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](#).

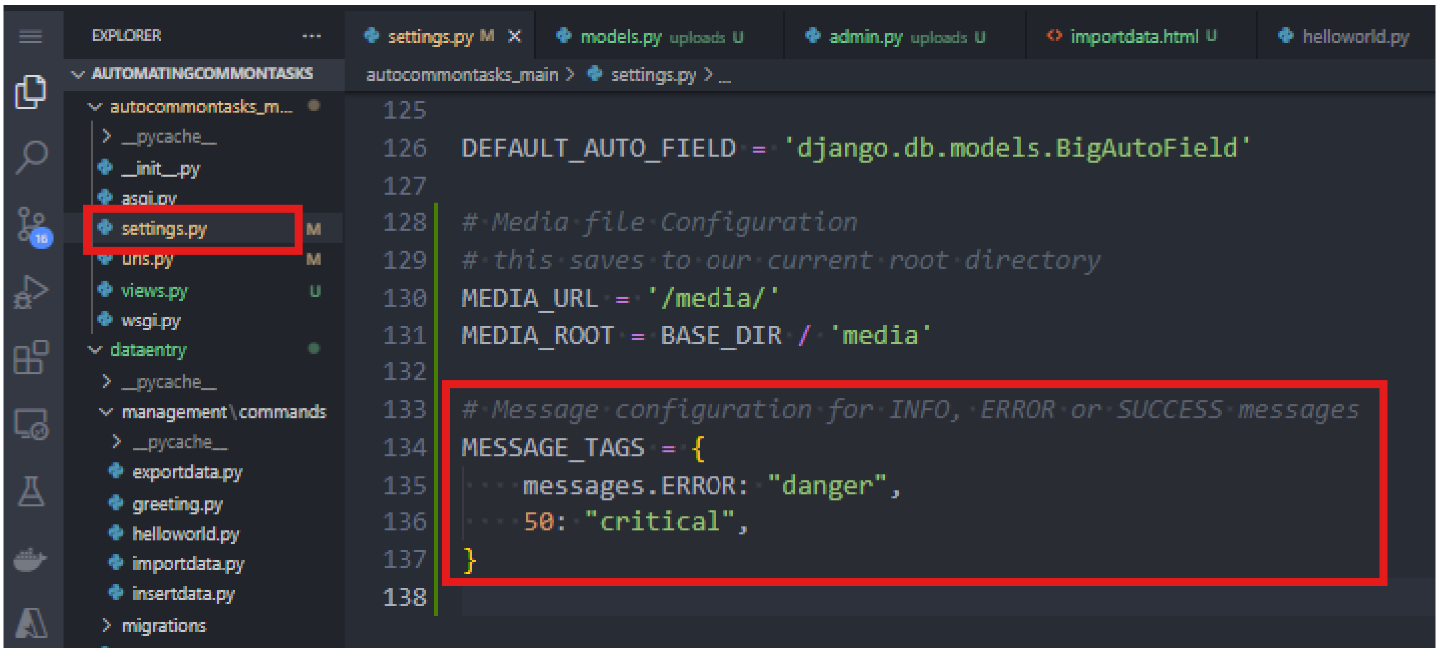
SETTINGS.PY has already the configurations needed to support the messages.

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",
}
```

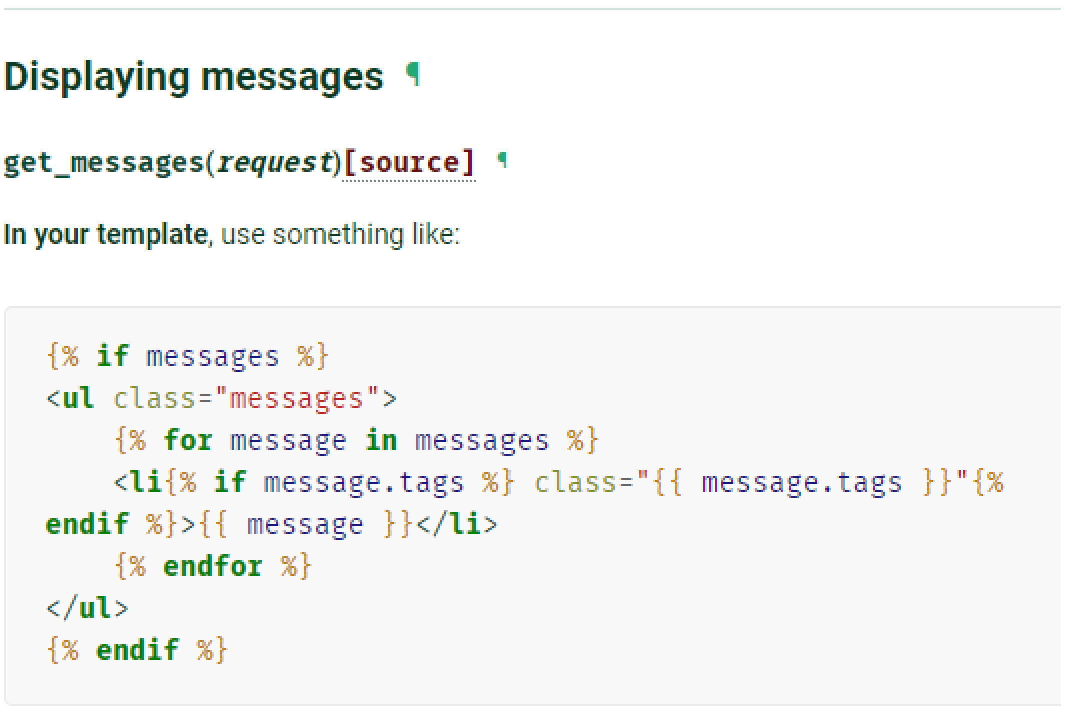
in our SETTINGS.PY, we modify it to where 'danger' is a bootstrap name.



```
DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

# Media file Configuration
# this saves to our current root directory
MEDIA_URL = '/media/'
MEDIA_ROOT = BASE_DIR / 'media'

# Message configuration for INFO, ERROR or SUCCESS messages
MESSAGE_TAGS = {
    messages.ERROR: "danger",
    50: "critical",
}
```

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

from our documentation:



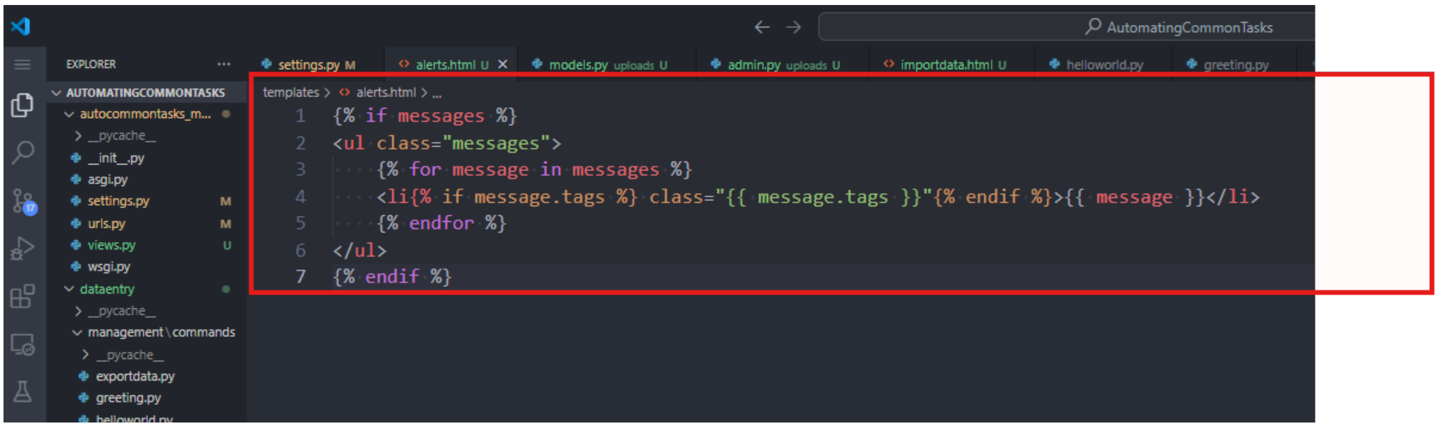
Displaying messages ¶

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 %}
```

