

Topic: Custom Management Command Part 6: Adding Comments or Messages

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



Django has built-in MESSAGES FRAMEWORK that we can use to show ERROR OR SUCCESS MESSAGES. [See Django's documentation.](#)

SETTINGS.PY has already the configurations needed to support the messages.

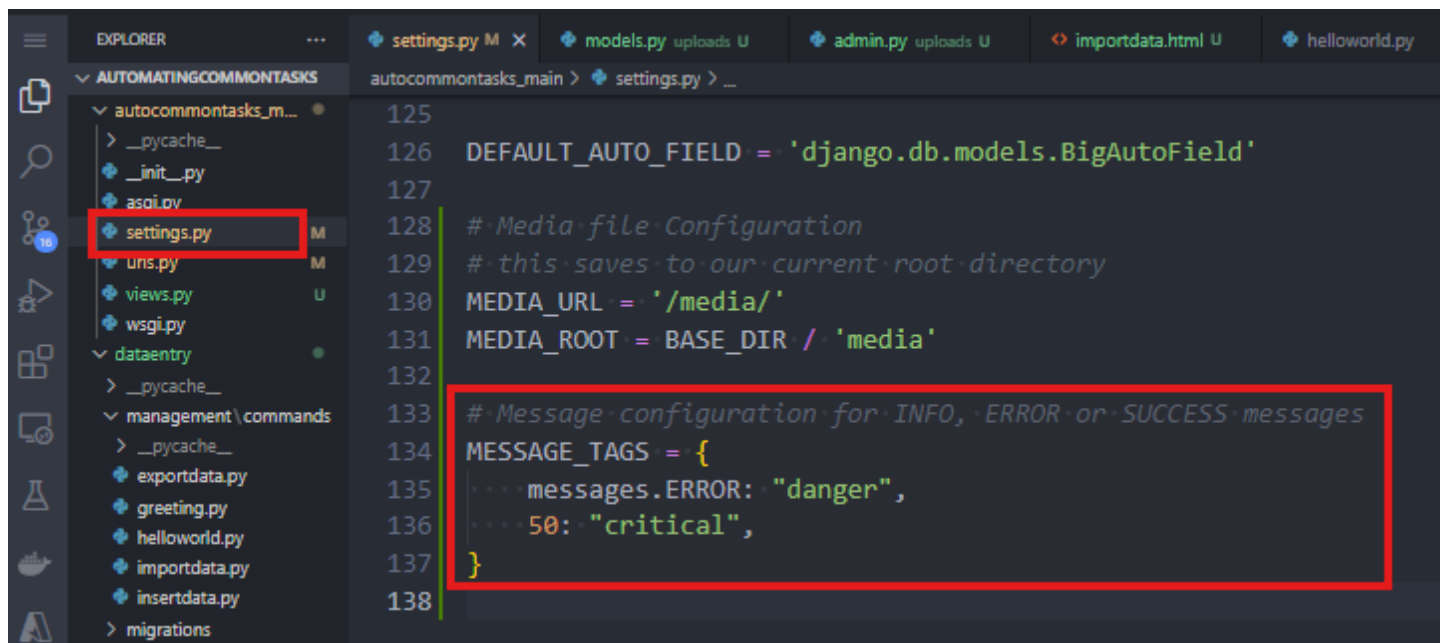
1. So, we just need to include the following in our SETTINGS.PY.

From Django documentation, we copy this and modify based on our needs.

```
from django.contrib.messages import constants as messages

MESSAGE_TAGS = {
    messages.INFO: "",
    50: "critical",
}
```

in our SETTINGS.PY, we modify it to where 'danger' is a bootstrap name.



2. Then in the TEMPLATES FOLDER, create a new HTML, ALERT.HTML and design as:

from our documentation:

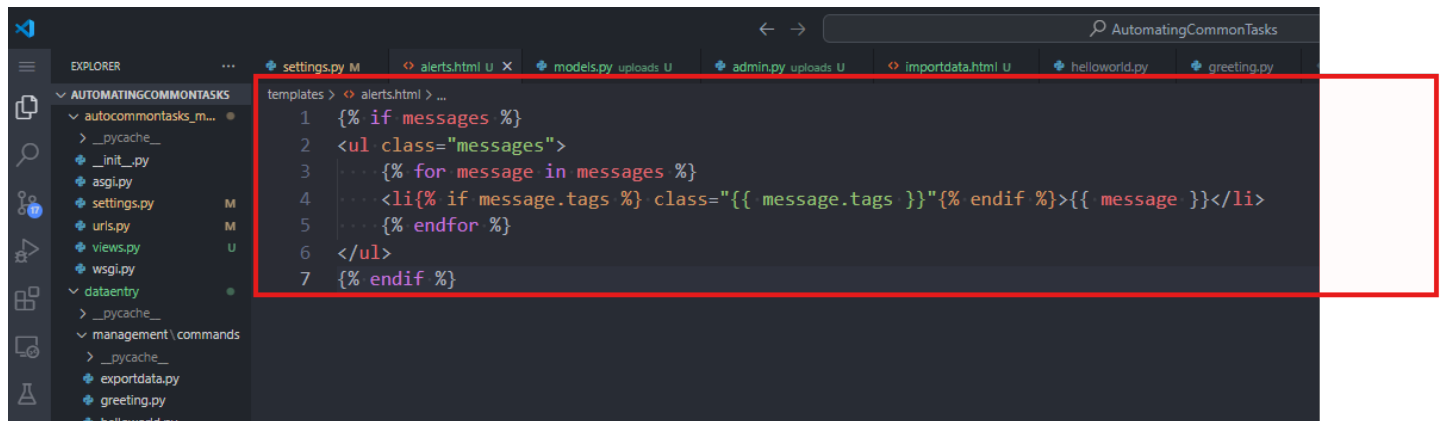
Displaying messages ¶

`get_messages(request)[source]` ¶

In your template, use something like:

```
{% if messages %}
<ul class="messages">
    {% for message in messages %}
    <li{% if message.tags %} class="{{ message.tags }}" {%
endif %}>{{ message }}</li>
    {% endfor %}
</ul>
{% endif %}
```

So, in our ALERTS.HTML, we add the same code:



```
templates > alerts.html > ...
1  {% if messages %}
2  <ul class="messages">
3      {% for message in messages %}
4      <li{% if message.tags %} class="{{ message.tags }}" {%endif %}>{{ message }}</li>
5      {% endfor %}
6  </ul>
7  {% endif %}
```

3. Now you can call this ALERTS.HTML in ANY HTML that will require messages or comments.

So, in DATAENTRY\IMPORTDATA.HTML, we add

```

<div class="container">
    <h3 class="text-center">Import Data From CSV File to Database Tables</h3>
    <form action="{% url 'import_data' %}" method="POST" enctype="multipart/form-d
    style="max-width: 600px;margin:auto;padding-top: 50px;">
        {% csrf_token %}
        <div class="form-group">
            <label for="file_path">Upload CSV File</label>
            <input type="file" name="file_path" class="form-control" required>
        </div>

        <div class="form-group">
            <label for="model_name">Select Database Table</label>
            <select name="model_name" class="form-control" required>
                <option value="" disabled selected>Select</option>
                {% for model in custom_models %}
                <option value="{{model}}">{{model}}</option>
                {% endfor %}
            </select>
        </div>

        <input type="submit" value="Import Data" class="btn btn-primary">

        <!-- insert our custom alert messages -->
        {% include 'alerts.html' %}

    </form>

```

4. To add our custom messages, we go to DATAENTRY\VIEWS.PY

```

AutomatingCommonTasks

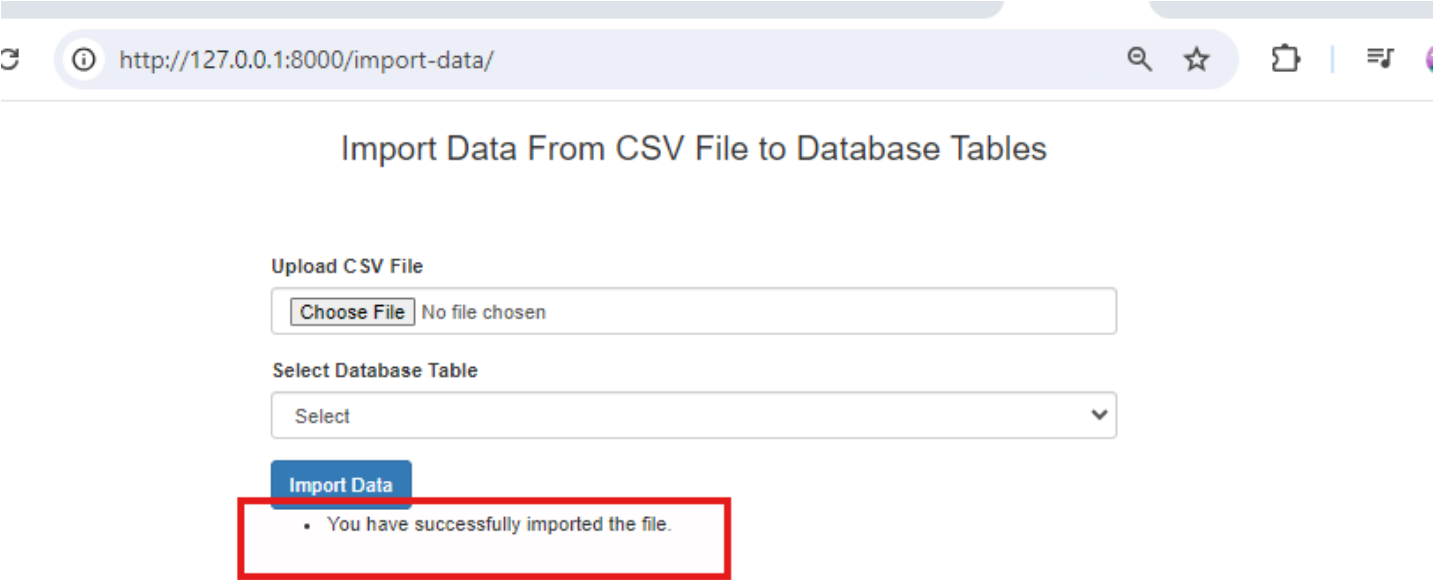
EXPLORER
AUTOMATINGCOMMONTASKS
  autocommontasks_m...
    __pycache__
    __init__.py
    asgi.py
    settings.py M
    urls.py M
    views.py U
    wsgi.py
  dataentry
    __pycache__
    management\commands
      __pycache__
      exportdata.py
      greeting.py
      helloworld.py
      importdata.py
      insertdata.py
      migrations
      __init__.py
      admin.py
      apps.py
      models.py
      tests.py
      urls.py U
      utils.py U
      views.py M

dataentry > views.py > ...
1 from django.shortcuts import render, redirect
2 from .utils import get_all_custom_models
3 from uploads.models import Upload
4 from django.conf import settings
5 from django.core.management import call_command
6 from django.contrib import messages
7
8
9 def import_data(request):
10     if request.method == 'POST':
11         # gets the user-submitted file using its path
12         file_path = request.FILES.get('file_path')
13         # gets the user-submitted MODEL name
14         model_name = request.POST.get('model_name')
15
16         # store this file inside the Upload model
17         upload = Upload.objects.create(file=file_path, model_name=model_name)
18
19         # construct the full path of the file with the file_path for import
20         relative_path = str(upload.file.url)
21         # gets the directory of our root directory
22         base_url = str(settings.BASE_DIR)

