

****

**Setting up a Fake SMTP server**:

To send emails we need an *SMTP* server.

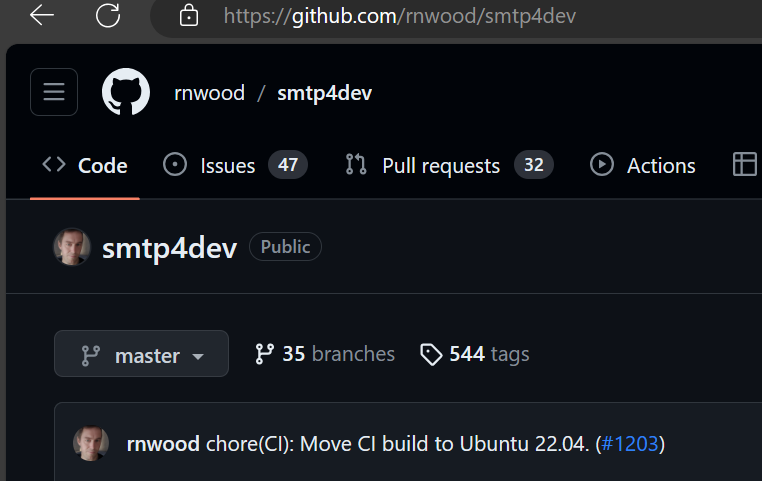


A SMTP server is *a software that knows how to send and receive emails*.

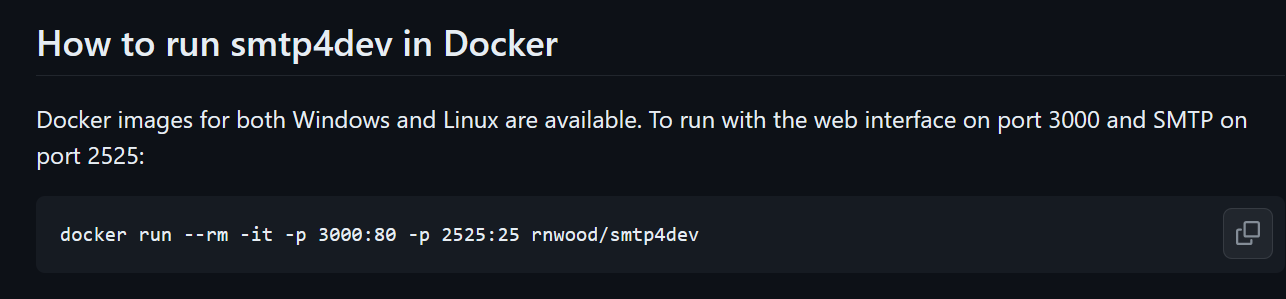
In the real world we need to use a real SMTP server and quite often it cost us money (*will talk later about it when we go to production*).



For now, as part of our development, we need to set up a fake SMTP server. Here we will use ***smtp4dev*** for this purpose.