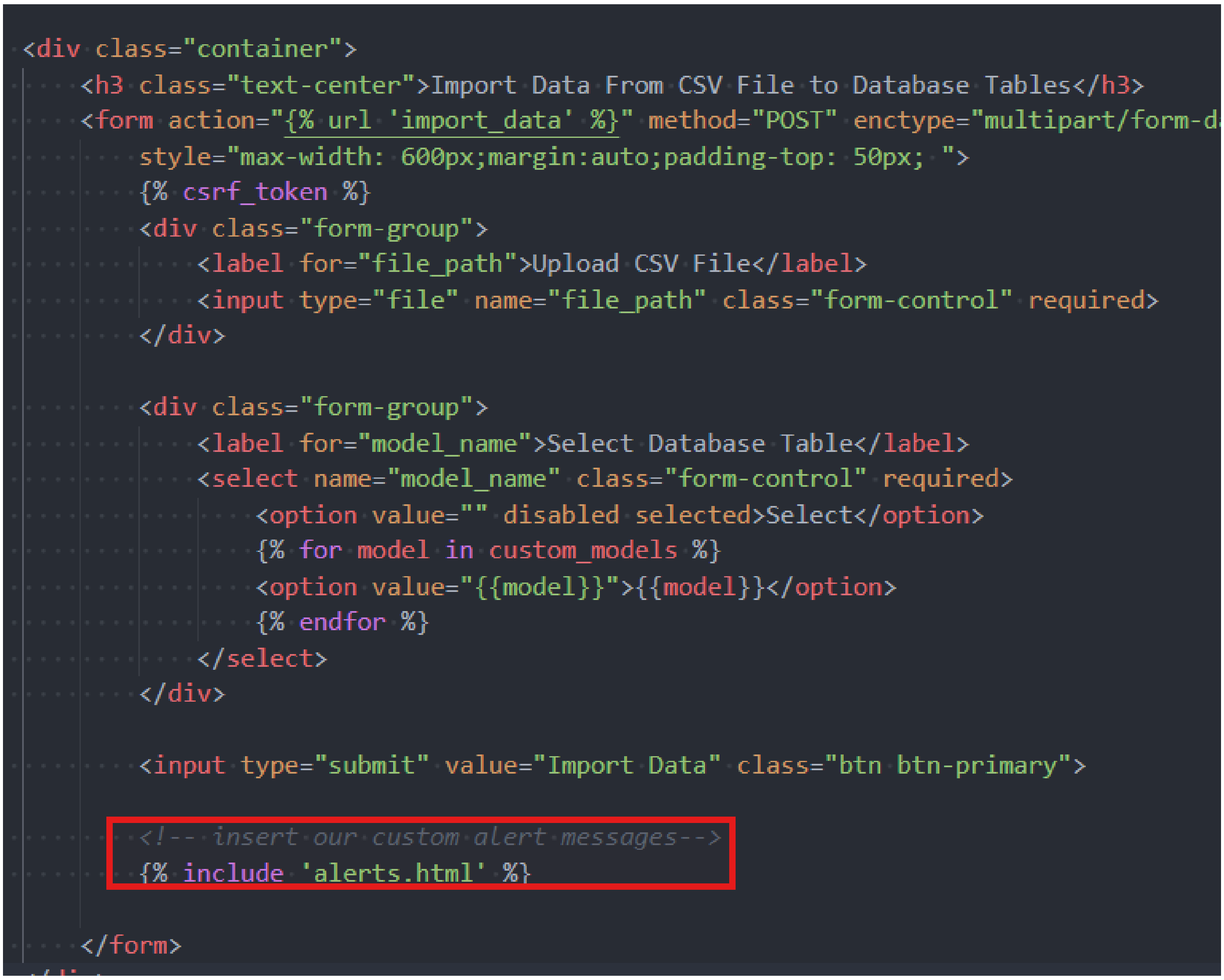
So, in our ALERTS.HTML, we add the same code:



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

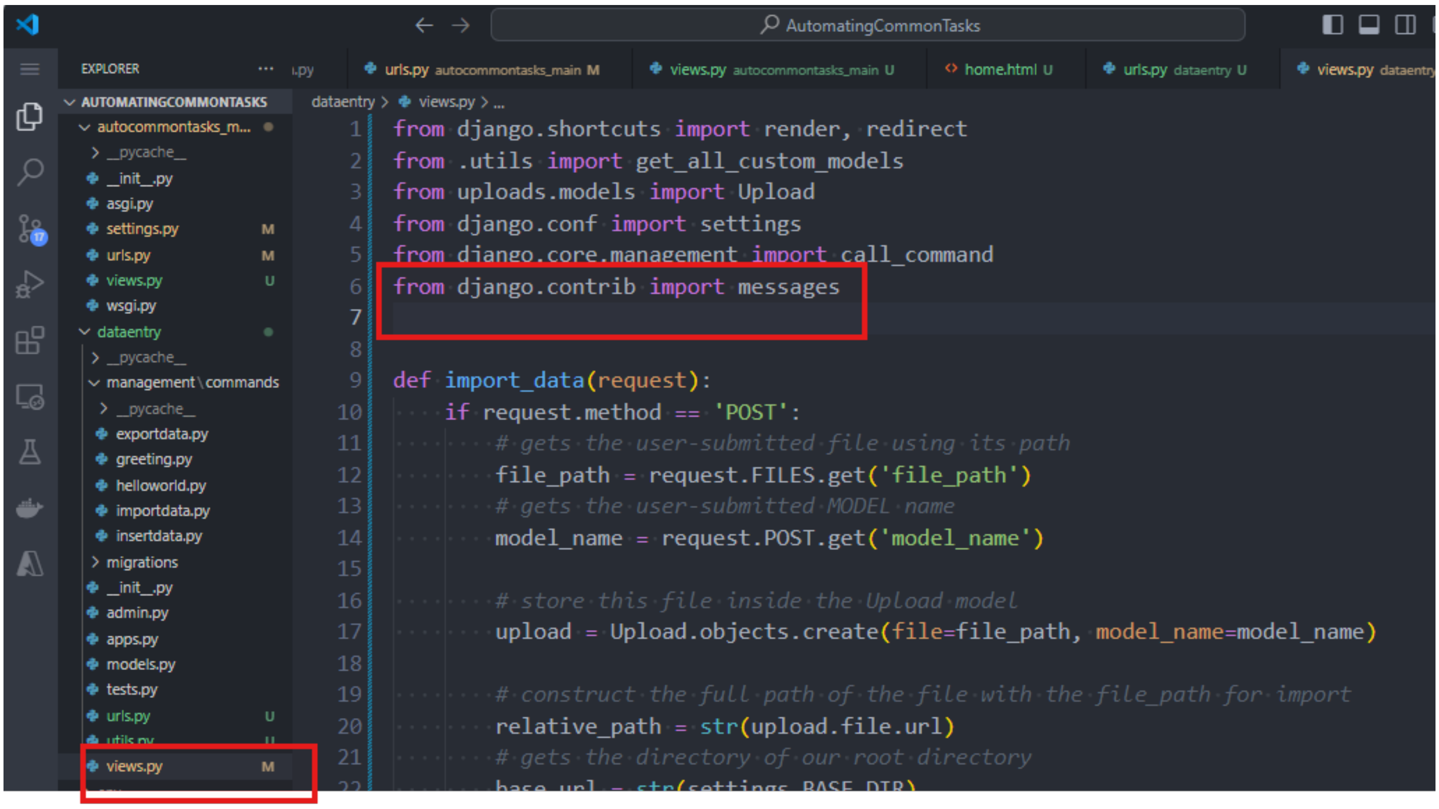
3. Now you can call this ALERTS.HTML in ANY HTML that will require messages or comments.

