success(request, 'Email sent successfully.')
29     return redirect('send_email')
30     return
31 else:
32     email_form = EmailForm()
33     context = {
34         'email_form': email_form,
35     }
36     return render(request, 'emails/send_email.html', context)
```

3. Go to your CELERY BASH TERMINAL, press CTRL + C to stop it from running. Now execute the command to reload Celery **IMPORTANT REMINDER: When you run your celery terminal, it has error notifications for installed packages, CRISPY FORM AND CRISPY BOOTSTRAP5, so we installed it again..**



IT APPEARS THAT THE VIRTUAL ENVIRONMENT OF DJANGO-SERVER TERMINAL AND CELERY TERMINALS ARE DIFFERENT.

Plugging this confusion in CHATGPT, it shows this message:



Memory updated

The difference in installed packages between your two virtual environments (`django-server` and `celery`) is due to them being separate environments. Each virtual environment in Python is isolated, meaning it has its own installed packages independent of other environments.

Even though both environments are in the same project or use the same files and folders, they maintain their own package dependencies. This allows you to have different versions of the same package or even different sets of packages in different environments, depending on the needs of your project.

If you want both environments to have the same packages, you can do the following:

1. Freeze the requirements of one environment:

```
bash Copy code  
  
pip freeze > requirements.txt
```

2. Install the same requirements in the other environment:

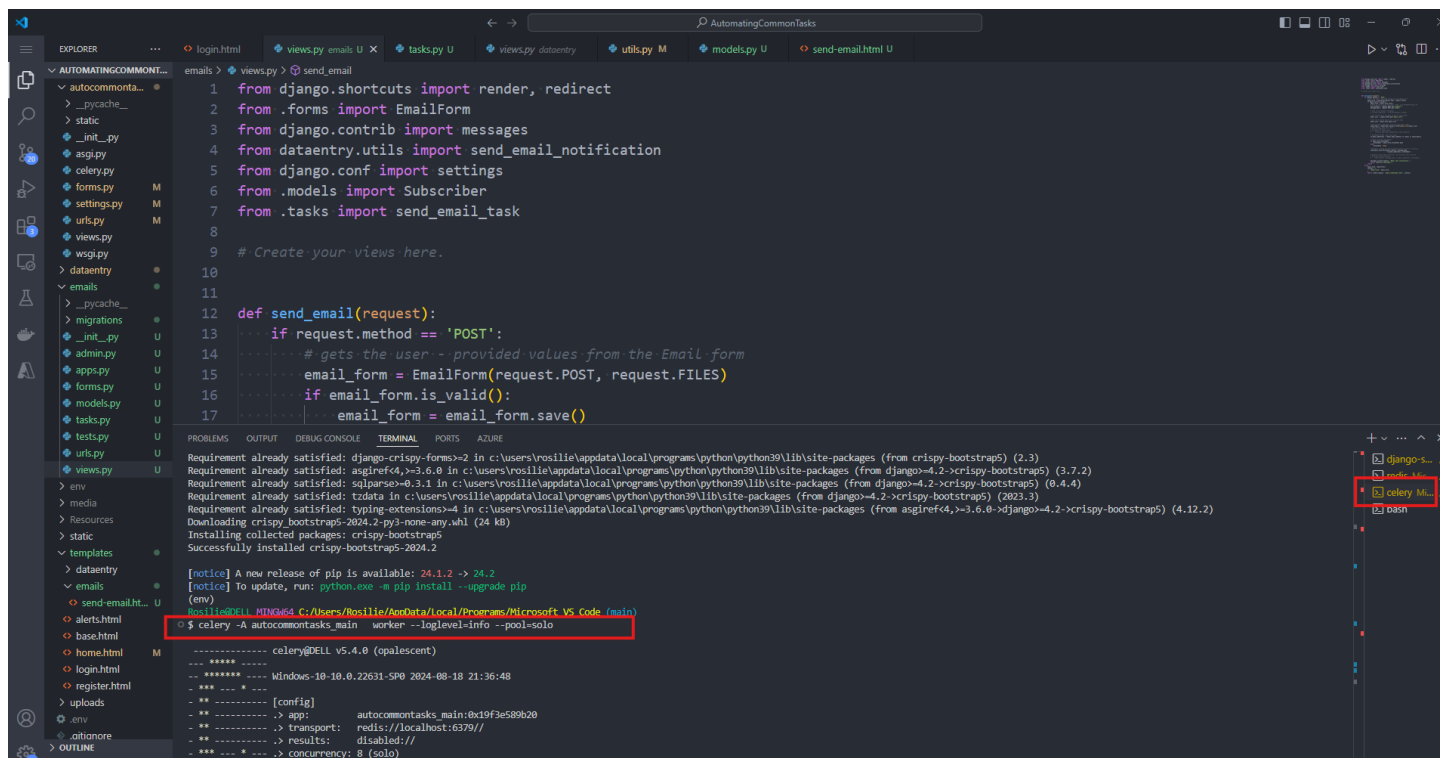
```
bash Copy code  
  
pip install -r requirements.txt
```

Alternatively, ensure you activate the correct environment based on the tasks you're performing in your project.



4 We run our celery command again:

```
$ celery -A autocommtasks_main worker --loglevel=info --pool=solo
```



We load our URL link: <http://127.0.0.1:8000/emails/send-email/>



Logged in as: commontask_admin [Logout](#)

Compose a Bulk Email

Email list*


Blog_subscribers ▼

Subject*

Celery Test

Body*

This is a test with Celery

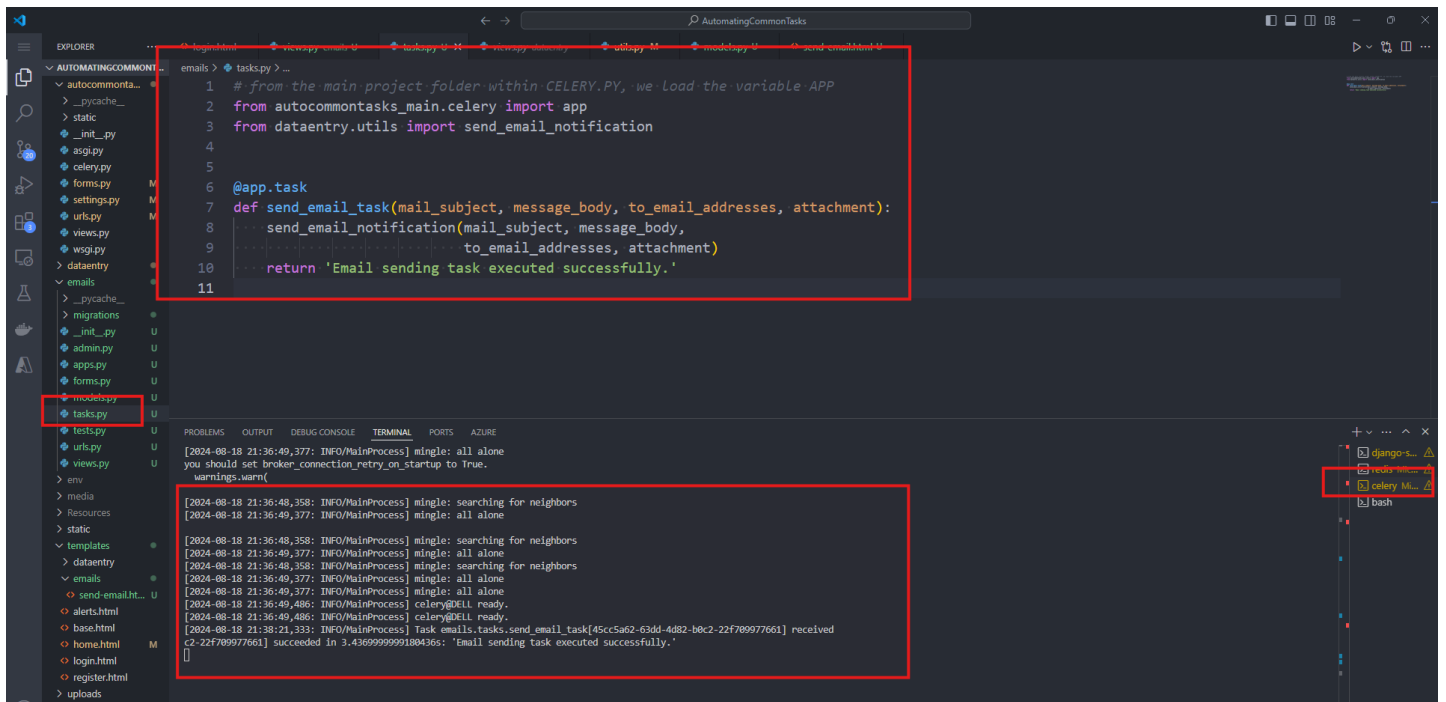


Attachment

Choose File pexels-digitalbuggu-171198.jpg

[Send](#)

Our Celery terminal shows this:



5.