Let’s look at the installation instructions (*using docker*),



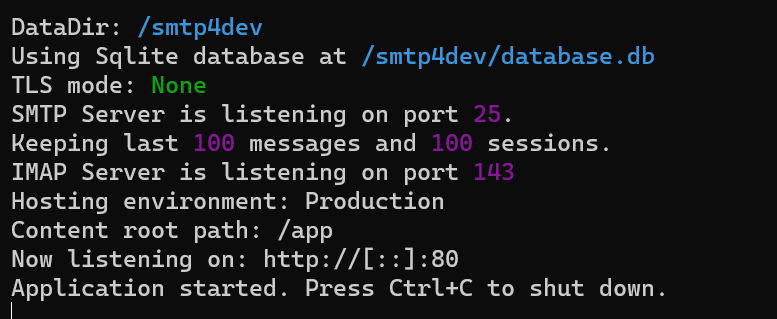
Install docker then run this command,

docker run --rm -it -p 3000:80 -p 2525:25 rnwood/smtp4dev

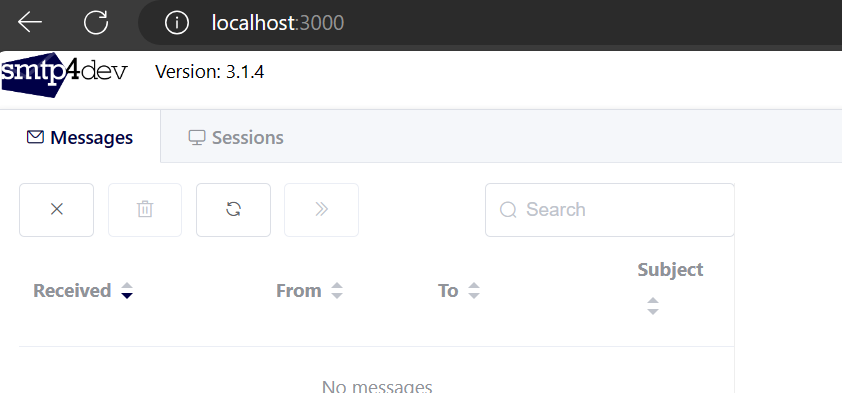


We don’t have this image ‘rnwood/smtp4dev:latest’ on our machine so Docker is going to download this image from *DockerHub* (*repository for docker images*).

Our SMTP server is up an running on our machine now,



Beautiful thing about this fake SMTP server is that it also gives us an administrative panel.

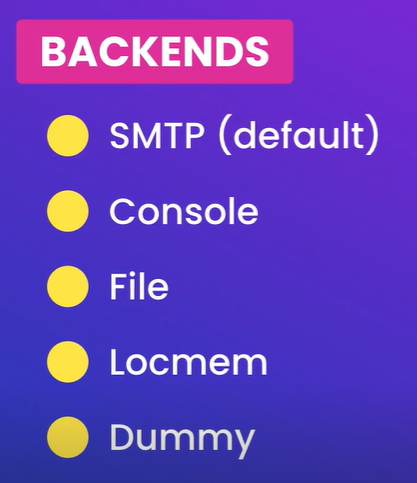


This is like our outlook / mail client. So as we send emails in our application, those emails will appear here.

**Configuring the Email Backend**:

Now we need to configure our email backend. An *email backend* is *essentially an engine responsible for sending emails*.

In Django we have a few different Email backends,



Console backend (*emails that we send will appear in the console or terminal window*)

File (*Emails will be written in a file*)

Locmem (*Writing emails to local memory*)

Let’s see how can we configure the backend:

In the settings module, write *EMAIL\_BACKEND* to ‘*django.core.mail.backends.smtp.EmailBackend*’

Note: In the *backends* we have a module called *smtp* and this module we have a class called *EmailBackend*. If we need to use console backend we just have to use name of the module like this…

‘*django.core.mail.backends.****console****.EmailBackend*’

OR

‘*django.core.mail.backends.****filebased****.EmailBackend*’

Here we will use the default one,

EMAIL\_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'

When using an SMTP server, we should also set a few other settings.

*EMAIL\_HOST* to the address of the SMTP server.

EMAIL\_HOST = 'localhost'

*EMAIL\_HOST\_USER* & *EMAIL\_HOST\_PASSWORD*(*since this fake smtp server does not have a username or password, so we can leave them empty*)

EMAIL\_HOST\_USER = ""

EMAIL\_HOST\_PASSWORD = ""

We should also set *EMAIL\_PORT*. *By default smtp servers run on port 25* but since this is a fake smtp server, it runs on port 2525.

EMAIL\_PORT = 2525

Optionally we can also set *DEFAULT\_FROM\_EMAIL*.

DEFAULT\_FROM\_EMAIL = 'from@lemonrobotics.com'

All settings in nutshell,

EMAIL\_BACKEND = "django.core.mail.backends.smtp.EmailBackend"

EMAIL\_HOST = "localhost"

EMAIL\_HOST\_USER = ""

EMAIL\_HOST\_PASSWORD = ""

EMAIL\_PORT = 2525

DEFAULT\_FROM\_EMAIL = 'from@lemonrobotics.com'

This is how can configure our email backend.

**Sending Emails**:

With everything in place, let’s see how can we send emails.

So let’s go to our playground app and open the views module.

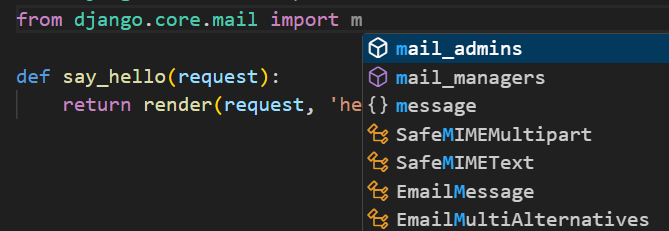
from django.shortcuts import render

def say\_hello(request):

    return render(request, 'hello.html', {'name': 'Mosh'})

Here before rendering this template, let’s send an email.

On the top from *django.core.mail* let’s import a couple of mail functions.