So, in DATAENTRY\IMPORTDATA.HTML, we add

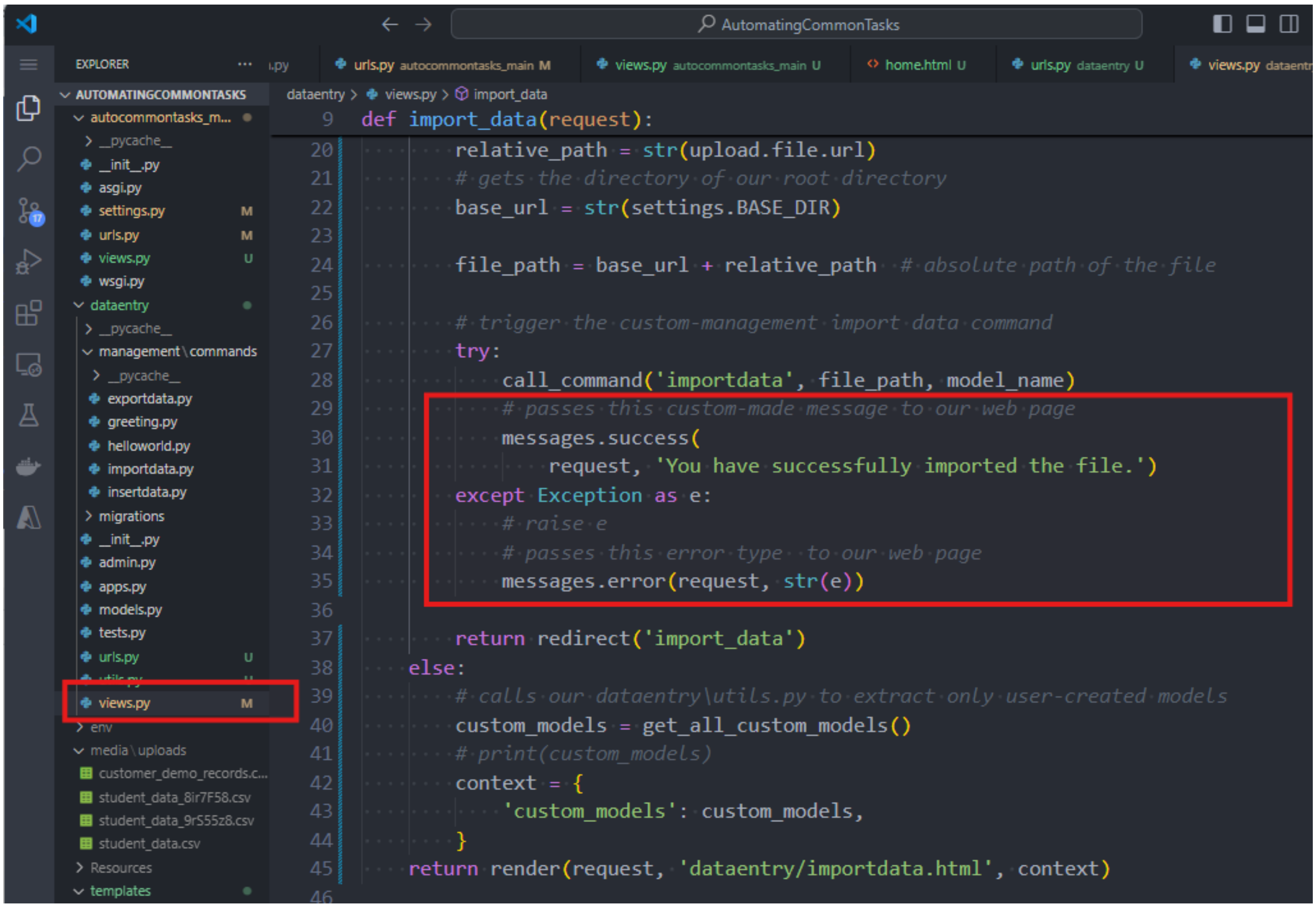


```html
<div class="container">
    <h3 class="text-center">Import Data From CSV File to Database Tables</h3>
    <form action="{% url 'import_data' %}" method="POST" enctype="multipart/form-d
        style="max-width: 600px;margin:auto;padding-top: 50px; ">
        {% csrf_token %}
        <div class="form-group">
            <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

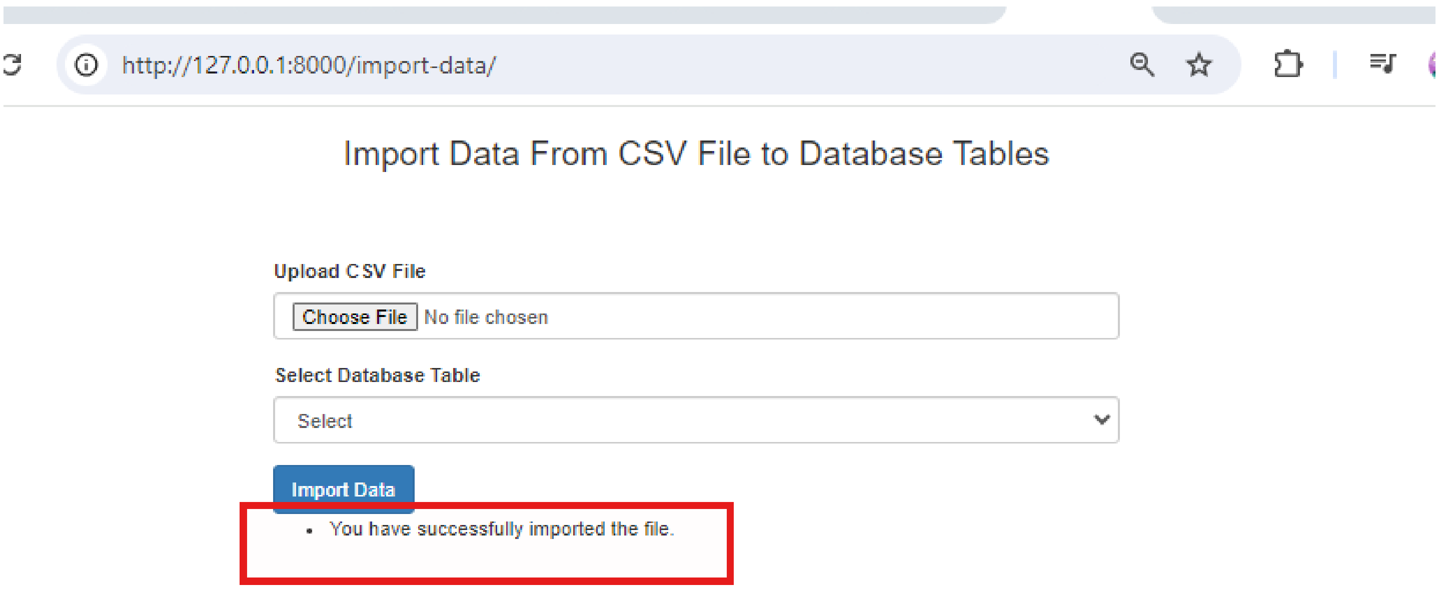


```python
from django.shortcuts import render, redirect
from .utils import get_all_custom_models
from uploads.models import Upload
from django.conf import settings
from django.core.management import call_command
from django.contrib import messages


def import_data(request):
    if request.method == 'POST':
        # gets the user-submitted file using its path
        file_path = request.FILES.get('file_path')
        # gets the user-submitted MODEL name
        model_name = request.POST.get('model_name')

        # store this file inside the Upload model
        upload = Upload.objects.create(file=file_path, model_name=model_name)

        # construct the full path of the file with the file_path for import
        relative_path = str(upload.file.url)
        # gets the directory of our root directory
        base_url = str(settings.BASE_DIR)
```



```python
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.



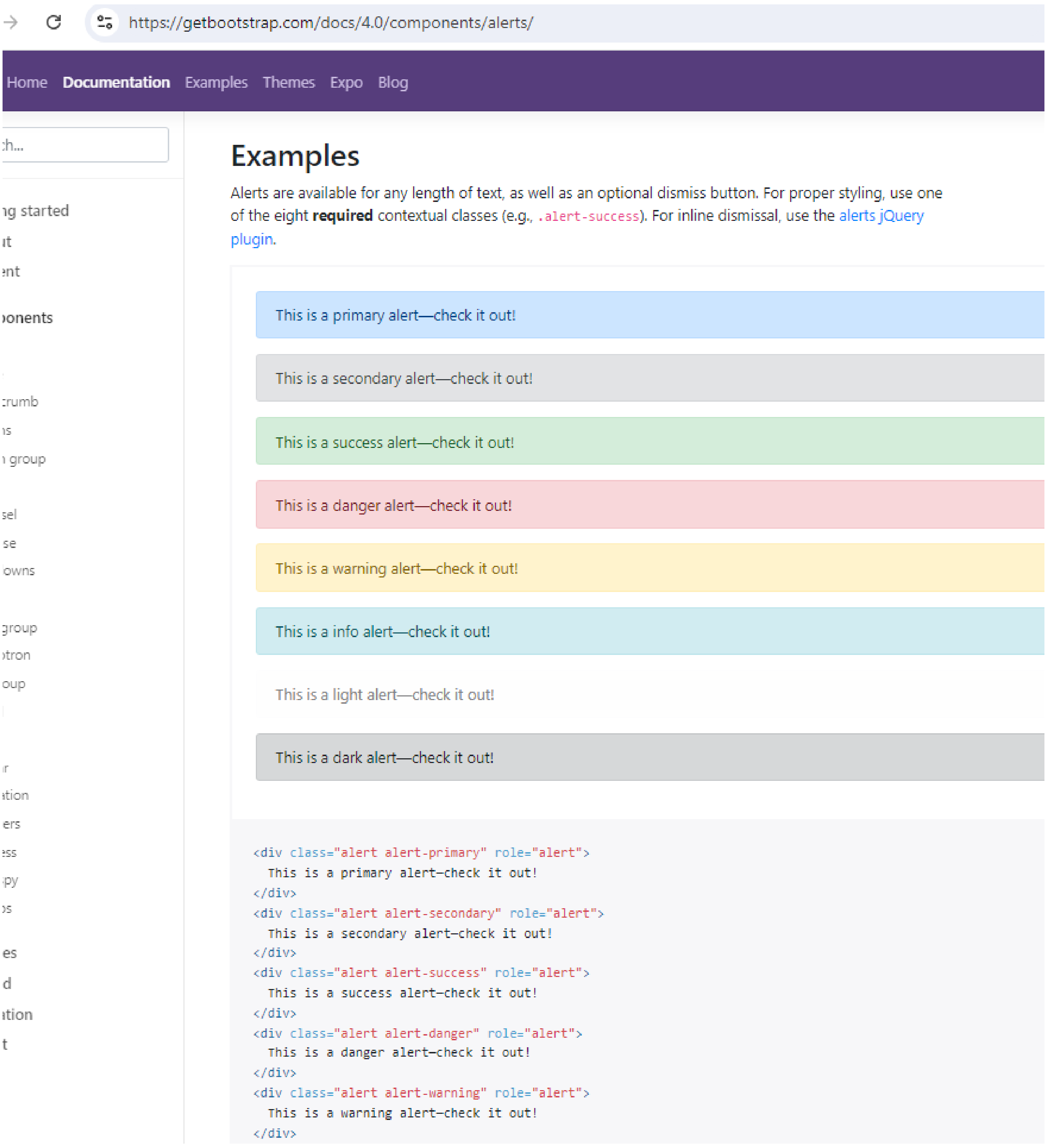
Import Data From CSV File to Database Tables

Upload CSV File

Select Database Table

- You have successfully imported the file.

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


https://getbootstrap.com/docs/4.0/components/alerts/
Home Documentation Examples Themes Expo Blog

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!