```

```
def import_data(request):
    ...
    relative_path = str(upload.file.url)
    ...
    # gets the directory of our root directory
    base_url = str(settings.BASE_DIR)
    ...
    file_path = base_url + relative_path # absolute path of the file
    ...
    # trigger the custom-management import data command
    try:
        call_command('importdata', file_path, model_name)
        ...
        # passes this custom-made message to our web page
        messages.success(
            request, 'You have successfully imported the file.'
        )
    except Exception as e:
        ...
        # raise e
        ...
        # passes this error type to our web page
        messages.error(request, str(e))
    ...
    return redirect('import_data')
else:
    ...
    # calls our dataentry\utils.py to extract only user-created models
    custom_models = get_all_custom_models()
    ...
    # print(custom_models)
    context = {
        'custom_models': custom_models,
    }
    return render(request, 'dataentry/importdata.html', context)
```

5. Now when you add a CSV file again, you should be able to see an error message.



6. Now to make the style of the message that follows the bootstrap style, go to this [BOOTSTRAP ALERT documentation](#).

h...

ing started

it

ent

onents

:

crumb

is

group

sel

se

owns

group

atron

oup

ir

ition

ers

ss

py

ss

es

d

ition

t

Examples

Alerts are available for any length of text, as well as an optional dismiss button. For proper styling, use one of the eight **required** contextual classes (e.g., `.alert-success`). For inline dismissal, use the [alerts jQuery plugin](#).

This is a primary alert—check it out!

This is a secondary alert—check it out!

This is a success alert—check it out!

This is a danger alert—check it out!

This is a warning alert—check it out!

This is a info alert—check it out!

This is a light alert—check it out!

This is a dark alert—check it out!

```
<div class="alert alert-primary" role="alert">
  This is a primary alert-check it out!
</div>
<div class="alert alert-secondary" role="alert">
  This is a secondary alert-check it out!
</div>
<div class="alert alert-success" role="alert">
  This is a success alert-check it out!
</div>
<div class="alert alert-danger" role="alert">
  This is a danger alert-check it out!
</div>
<div class="alert alert-warning" role="alert">
  This is a warning alert-check it out!
</div>
```

ALERTS.HTML BEFORE:

```

1  {% if messages %}
2  <ul class="messages">
3    {% for message in messages %}
4    <li{% if message.tags %} class="{% message.tags %}"{% endif %}>{{ message }}</li>
5    {% endfor %}
6  </ul>
7  {% endif %}

```

ALERTS.HTML AFTER:

```
templates > > alerts.html > ...
1  {% if messages %}
2  <div class="messages">
3  ...{% for message in messages %}
4  ...<div
5  ...  {% if message.tags %} class=" alert alert-{{ message.tags }}" {% endif %}>{{ message }}
6  ...</div>
7  ...{% endfor %}
8  </div>
9  {% endif %}
```

7. Run the server again and check the message shown.

Saving Student.CSV to Student Model.

http://127.0.0.1:8000/import-data/

Import Data From CSV File to Database Tables

Upload CSV File

Choose File

No file chosen

Select Database Table

Select

Import Data

You have successfully imported the file.

Saving Student.CSV to Customer Model

http://127.0.0.1:8000/import-data/

Import Data From CSV File to Database Tables

Upload CSV File

Choose File

No file chosen

Select Database Table

Select

Import Data

Customer() got unexpected keyword arguments: 'roll_no', 'name', 'age'

Or a different file to CUSTOMER model.

8. To modify the error message into something user-friendly especially the user has used a different file with a different model, we update our IMPORTDATA.PY
To get the fields of our model, in our IMPORTDATA.PY, we add this:

```
# compare CSV header with the model's field names and throw appropriate message
# get all the field names of the model that we found
model_fields = [field.name for field in target_model._meta.fields]
print(model_fields)
```

So, when we upload our STUDENT.CSV FILE, we see this in our terminal:

```
[11/Aug/2024 15:41:11] "GET /import-data/ HTTP/1.1" 200 1907
['id', 'roll_no', 'name', 'age']
Data imported from the CSV file successfully.
```

CUSTOMER MODEL:

```
Django version 4.2.14, using settings 'autocommontasks_main.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

['id', 'customer_name', 'country']
[11/Aug/2024 16:07:29] "POST /import-data/ HTTP/1.1" 302 0
[11/Aug/2024 16:07:29] "GET /import-data/ HTTP/1.1" 200 2251
```

So, when we import a different file like EMPLOYEE.CSV into CUSTOMER Model, we get this user- friendly message:

http://127.0.0.1:8000/import-data/

Import Data From CSV File to Database Tables

Upload CSV File

Choose File

No file chosen

Select Database Table

Select

Import Data

CSV file does not match with the Customer table fields

Instead of this:

Import Data From CSV File to Database Tables

Upload CSV File

No file chosen

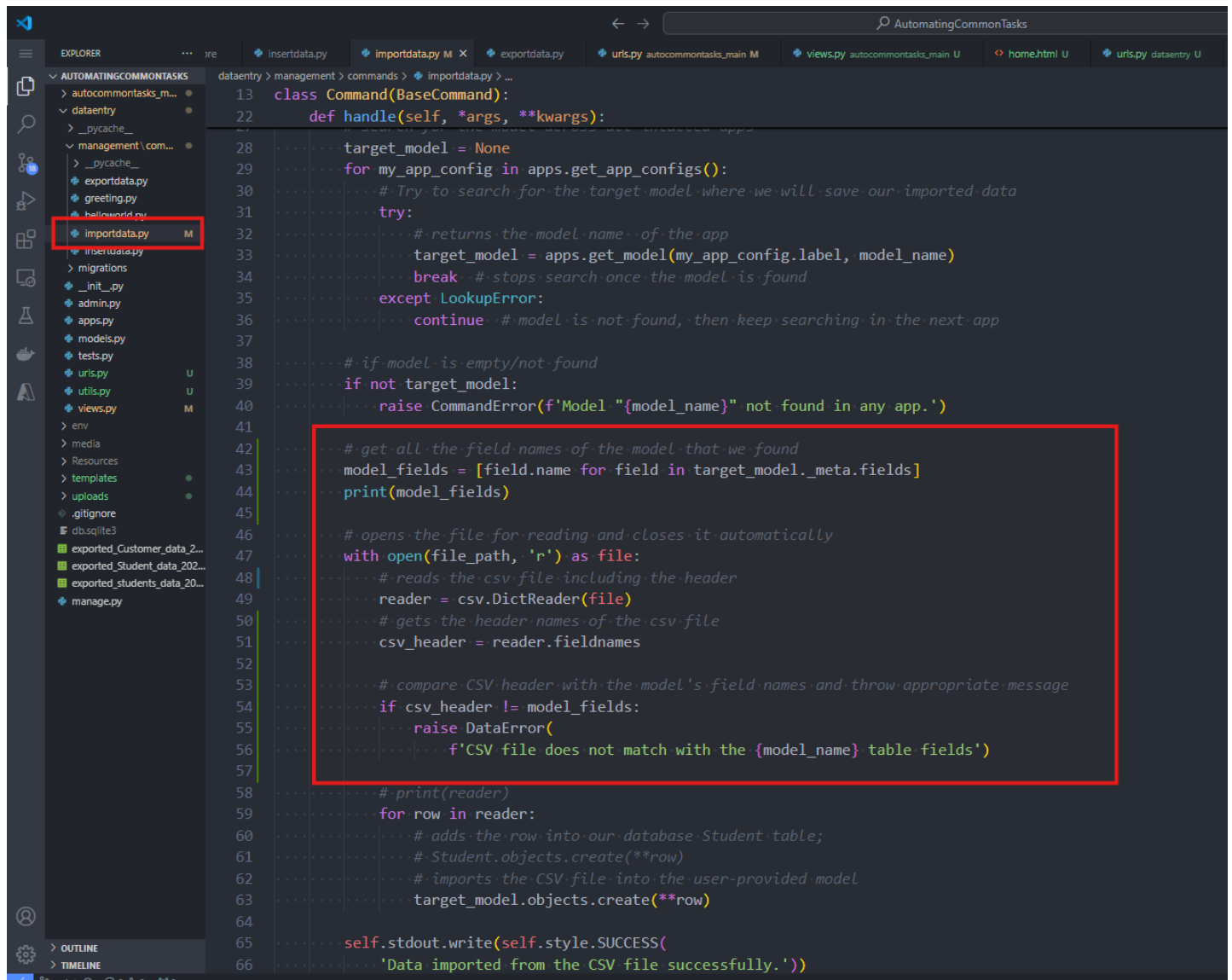
Select Database Table

Select

Import Data

Customer() got unexpected keyword arguments: 'employee_id', 'employee_name', 'designation', 'salary', 'retirement', 'other_benefits', 'total_benefits', 'total_compensation'

9. Your IMPORTDATA.PY should be:



```
13 class Command(BaseCommand):
22     def handle(self, *args, **kwargs):
23         # Search for the model where we will save our imported data
24         target_model = None
25         for my_app_config in apps.get_app_configs():
26             # Try to search for the target model where we will save our imported data
27             try:
28                 # returns the model name of the app
29                 target_model = apps.get_model(my_app_config.label, model_name)
30                 break # stops search once the model is found
31             except LookupError:
32                 continue # model is not found, then keep searching in the next app
33         # if model is empty/not found
34         if not target_model:
35             raise CommandError(f'Model "{model_name}" not found in any app.')
36         # get all the field names of the model that we found
37         model_fields = [field.name for field in target_model._meta.fields]
38         print(model_fields)
39         # opens the file for reading and closes it automatically
40         with open(file_path, 'r') as file:
41             # reads the csv file including the header
42             reader = csv.DictReader(file)
43             # gets the header names of the csv file
44             csv_header = reader.fieldnames
45             # compare CSV header with the model's field names and throw appropriate message
46             if csv_header != model_fields:
47                 raise DataError(
48                     f'CSV file does not match with the {model_name} table fields')
49             # print(reader)
50             for row in reader:
51                 # adds the row into our database Student table;
52                 # Student.objects.create(**row)
53                 # imports the CSV file into the user-provided model
54                 target_model.objects.create(**row)
55         self.stdout.write(self.style.SUCCESS(
56             'Data imported from the CSV file successfully.'))
```

10. But this will give us an error message since we have the the field 'ID' in our table but not in our CSV file, so we update our IMPORTDATA.PY as


```

...# get all the field names of the model that we found EXCEPT THE PK or ID
model_fields = [field.name for field in target_model._meta.fields if field.name !=
                 'id']
print(model_fields)

...# opens the file for reading and closes it automatically
with open(file_path, 'r') as file:
    ...# reads the csv file including the header
    reader = csv.DictReader(file)
    ...# gets the header names of the csv file
    csv_header = reader.fieldnames

    ...# compare CSV header with the model's field names and throw appropriate message
    if csv_header != model_fields:
        raise DataError(
            f'CSV file does not match with the {model_name} table fields')

```

11. IF WE WANT TO HIDE THE ERROR MESSAGES RIGHT AWAY AFTER IT WAS DISPLAYED, we create a JAVASCRIPT. But we need to update our SETTINGS.PY STATIC FILES.

FROM:

```

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/4.2/howto/static-files/

STATIC_URL = 'static/'

```

TO:

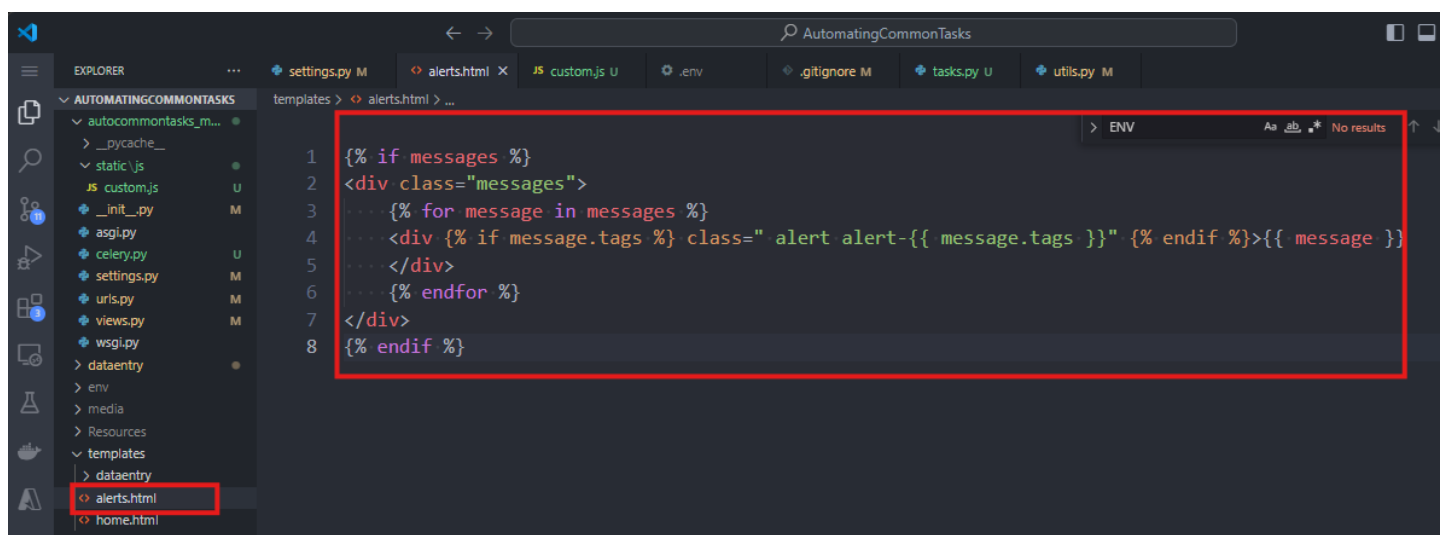
```

STATIC_URL = 'static/'
STATICFILES_DIRS = ['autocommontasks_main/static'] # this is the source folder
# this is the destination folder after COLLECTSTATIC command
STATIC_ROOT = BASE_DIR / 'static'

```

12. Create an ID FOR YOUR ALERTS.HTML

FROM:

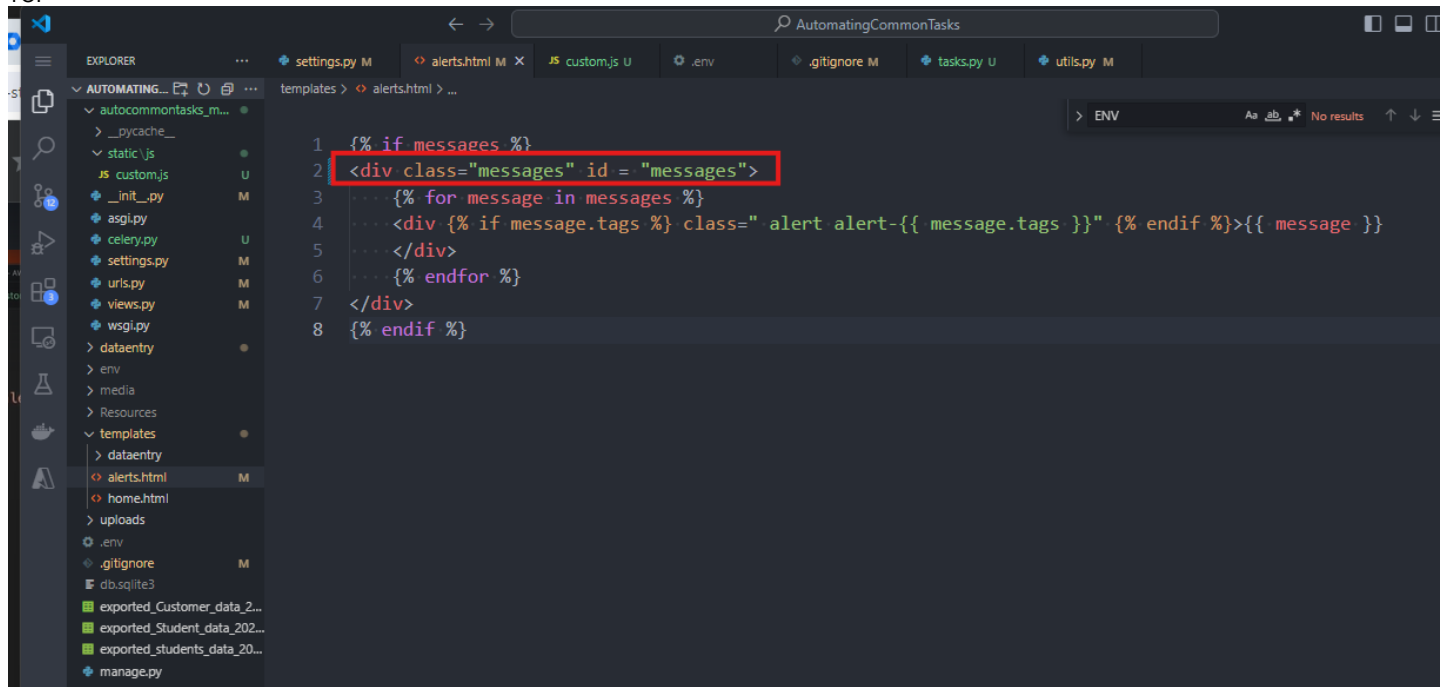


```

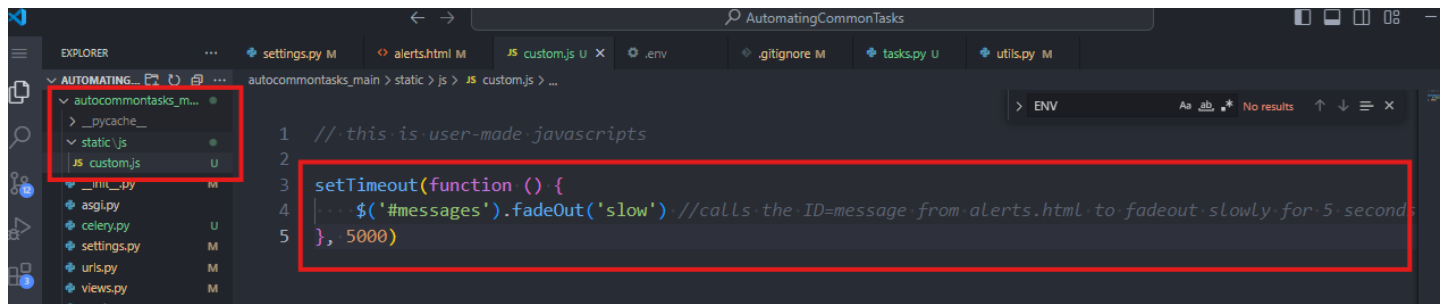
1 {% if messages %}
2 <div class="messages">
3     {% for message in messages %}
4     <div {% if message.tags %} class=" alert alert-{{ message.tags }}" {% endif %}>{{ message }}
5     </div>
6     {% endfor %}
7 </div>
8 {% endif %}

```

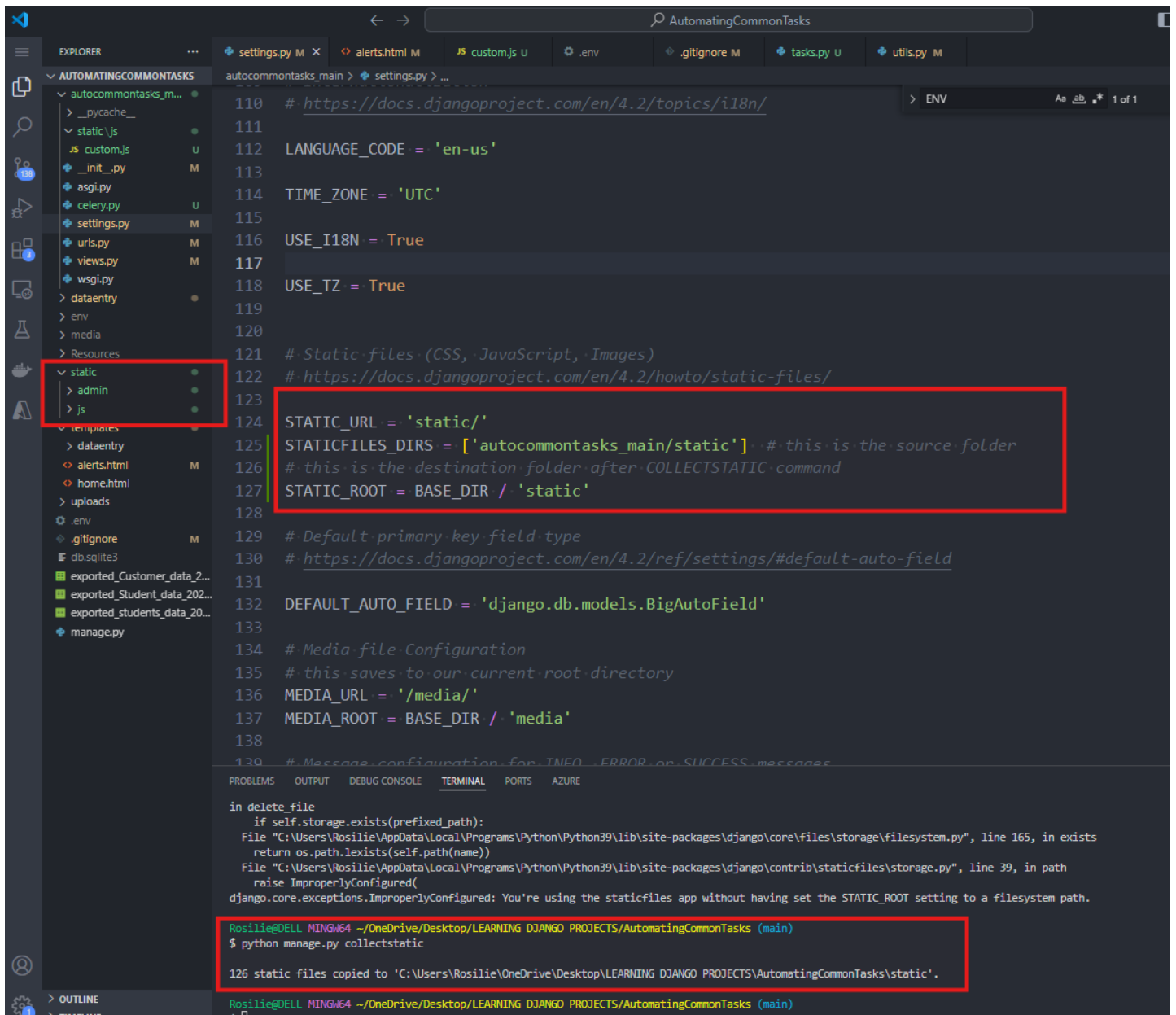
TO:



13. In our main project, we create a new folder and JS file, so in AUTOCOMMONTASKS_MAIN\STATIC\JS\CUSTOM.JS



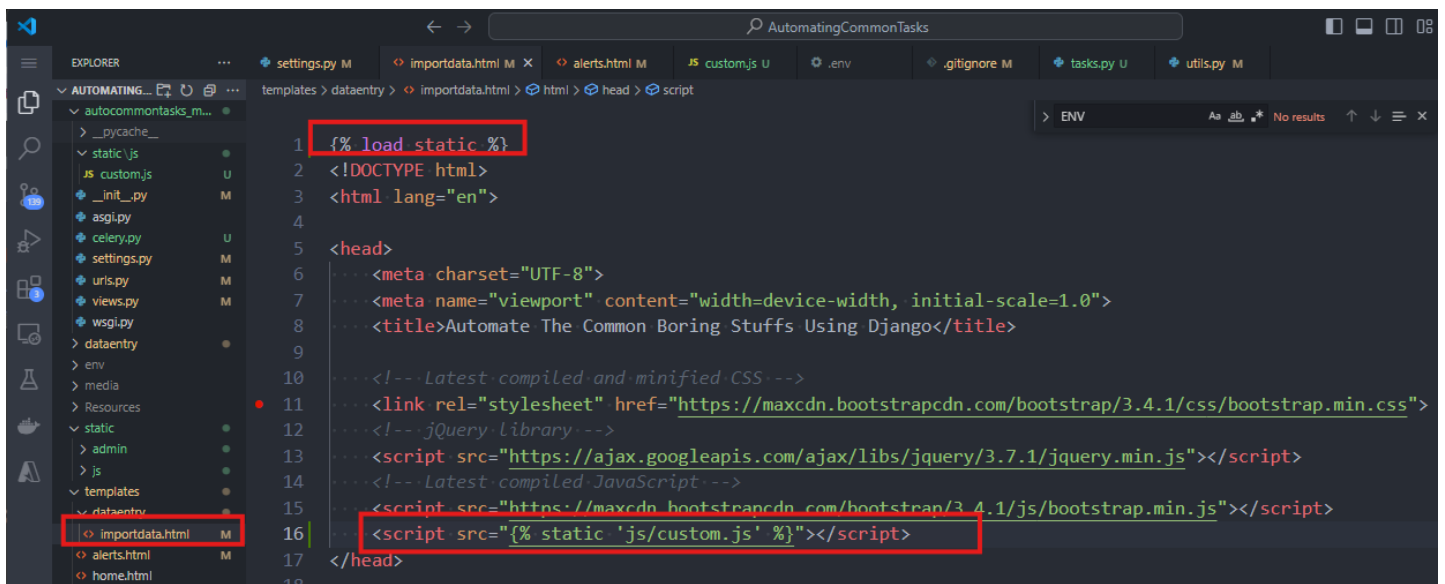
14. Since we have our own static files like CUSTOM.JS, we need to run COLLECTSTATIC COMMAND in our DJANGO-SERVER bash terminal. This will create a new folder STATIC in the root directory.




```
Rosilie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
$ python manage.py collectstatic

126 static files copied to 'C:\Users\Rosilie\OneDrive\Desktop\LEARNING DJANGO PROJECTS\AutomatingCommonTasks\static'.
```

15. Include this CUSTOM.JS JAVASCRIPT FILE in our DATAENTRY.HTML



16. Run the Django server again for testing. After 5 seconds, your alert message should disapper.

 http://127.0.0.1:8000/import-data/

Import Data From CSV File to Database Tables

Upload CSV File

Choose File

No file chosen

Select Database Table

Select▼

Import Data

Push all changes to Github.