Here we have a couple of functions here.

For example, *mail\_admins* (*for sending an email to admins*).

Note: There is also *send\_mail* and *send\_mass\_mail*. The difference between these two is that *send\_mass\_mail* opens a single connection and sends all these emails then close the connection but *send\_mail* opens a new connection for each email.

So for performance reasons if you have a large number of emails or if you have more then one email, you should always use *send\_mass\_mail*.

Here we are going to import,

from django.core.mail import send\_mail, mail\_admins

To send an email, we call *send\_mail* function and give it a bunch of arguments.

from django.shortcuts import render

from django.core.mail import send\_mail, mail\_admins

def say\_hello(request):

    send\_mail('subject', 'message', 'info@lemonrobotics.com', ['bob@lemonrobotics.com']) 🡪 *Here*

    return render(request, 'hello.html', {'name': 'Mosh'})

First one is ‘*subject’* then ‘*message’* and third one is the FROM\_EMAIL with which we can overwrite the default we set in the settings. The fourth argument is the list of recipients.

Note: This function will throw an exception if an attacker tries to modify email headers and control to and from fields. (*Sometimes we come across fake emails that pretend to be from google or your bank*)

To avoid such attacks, we need to import a class called *BadHeaderError* then we should wrap our calls inside a try -catch block like this.

from django.shortcuts import render

from django.core.mail import send\_mail, mail\_admins, BadHeaderError 🡪 *Here*

def say\_hello(request):

    try:

        send\_mail("subject", "message", "info@lemonrobotics.com", ["bob@lemonrobotics.com"])

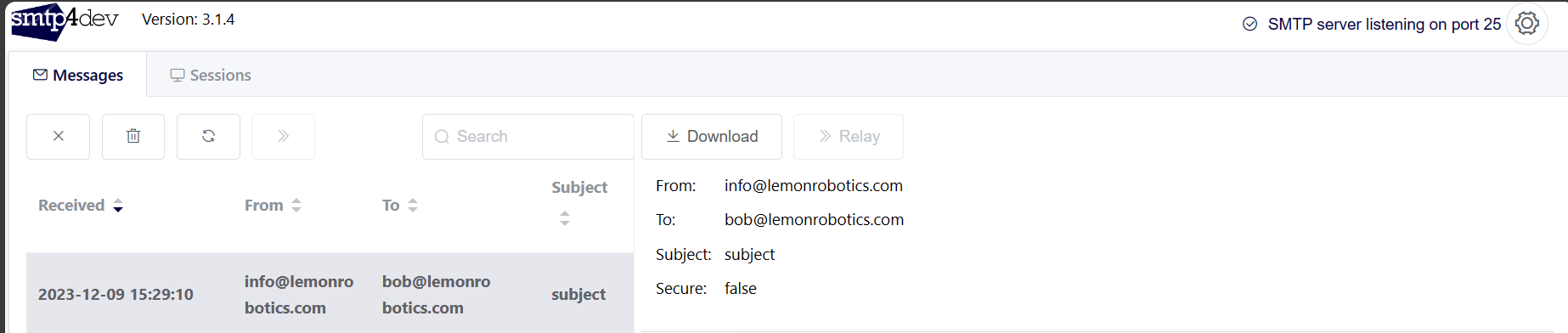
    except BadHeaderError:

        pass 🡪 *We can also provide a message here*

    return render(request, "hello.html", {"name": "Mosh"})

Go to *playground/hello/* endpoint and refresh it.

And now when we refresh our smtp dev admin panel at localhost:3000 we can see our email.



Here we can see the subject and body of the email.

Now let’s talk about sending emails to site admins, so change our email function to *mail\_admins*.

from django.shortcuts import render

from django.core.mail import send\_mail, mail\_admins, BadHeaderError

def say\_hello(request):

    try:

        mail\_admins("subject", "message", html\_message="message") 🡪 *Here*

    except BadHeaderError:

        pass

    return render(request, "hello.html", {"name": "Mosh"})

The signature of this mail\_admins function is a little bit different. First we specify the subject then the message. We can also supply an *html\_message* (*also available in send\_mail function*).

In html\_message we can put HTML content but what we put in second argument is just plain text content.

Note: For this to work we need to configure our site admins. *This function is not going to go into our database and not going to look through user table and find the site admins*.

So we need to go to our settings module and define our site admins using a key called *ADMINS*.