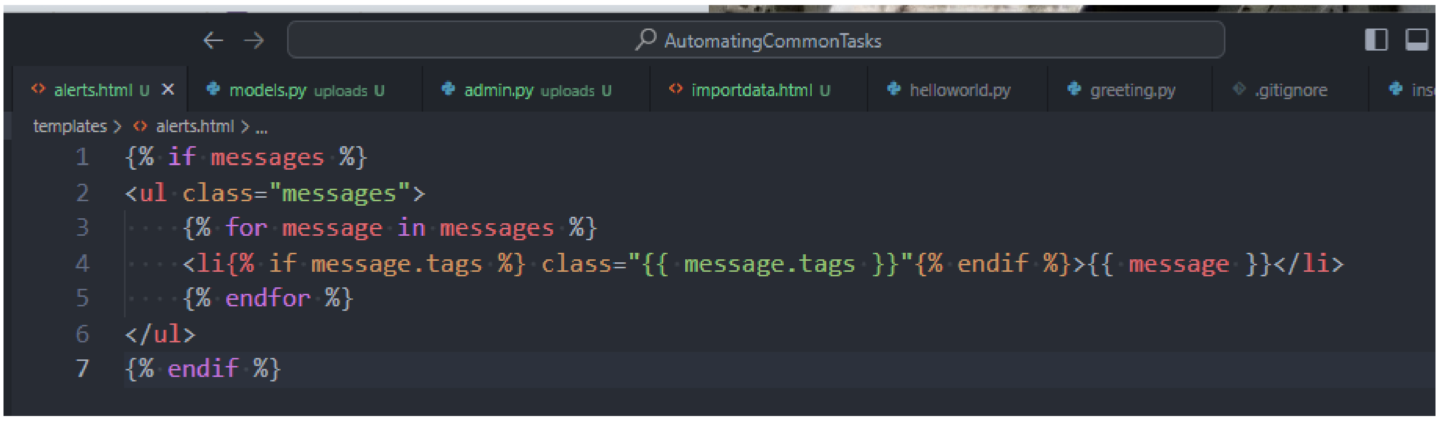
This is a info alert—check it out!

This is a light alert—check it out!

This is a dark alert—check it out!

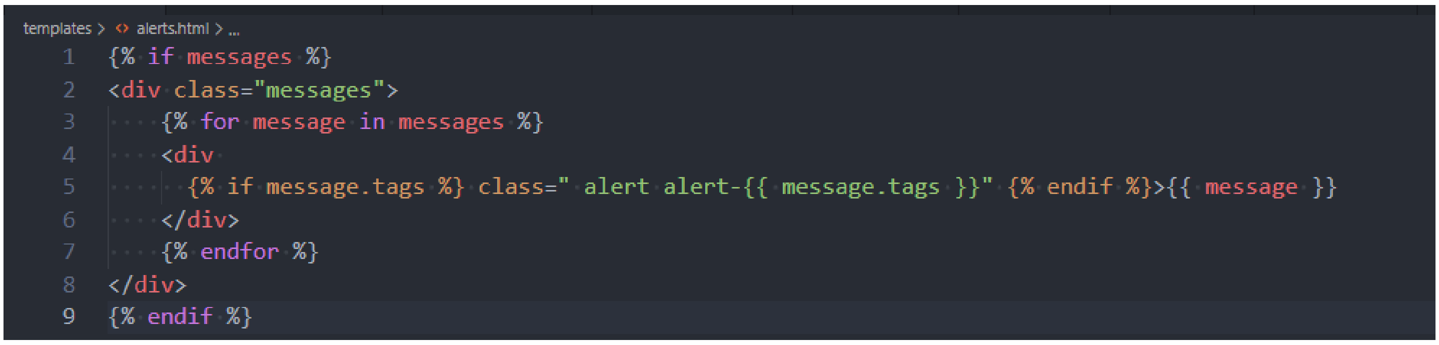
```
<div class="alert alert-primary" role="alert">
  This is a primary alert—check it out!
</div>
<div class="alert alert-secondary" role="alert">
  This is a secondary alert—check it out!
</div>
<div class="alert alert-success" role="alert">
  This is a success alert—check it out!
</div>
<div class="alert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



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

ALERTS.HTML AFTER:

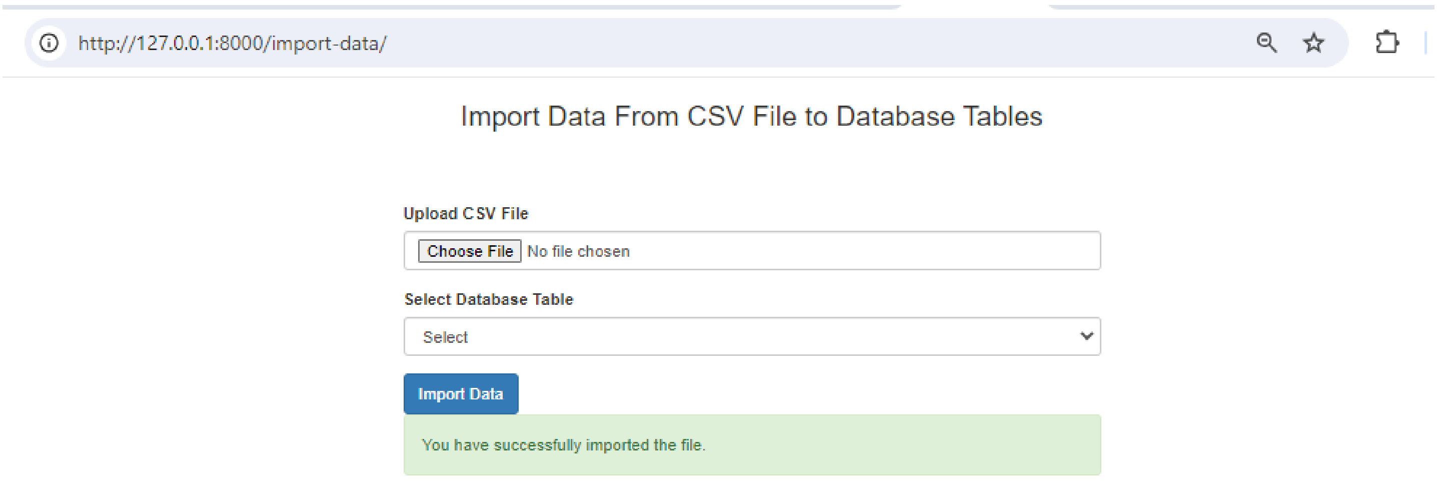


templates > alerts.html > ...

```
{% if messages %}
<div class="messages">
    {% for message in messages %}
    <div
        {% if message.tags %} class=" alert alert-{{ message.tags }}" {% endif %}>{{ message }}
    </div>
    {% endfor %}
</div>
{% 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

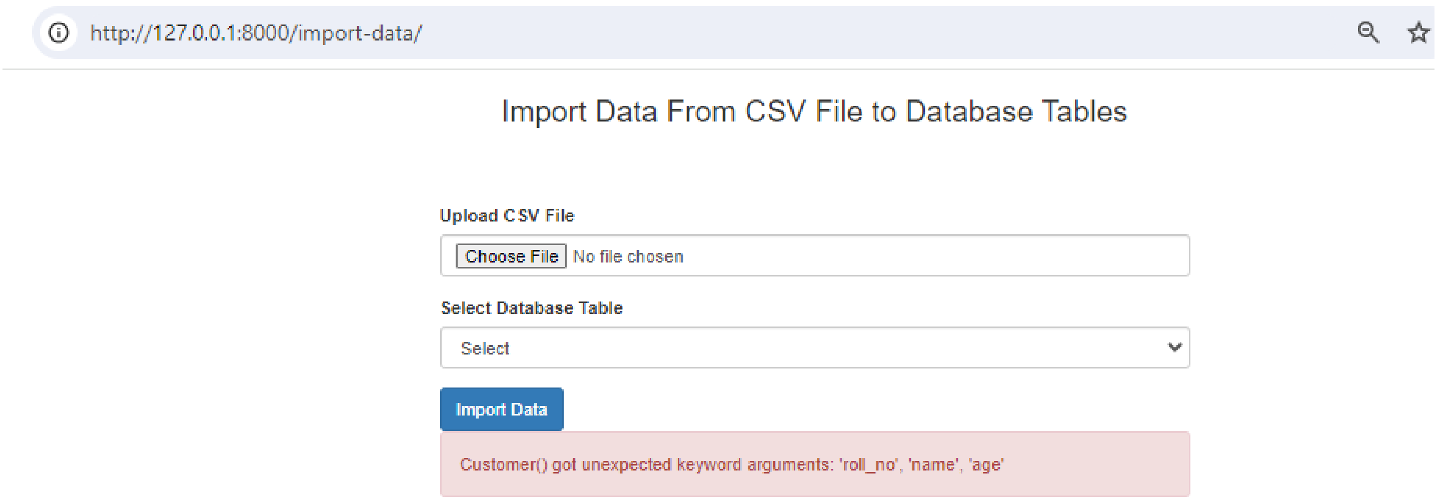
Select Database Table

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

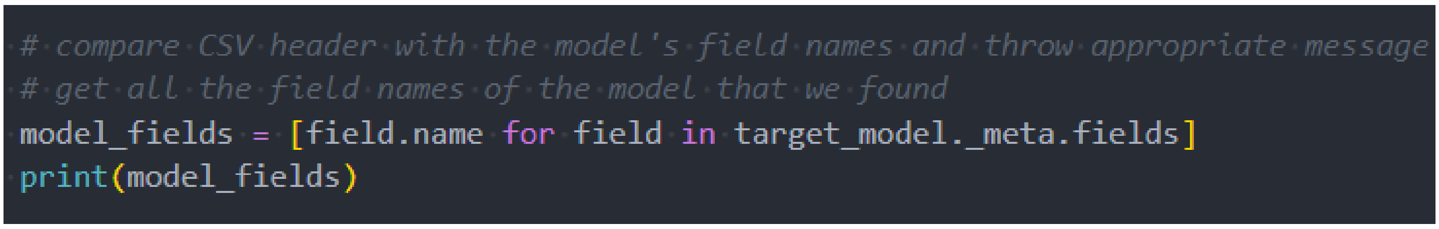
Import Data

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

Or a different file to CUSTOMER model.

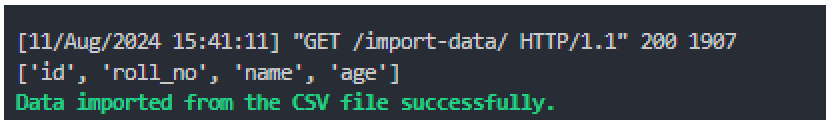
8. To modify the error message into something user-friendly especially the user has used a different file with a different model, we update our IMPORTDATA.PY

To get the fields of our model, in our IMPORTDATA.PY, we add this:



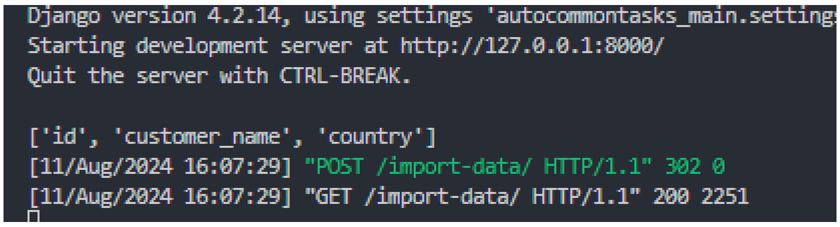
```python
# compare CSV header with the model's field names and throw appropriate message
# get all the field names of the model that we found
model_fields = [field.name for field in target_model._meta.fields]
print(model_fields)
```

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



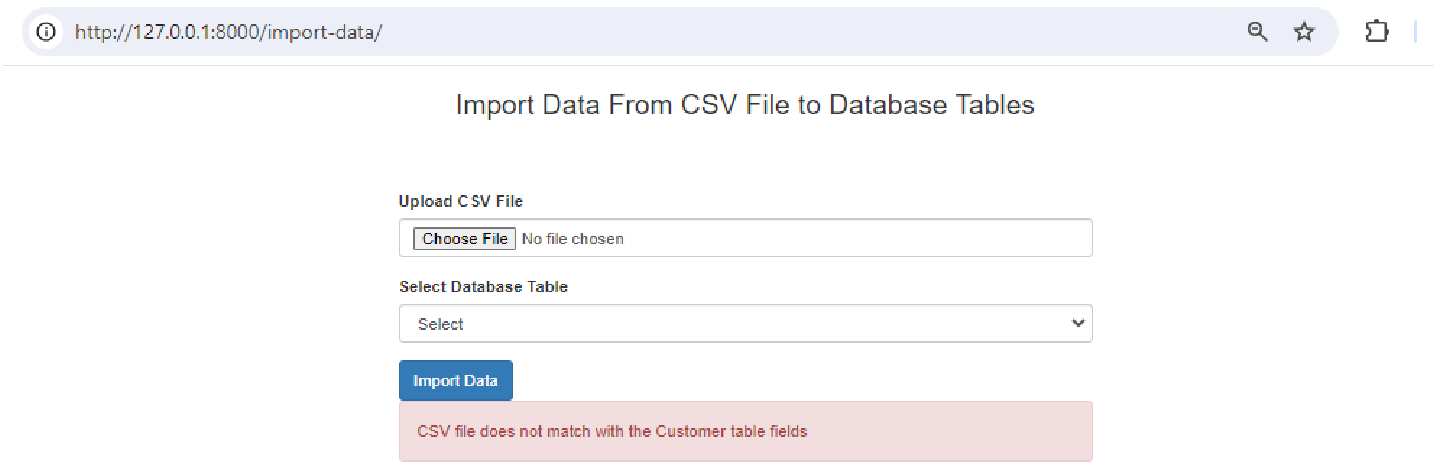
```
[11/Aug/2024 15:41:11] "GET /import-data/ HTTP/1.1" 200 1907
['id', 'roll_no', 'name', 'age']
Data imported from the CSV file successfully.
```

CUSTOMER MODEL:



```
Django version 4.2.14, using settings 'autocommontasks_main.setting
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:



Import Data From CSV File to Database Tables

Upload CSV File

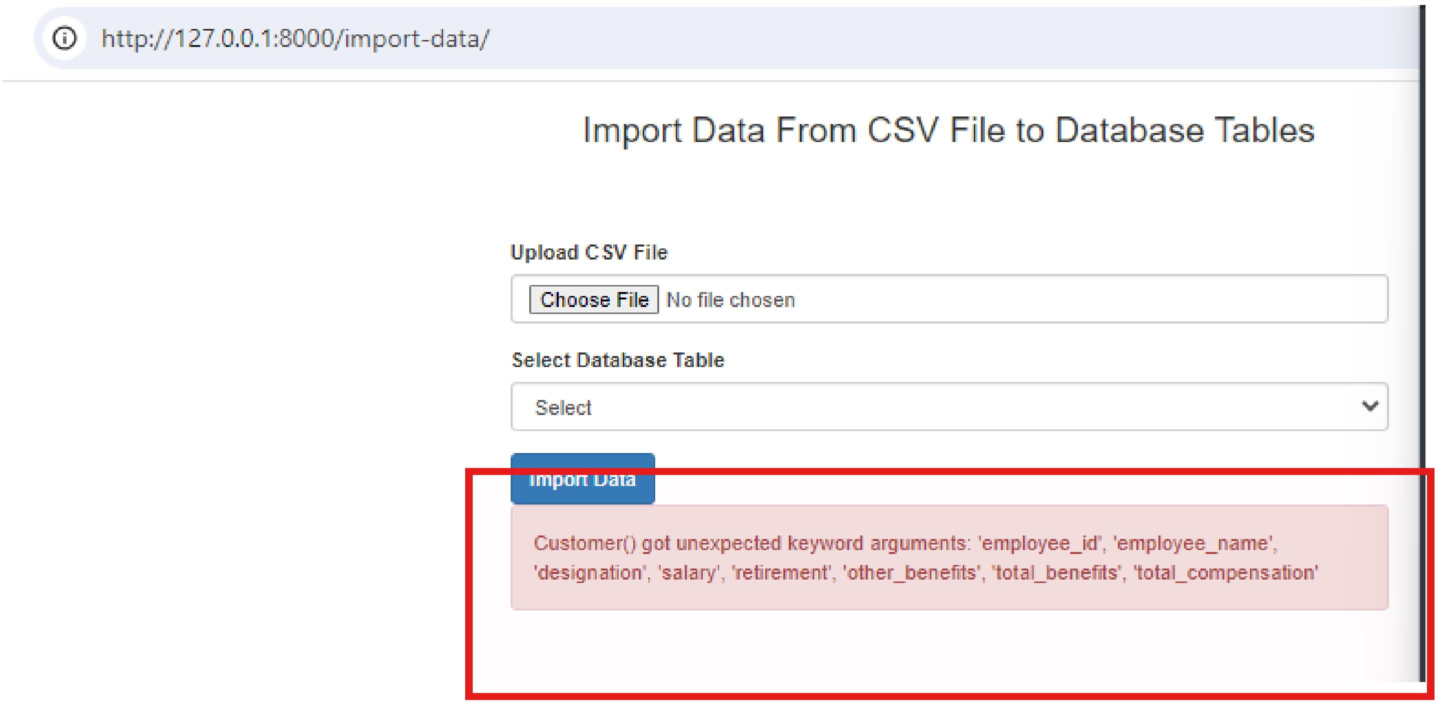
No file chosen

Select Database Table

Select

CSV file does not match with the Customer table fields

Instead of this:



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

Import Data From CSV File to Database Tables

Upload CSV File

Choose File No file chosen

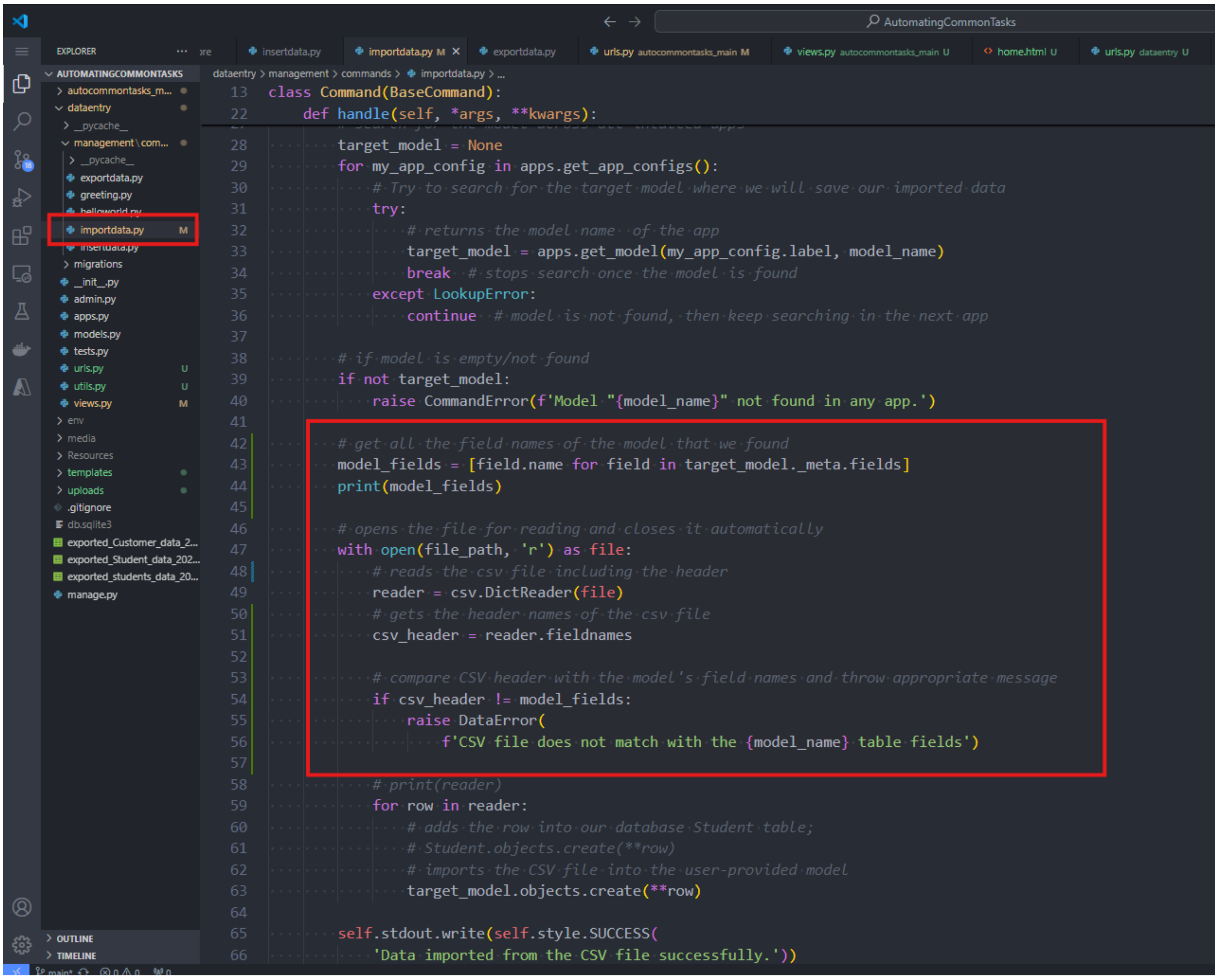
Select Database Table

Select

Import Data

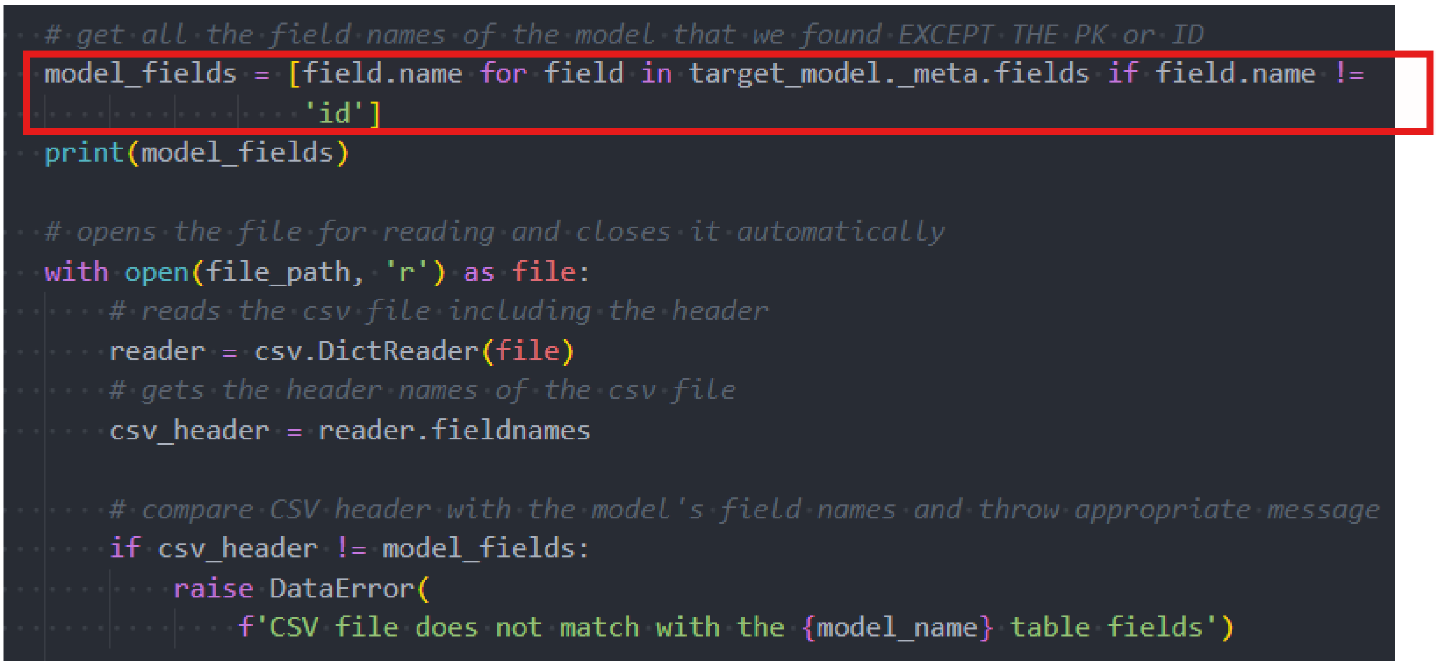
```
Customer() got unexpected keyword arguments: 'employee_id', 'employee_name',
'designation', 'salary', 'retirement', 'other_benefits', 'total_benefits', 'total_compensation'
```

9. Your IMPORTDATA.PY should be:



```python
class Command(BaseCommand):
    def handle(self, *args, **kwargs):
        target_model = None
        for my_app_config in apps.get_app_configs():
            # Try to search for the target model where we will save our imported data
            try:
                # returns the model name  of the app
                target_model = apps.get_model(my_app_config.label, model_name)
                break  # stops search once the model is found
            except LookupError:
                continue  # model is not found, then keep searching in the next app

        # if model is empty/not found
        if not target_model:
            raise CommandError(f'Model "{model_name}" not found in any app.')

        # 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)

        # 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')

            # print(reader)
            for row in reader:
                # adds the row into our database Student table;
                # Student.objects.create(**row)
                # imports the CSV file into the user-provided model
                target_model.objects.create(**row)

        self.stdout.write(self.style.SUCCESS(
            'Data imported from the CSV file successfully.'))
```

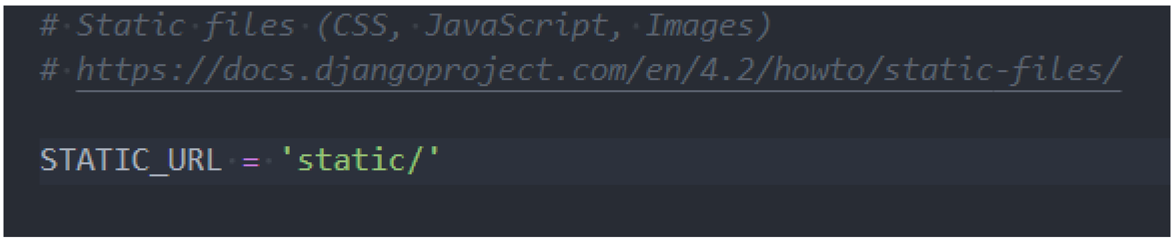
10. But this will give us an error message since we have the the field 'ID' in our table but not in our CSV file, so we update our IMPORTDATA.PY as



```
    # 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')
```

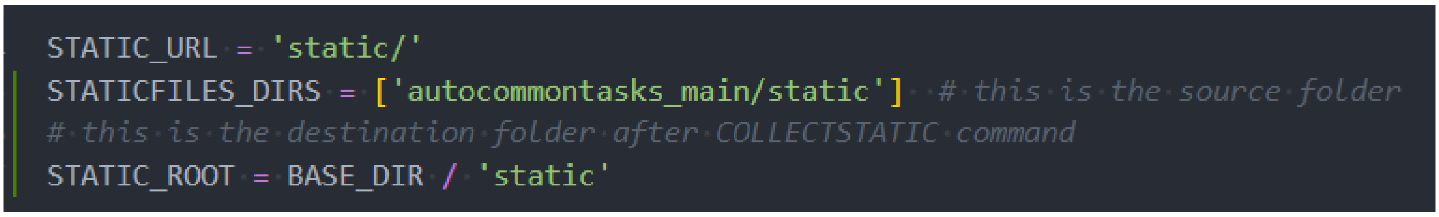
11. IF WE WANT TO HIDE THE ERROR MESSAGES RIGHT AWAY AFTER IT WAS DISPLAYED, we create a JAVASCRIPT. But we need to update our SETTINGS.PY STATIC FILES.

FROM:



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

STATIC_URL = 'static/'
```

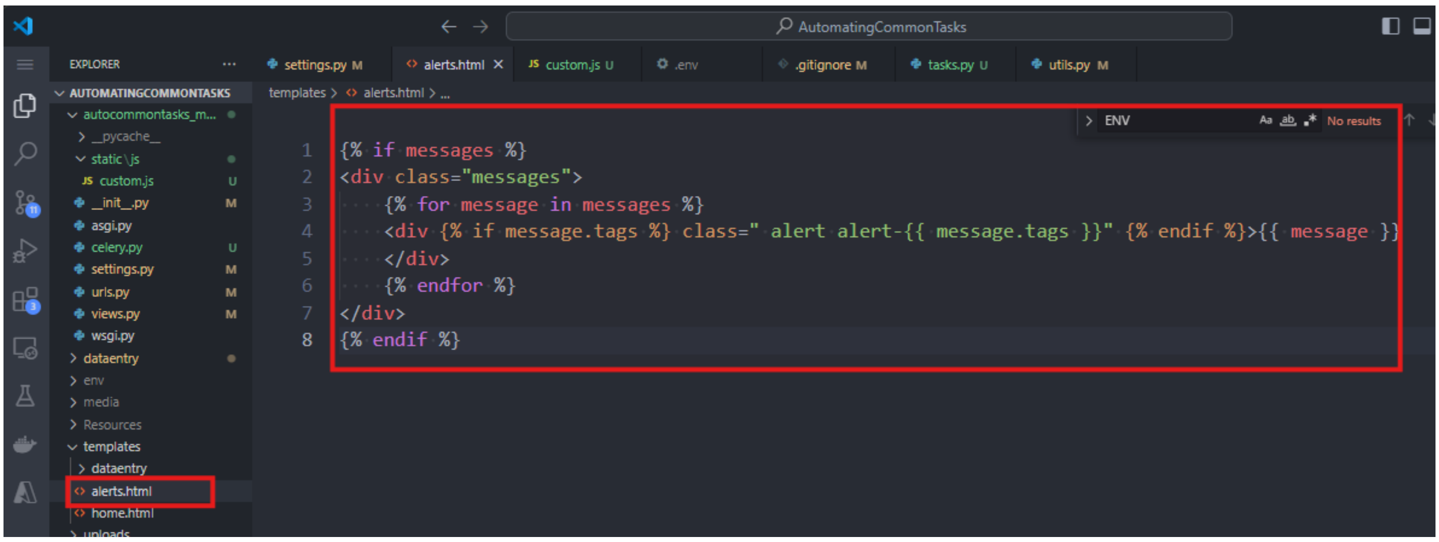
TO:



```
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'
```

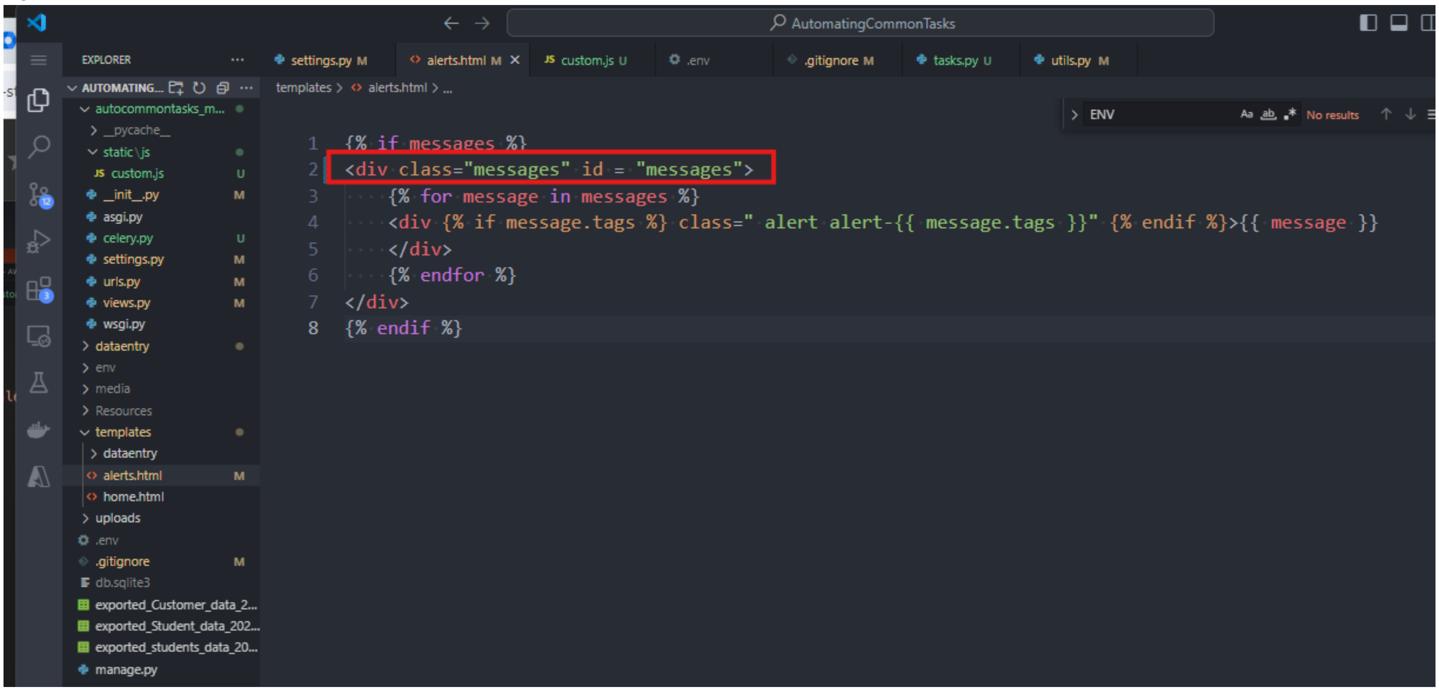
12. Create an ID FOR YOUR ALERTS.HTML

FROM:



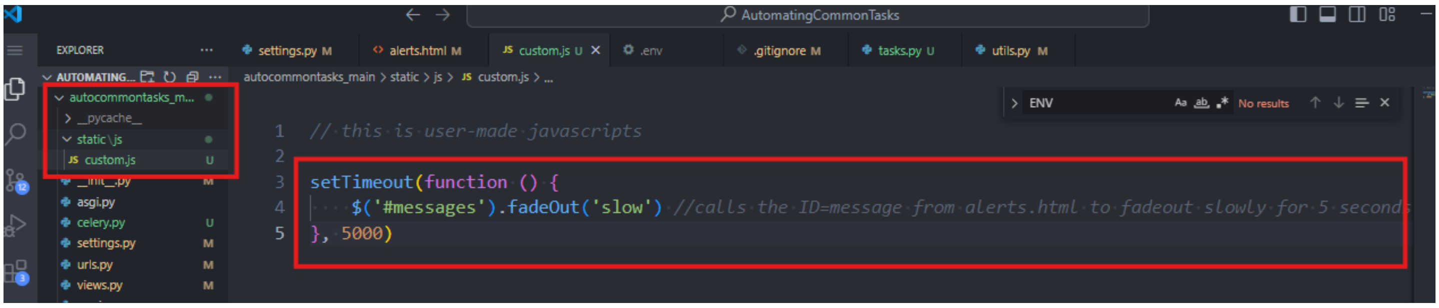
```
{% if messages %}
<div class="messages">
    {% for message in messages %}
    <div {% if message.tags %} class=" alert alert-{{ message.tags }}" {% endif %}>{{ message }}
    </div>
    {% endfor %}
</div>
{% endif %}
```

TO:



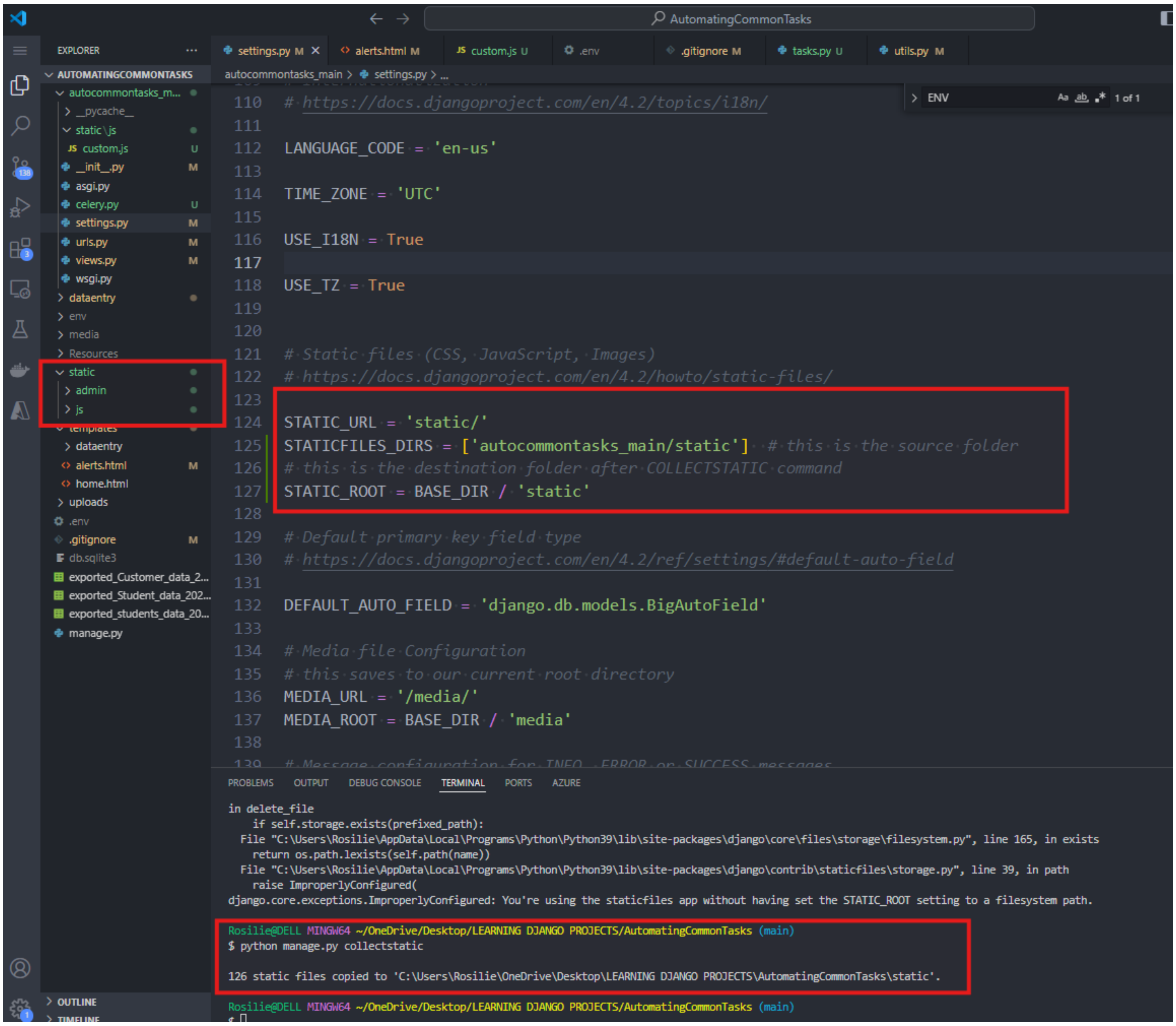
```
{% if messages %}
<div class="messages" id = "messages">
    {% for message in messages %}
    <div {% if message.tags %} class=" alert alert-{{ message.tags }}" {% endif %}>{{ message }}
    </div>
    {% endfor %}
</div>
{% endif %}
```

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



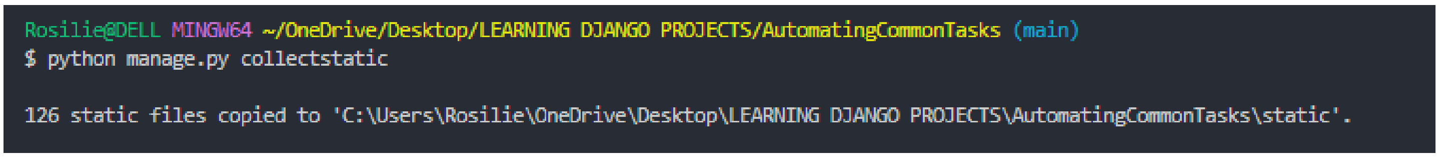
```
// this is user-made javascripts

setTimeout(function () {
    $('#messages').fadeOut('slow') //calls the ID=message from alerts.html to fadeout slowly for 5 second
}, 5000)
```

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.



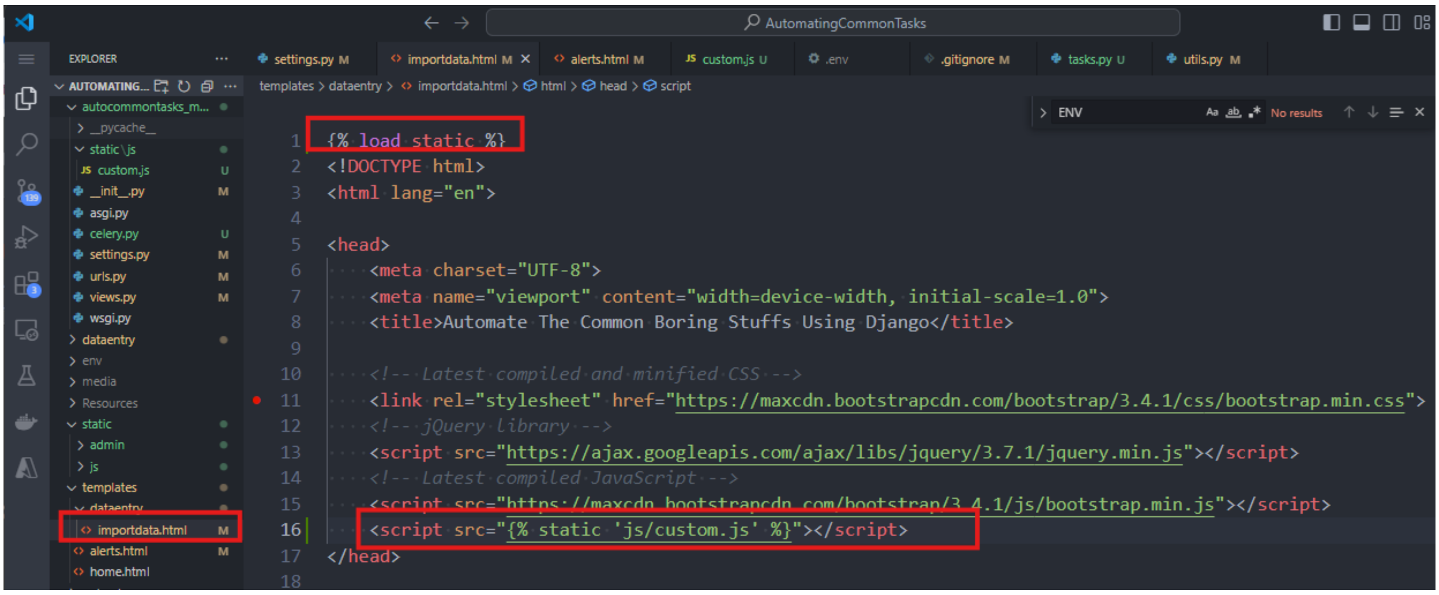
```python
# https://docs.djangoproject.com/en/4.2/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_TZ = True


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

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'

# Default primary key field type
# https://docs.djangoproject.com/en/4.2/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

# Media file Configuration
# this saves to our current root directory
MEDIA_URL = '/media/'
MEDIA_ROOT = BASE_DIR / 'media'

# Message configuration for INFO, ERROR or SUCCESS messages
```

```
in delete_file
    if self.storage.exists(prefixed_path):
  File "C:\Users\Rosilie\AppData\Local\Programs\Python\Python39\lib\site-packages\django\core\files\storage\filesystem.py", line 165, in exists
    return os.path.lexists(self.path(name))
  File "C:\Users\Rosilie\AppData\Local\Programs\Python\Python39\lib\site-packages\django\contrib\staticfiles\storage.py", line 39, in path
    raise ImproperlyConfigured(
django.core.exceptions.ImproperlyConfigured: You're using the staticfiles app without having set the STATIC_ROOT setting to a filesystem path.

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'.

Rosilie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
$
```



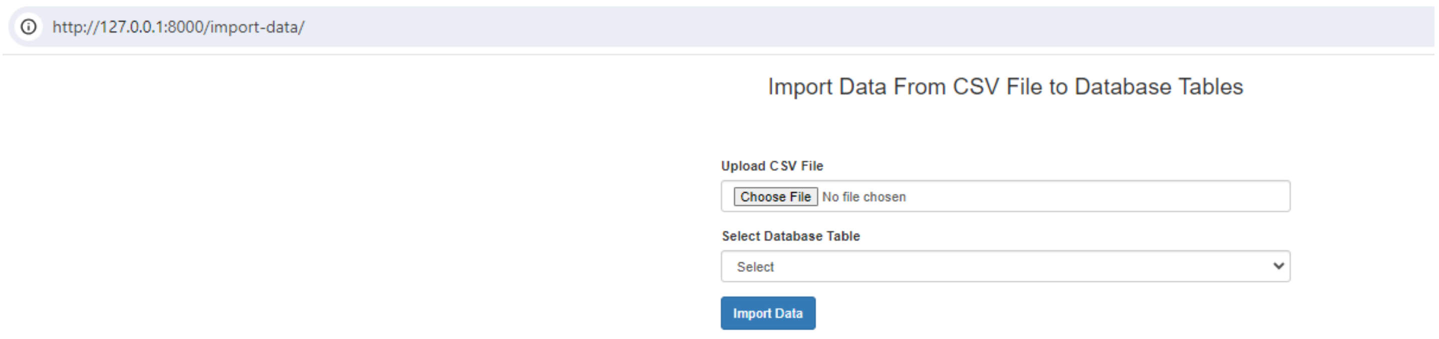
```
Rosilie@DELL MINGW64 ~/OneDrive/Desktop/LEARNING DJANGO PROJECTS/AutomatingCommonTasks (main)
$ python manage.py collectstatic

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



```html
{% load static %}
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Automate The Common Boring Stuffs Using Django</title>

    <!-- Latest compiled and minified CSS -->
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
    <!-- jQuery library -->
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.1/jquery.min.js"></script>
    <!-- Latest compiled JavaScript -->
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>
    <script src="{% static 'js/custom.js' %}"></script>
</head>
```

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.