ADMINS = [

    ()

]

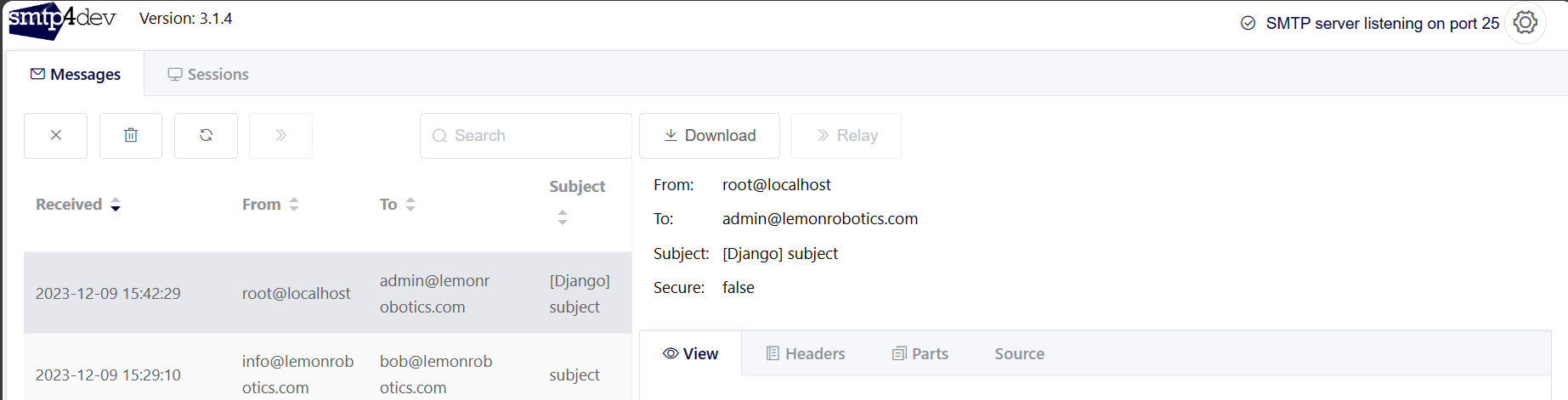
We set this ADMINS to a list of tuples. Each tuple should have two values (*a name and an email*).

ADMINS = [

    ('Himanshu', 'admin@lemonrobotics.com'),

]

Let’s test this as well, so refresh our playground/hello endpoint.



We can see the second email sent to [admin@lemonrobotics.com](mailto:admin@lemonrobotics.com)

Note: Mail clients that support HTML will show html content and other clients will display text content. So we should send both text and HTML emails.

**Attaching Files**:

Now what if we want to attach something in our email. For that we need to use our *EmailMessage* class.

So all these functions that we talked about in the previous lesson like send\_mail or mail\_admins, internally they use the *EmailMessage* class.

In this lesson instead of relying on these functions we are going to import *EmailMessage* class and create an EmailMessage object.

from django.core.mail import EmailMessage, BadHeaderError 🡪 *import class*

def say\_hello(request):

    try:

        EmailMessage () 🡪 *use it as an object*

Once again here we can set subject, message, from email, list of recipients… exactly like before.

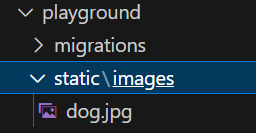
def say\_hello(request):

    try:

        message = EmailMessage ('subject', 'message', 'info@lemonrobotics.com',['himanshu@lemonrobotics.com'])

This *message* object has a method called *attach\_file* and we can *call this method by giving it a path relative to our project directory*.

Let’s create a new folder in our playground app and call it static—images—dog.jpg



Now we give it path like this playground/static/images/dog.jpg and to send this we use message.***send***()

def say\_hello(request):

    try:

        message = EmailMessage(

            "subject",

            "message",

            "info@lemonrobotics.com",

            ["himanshu@lemonrobotics.com"],

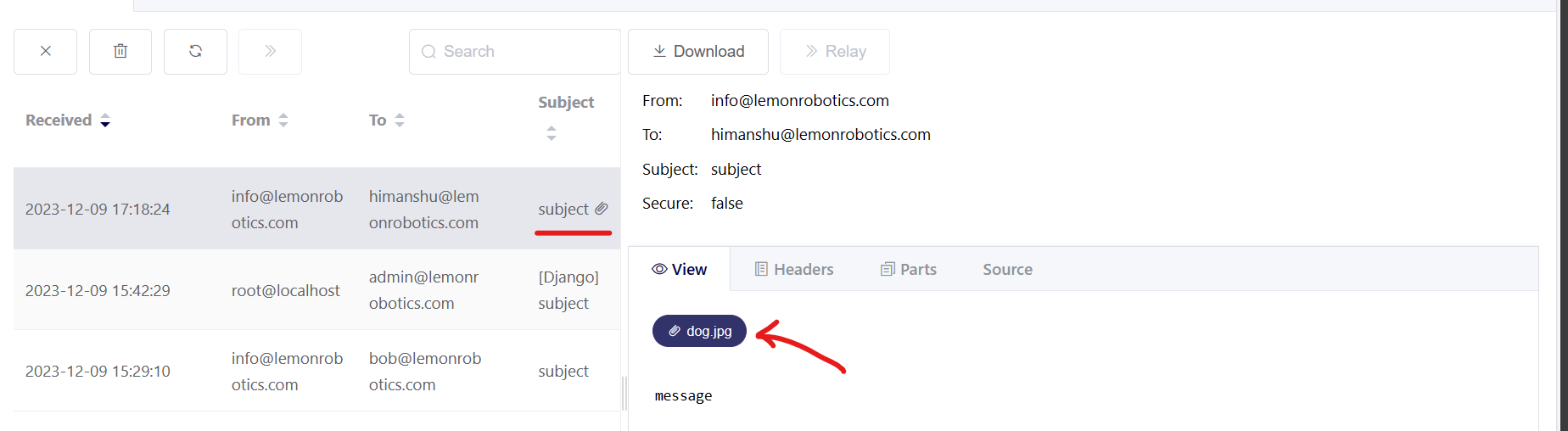
        )

        message.attach\_file("playground/static/images/dog.jpg")

        message.send()

To test this, refresh the endpoint.

And here we have a new email with our image as attachment.



If you want to have more control over your emails like if we want to attach something or need to use BCC or CC features, then you have to use *EmailMessage* class.

**Sending Templated Emails**:

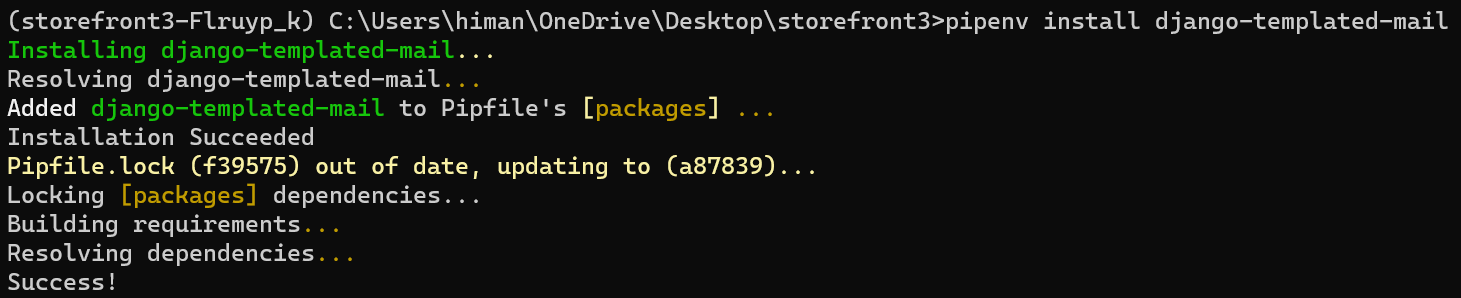
In Django we can send message using either *EmailMessage* class or one of the shortcut functions. Now in both examples we have seen so far, our email message was just one word.

In reality that’s never the case. Quite often *our emails have long text and we also want to dynamically insert data into the email body*. To do that we need a library called *django templated mail*.

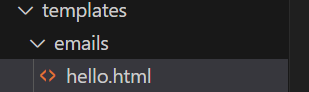
**“**I*t’s basically a thin wrapper around Django’s mailing functionality but allows us to store our email messages in template files****”***.

First install it using,

pipenv install django-templated-mail



Now we need to create an email template, so back to our *playground* app -- *template* folder; We can create a new folder called *emails*.



Here we add a new template called *hello*.html.

In this template file we can provide subject of our email as well its text and HTML body.

First we define a block using this syntax and call this block ‘subject’ like this,

{% block subject %}

Whenever we open a block, we should also close it using *endblock*,

{% block subject %}{% endblock %}

In between these blocks we can type subject of our email,

{% block subject %}This is a long subject{% endblock %}

Obviously, We don’t want to hard code these long strings in our code.

Similarly we have other blocks for creating the text and HTML body.

*For text body*, (*its optional*)

{% block text\_body%}

{% endblock %}

*For html content*,

{% block html\_body %}

{% endblock %}

In the HTML body, we can write our html tags.

{% block subject %}This is a long subject{% endblock %}

{% block html\_body %}

<h1>Hello</h1>

{% endblock %}

To dynamically insert content we use the same syntax as we have in django templates.

{% block html\_body %}

<h1>Hello</h1>

My Name is {{name:}}

{% endblock %}

Here we pass a variable called name and we will set its value from our code.

For that let’s go to our view and instead of this *EmailMessage* class, we are going to use another class defined in django-templated-mail library we just installed.

Here we import *BaseEmailMessage* class,

from templated\_mail.mail import BaseEmailMessage

This class EmailMessage class in Django, so it has all these features we have learned (*like attach\_file, send and so on…*), but the constructor is a little bit different so it does not have the same arguments.

def say\_hello(request):

    try:

        message = BaseEmailMessage(

        )

We start from the same by creating a message object from BaseEmailMessage class then we set a couple of keyword arguments.

First one is *template\_name* (*we set it to hello.html*) and second one is *context*(*context object that we use to pass data to our templates*).

def say\_hello(request):

    try:

        message = BaseEmailMessage(

            template\_name= emails/hello.html',

            context={'name':'Himanshu'} 🡪 *our key – value pair as expected.*

        )

So we have our message, then we send it.

def say\_hello(request):

    try:

        message = BaseEmailMessage(

            template\_name="emails/hello.html", context={"name": "Himanshu"}

        )

        message.send() 🡪 *send the message…*

    except BadHeaderError:

        pass

    return render(request, "hello.html", {"name": "Mosh"})

On our playground/hello page,



So the *send* method here is a little bit different, so here we should pass a list of recipients.

from django.shortcuts import render

from django.core.mail import EmailMessage, BadHeaderError

from templated\_mail.mail import BaseEmailMessage

def say\_hello(request):

    try:

        message = BaseEmailMessage(

            template\_name="emails/hello.html", context={"name": "Himanshu"}

        )

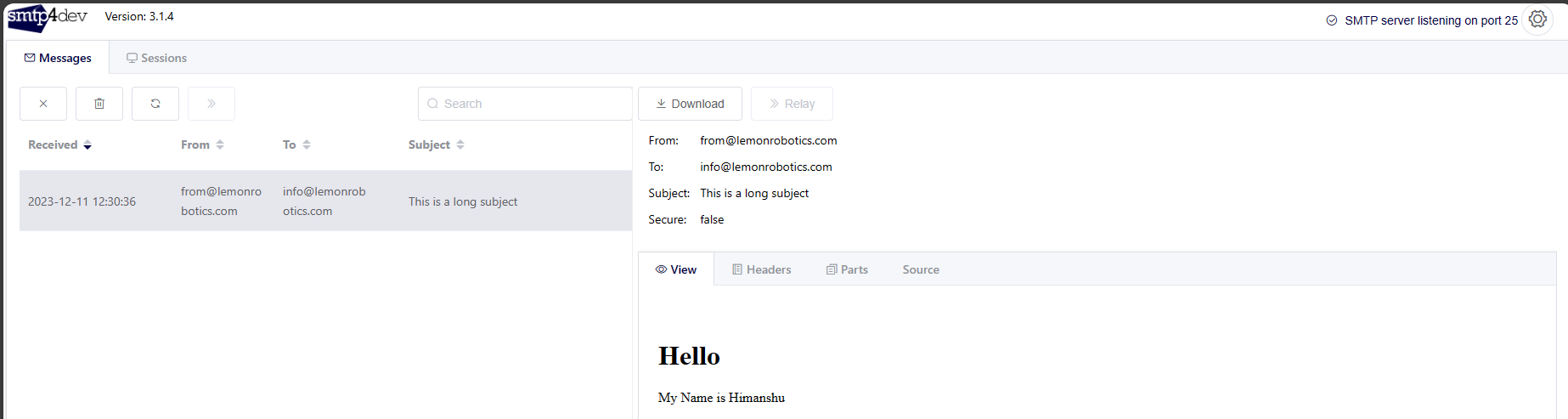
        message.send(["info@lemonrobotics.com"]) 🡪 *Here*

    except BadHeaderError:

        pass

    return render(request, "hello.html", {"name": "Mosh"})

Now we refresh the page and check our smtp4dev server,



In the email body we have our HTML content dynamically rendered.

**SMTP servers options**:

