Topic: Custom Management Command 9: Importing Large Data Using Celery & Redis Debugging

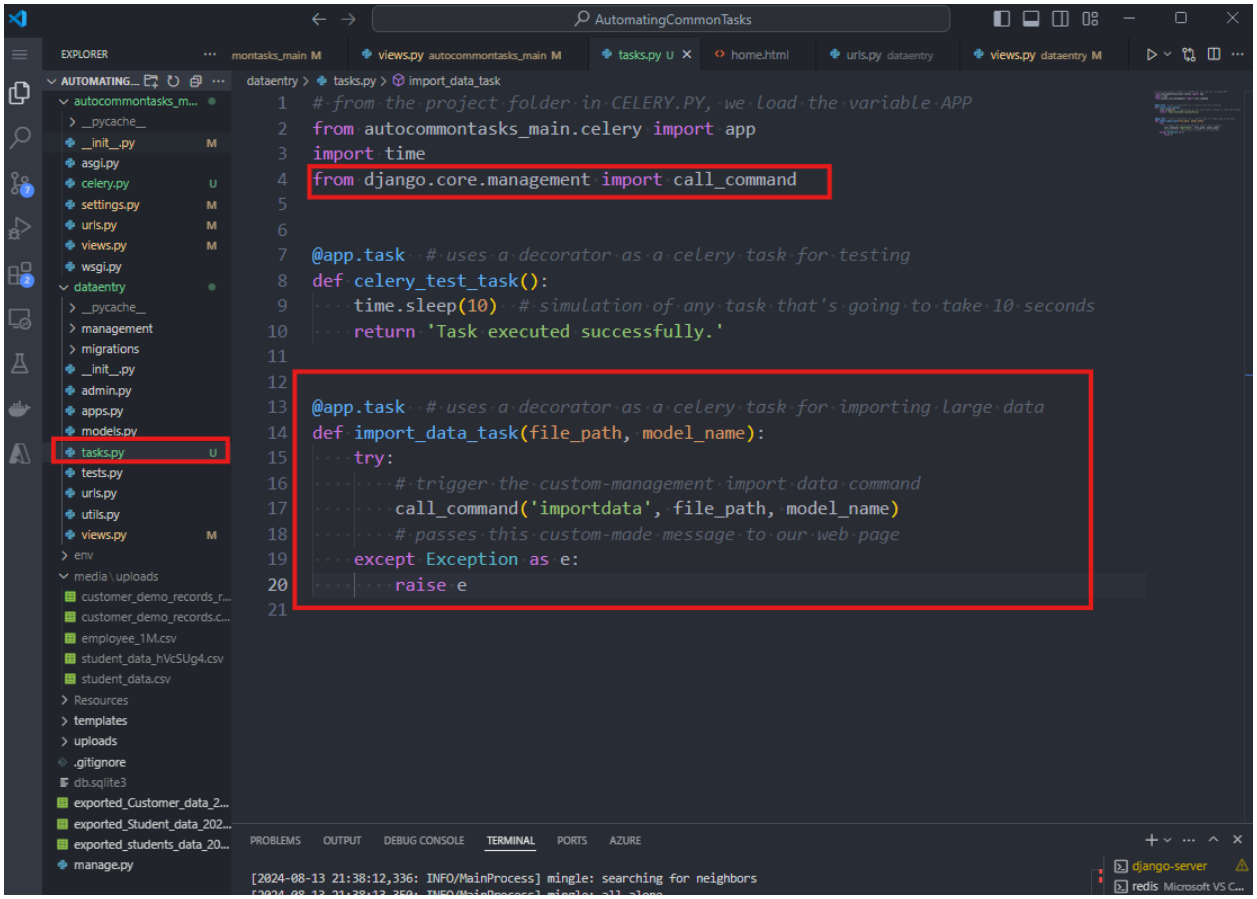
Speaker: Udemy Instructor Rathan Kumar | Notebook: Django: Automating Common Tasks

---

1. After we installed, set up, and tested our Celery and Redis, it is time to use this on our IMPORTDATA function.

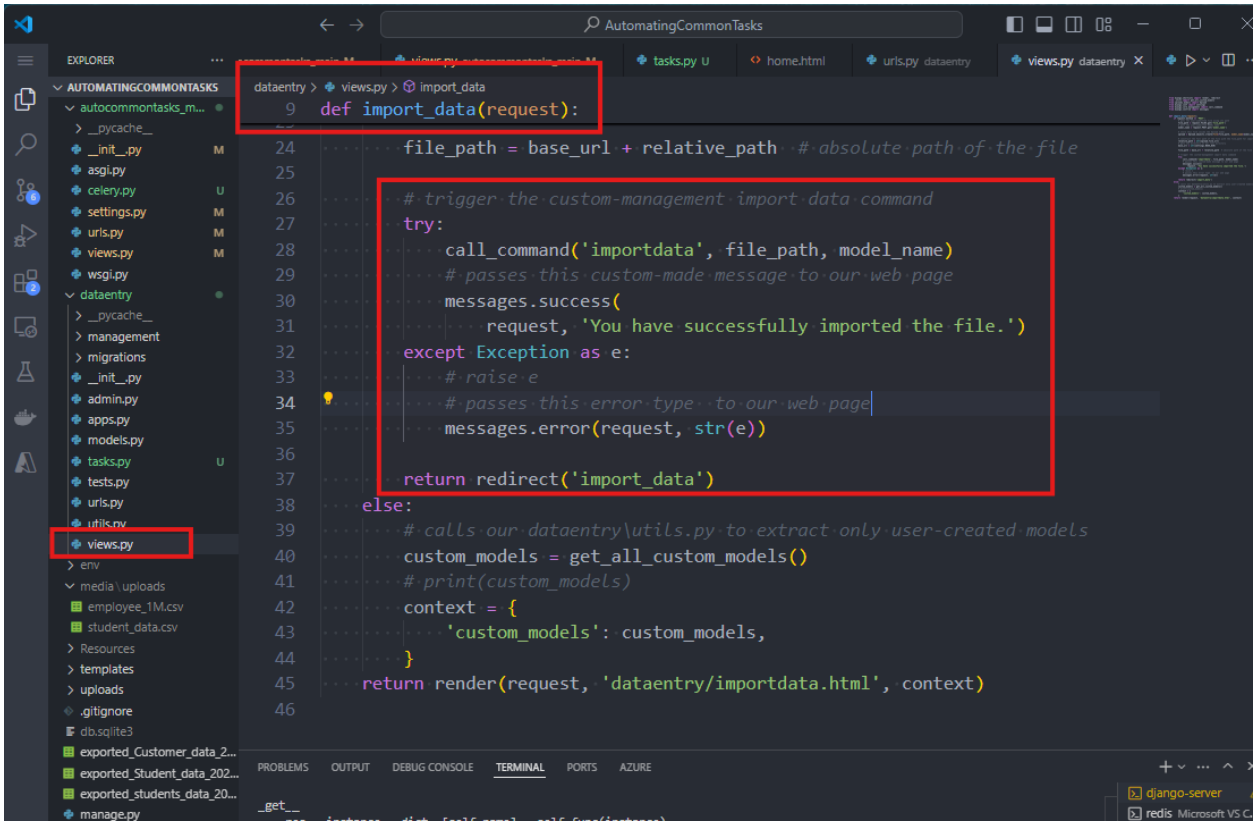
2. Our PROBLEM is that when we import large data like ONE MILLION records of employees, it may take some time (after testing it, it took at least 13 minutes long to save only 130K records, so it might take longer to save all the 1M records.). Our SOLUTION is to use the CELERY function to handle this time-consuming process while we see messages like 'Your data have been saved successfully' and you can do other things.

3. We create a new CELERY function in our DATAENTRY\TASKS.PY and update as:

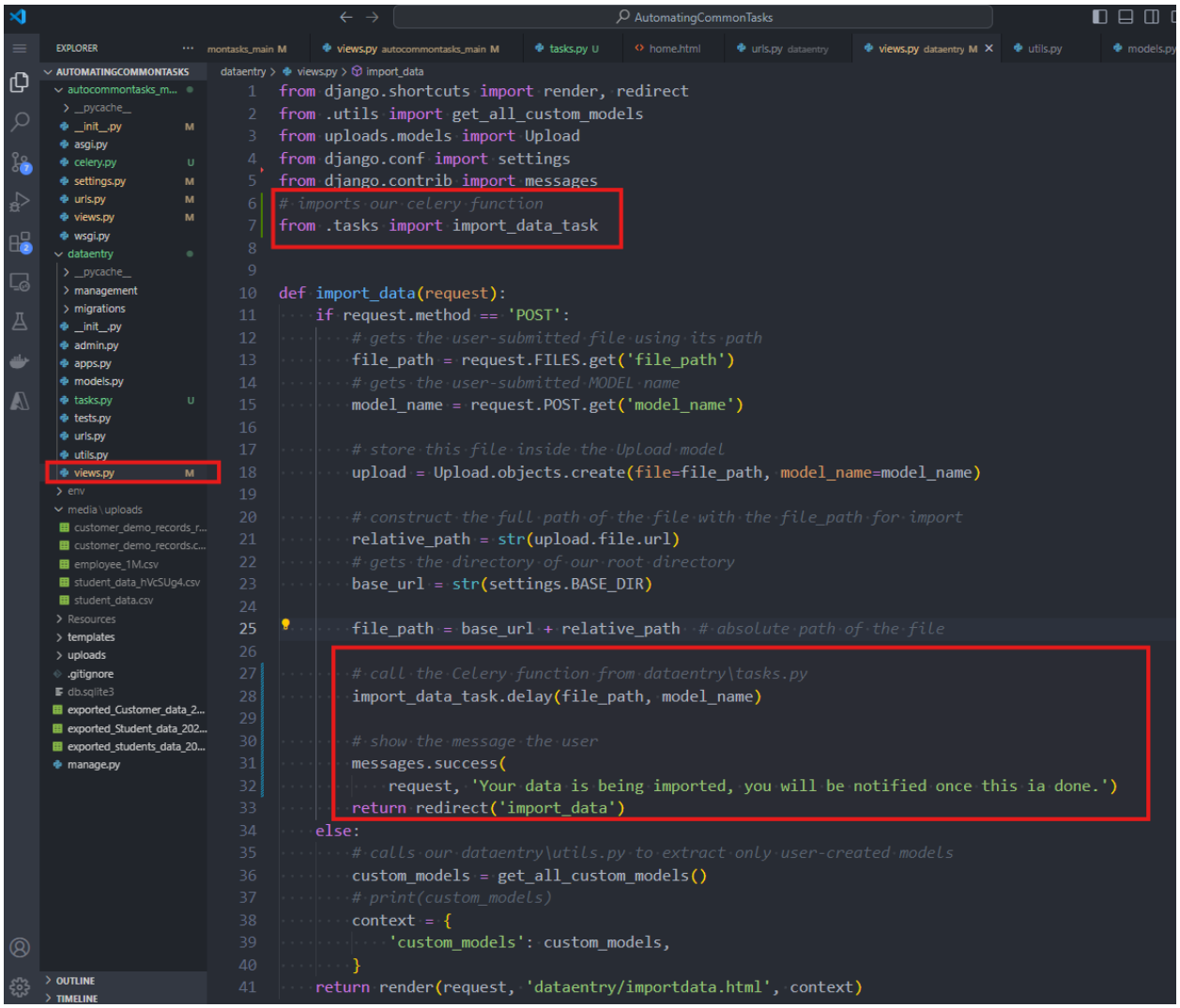


4. In DATAENTRY\VIEWS.PY, we have to update our IMPORT_DATA function, so that the importing part is handled by CELERY instead of by DJANGO.

FROM:

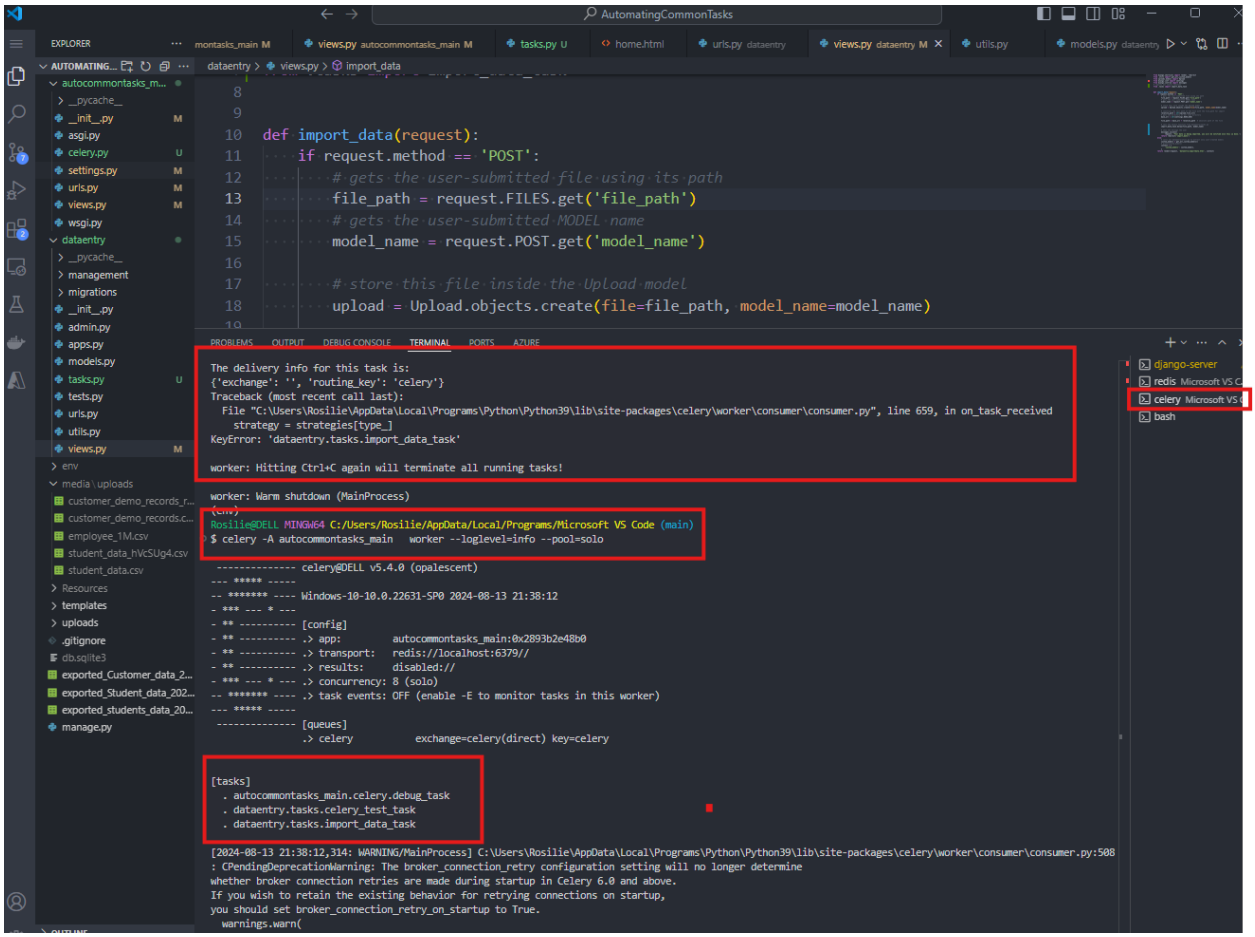


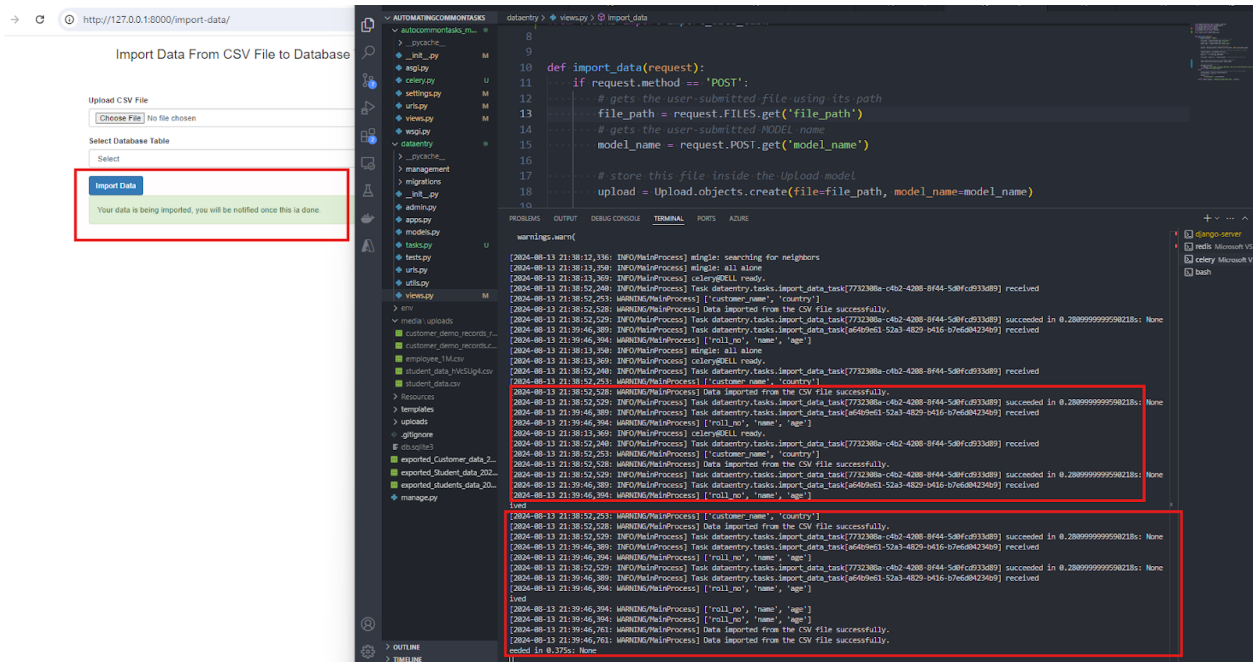
TO:



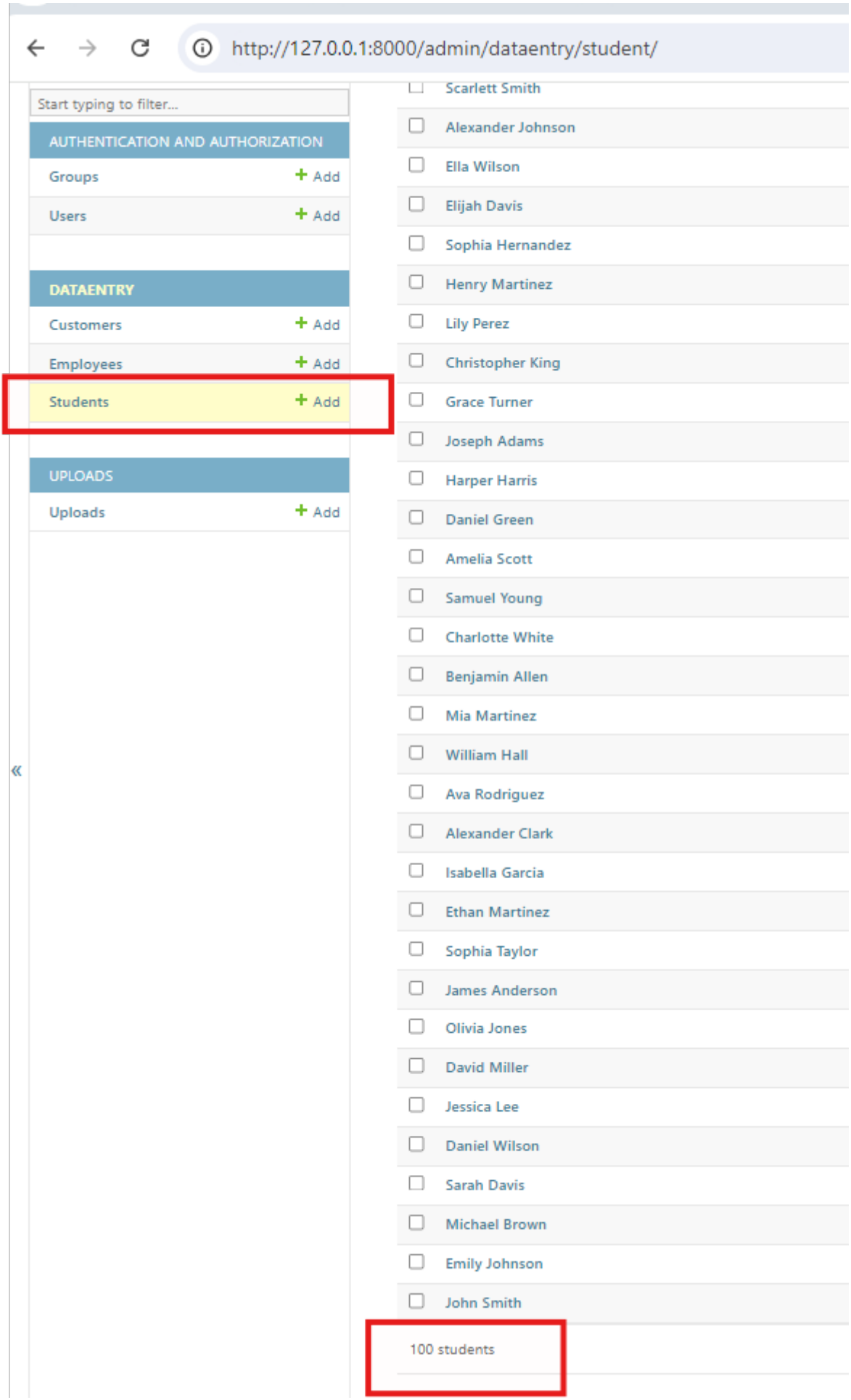
IMPORTANT REMINDER: IF YOU JUST RUN YOUR DJANGO SERVER RIGHT AWAY, YOU WONT SEE ANY UPDATE. THIS IS BECAUSE YOU NEED TO RUN YOUR CELERY COMMAND AGAIN:

$ celery -A autocommontasks_main worker --loglevel=info --pool=solo

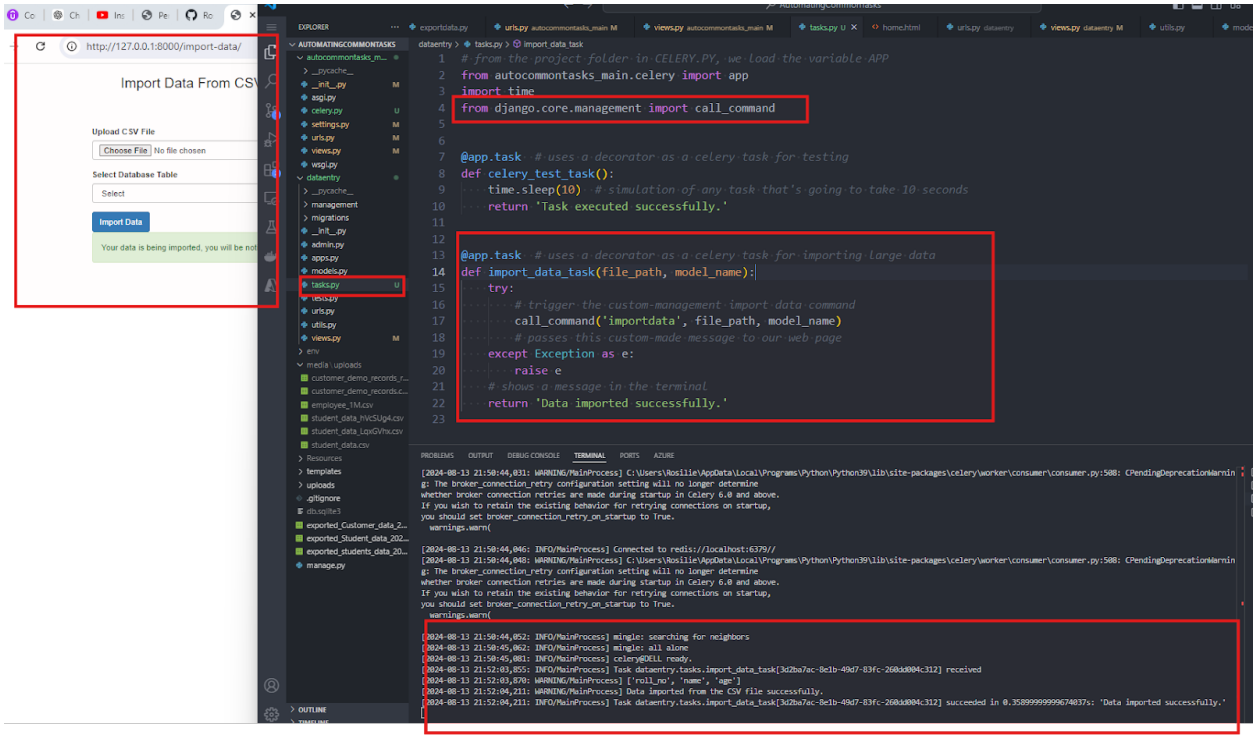




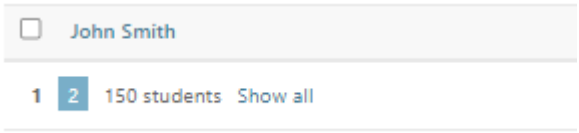
4. When you reload your ADMIN dashboard, you should be able to see the added records from CUSTOMER and STUDENT CSV files.



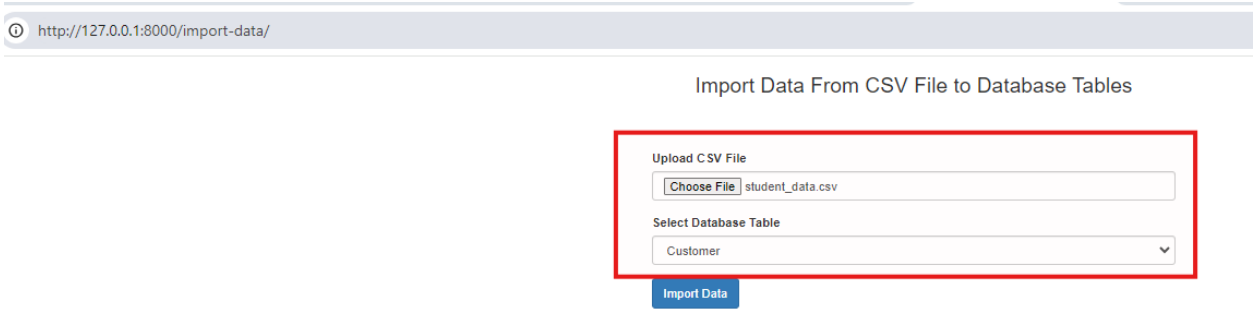
If we need to add a SUCCESS message in our CELERY FUNCTION for an upload of STUDENT CSV file again, we update and run our celery reload command again.

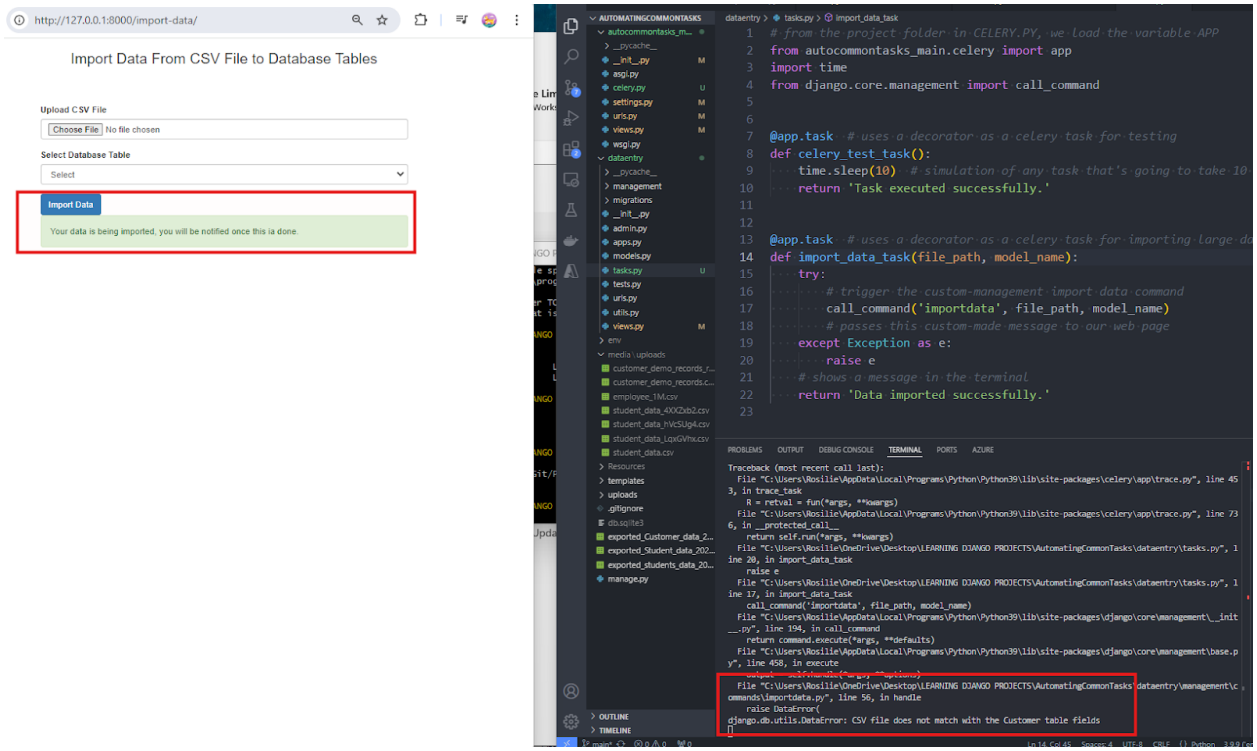


Reloading our ADMIN Dashboard for STUDENT TABLE, IT SHOULD INCREASE IN NUMBER



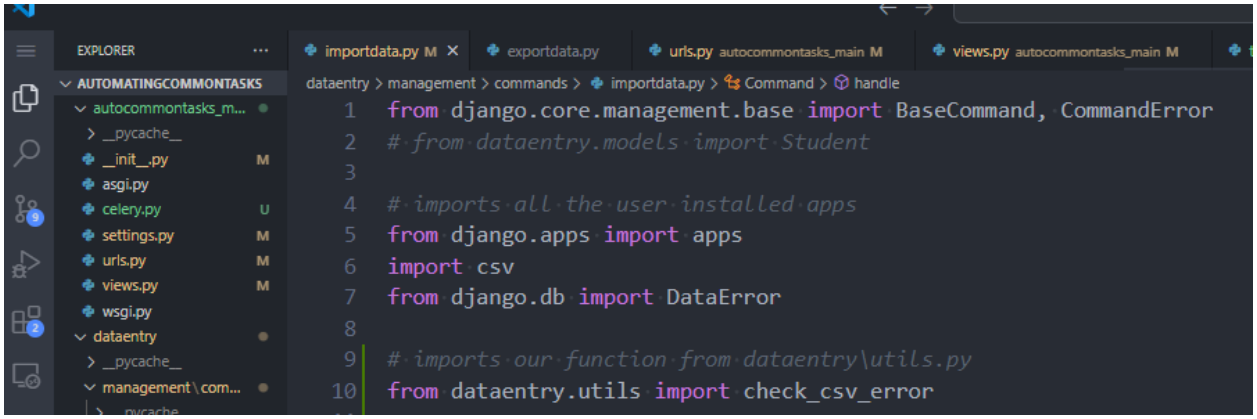
5. When we try to import a CSV file to an incorrect model, Celery will throw an error but Django won't.

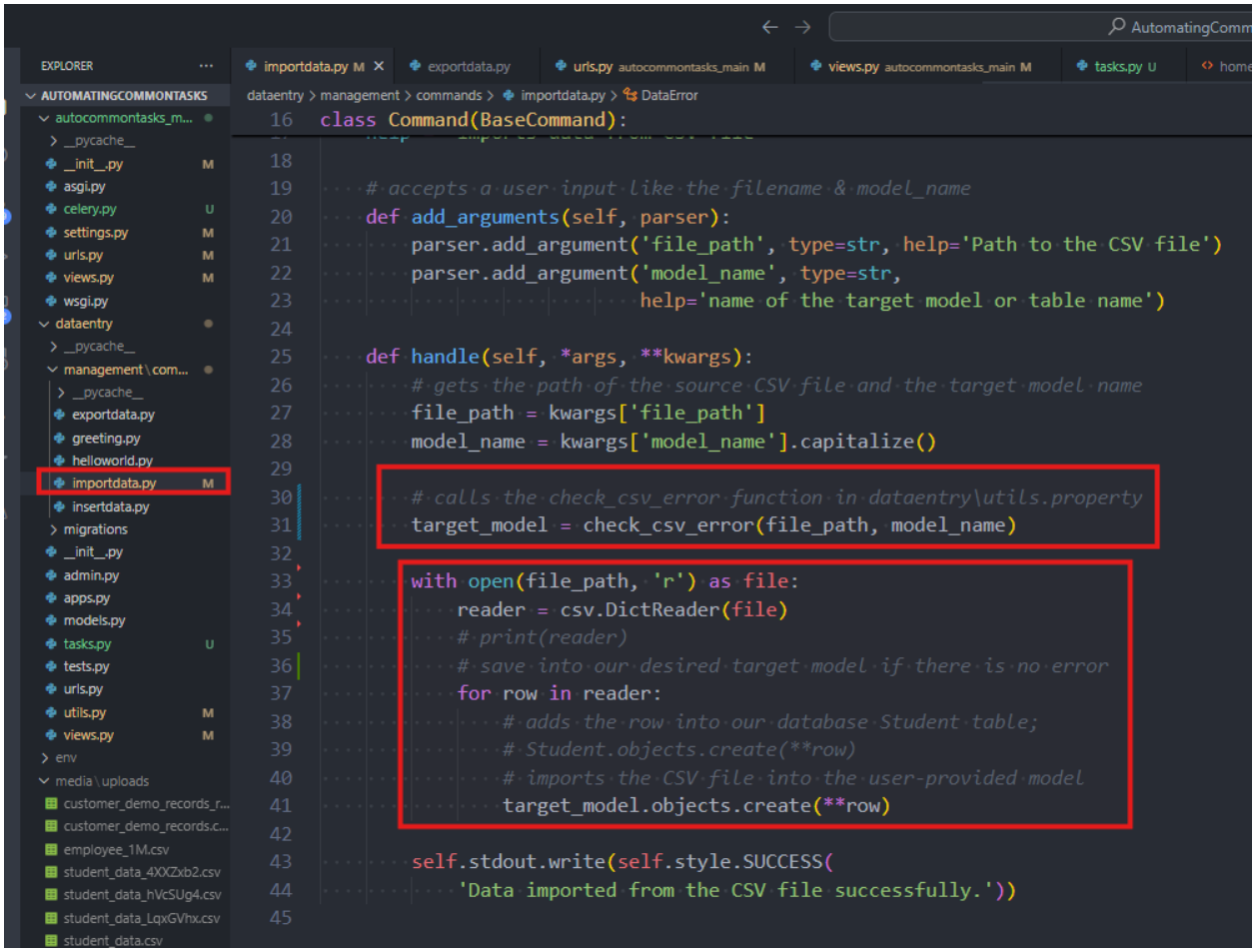




To correct this, we have to an error - handling command BEFORE THE CELERY FUNCTION is being called.

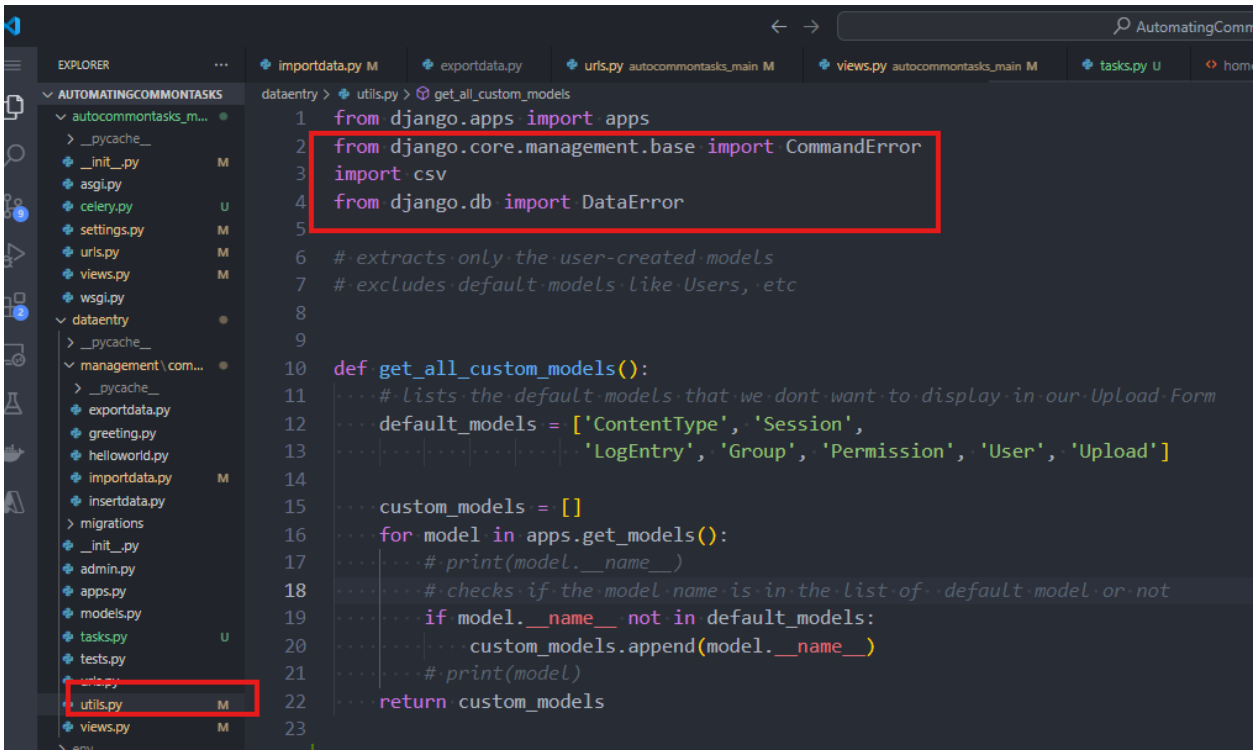
So, go to DATAENTRY\UTILS.PY and move some blocks from MANAGEMENT\COMMANDS\IMPORTDATA.PY. So, our IMPORTDATA.PY updated code now:





```python
class Command(BaseCommand):

    # accepts a user input like the filename & model_name
    def add_arguments(self, parser):
        parser.add_argument('file_path', type=str, help='Path to the CSV file')
        parser.add_argument('model_name', type=str,
                            help='name of the target model or table name')

    def handle(self, *args, **kwargs):
        # gets the path of the source CSV file and the target model name
        file_path = kwargs['file_path']
        model_name = kwargs['model_name'].capitalize()

        # calls the check_csv_error function in dataentry\utils.property
        target_model = check_csv_error(file_path, model_name)

        with open(file_path, 'r') as file:
            reader = csv.DictReader(file)
            # print(reader)
            # save into our desired target model if there is no error
            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.'))
```

Our UTILS.PY updated code where we created a new function, check_csv_error, to allow Error Handling via UTILS.PY instead from IMPORTDATA.PY

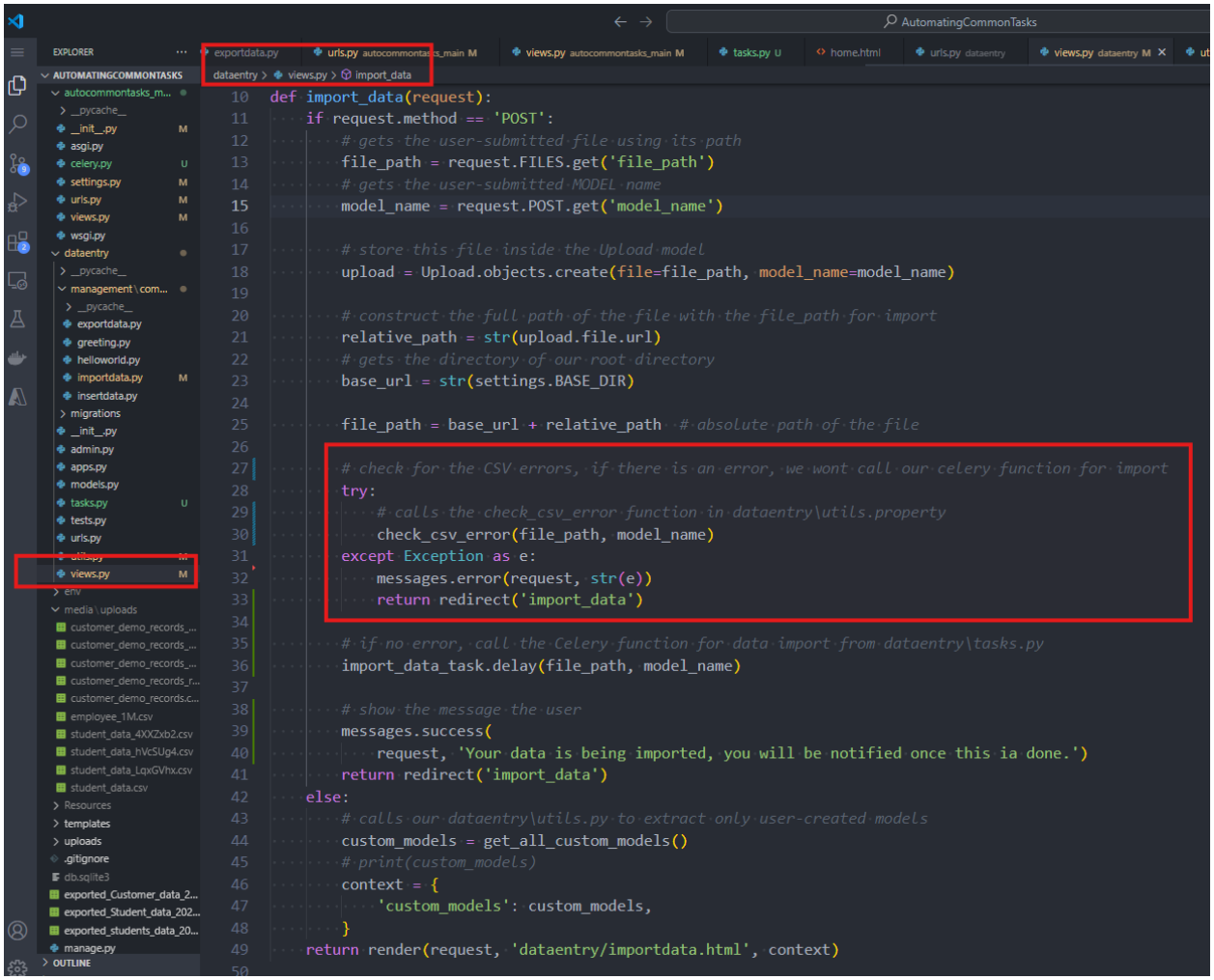


```python
from django.apps import apps
from django.core.management.base import CommandError
import csv
from django.db import DataError

# extracts only the user-created models
# excludes default models like Users, etc


def get_all_custom_models():
    # lists the default models that we dont want to display in our Upload Form
    default_models = ['ContentType', 'Session',
                      'LogEntry', 'Group', 'Permission', 'User', 'Upload']

    custom_models = []
    for model in apps.get_models():
        # print(model.__name__)
        # checks if the model name is in the list of default model or not
        if model.__name__ not in default_models:
            custom_models.append(model.__name__)
        # print(model)
    return custom_models
```

```python
def check_csv_error(file_path, model_name):
    # search for the model across all intalled apps
    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 EXCEPT THE PK or ID
    model_fields = [field.name for field in target_model._meta.fields if field.name !=
                    'id']
    # print(model_fields)
    try:
        # 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')
    except Exception as e:
        raise e
    # if there is no error, we return the model name
    return target_model
```

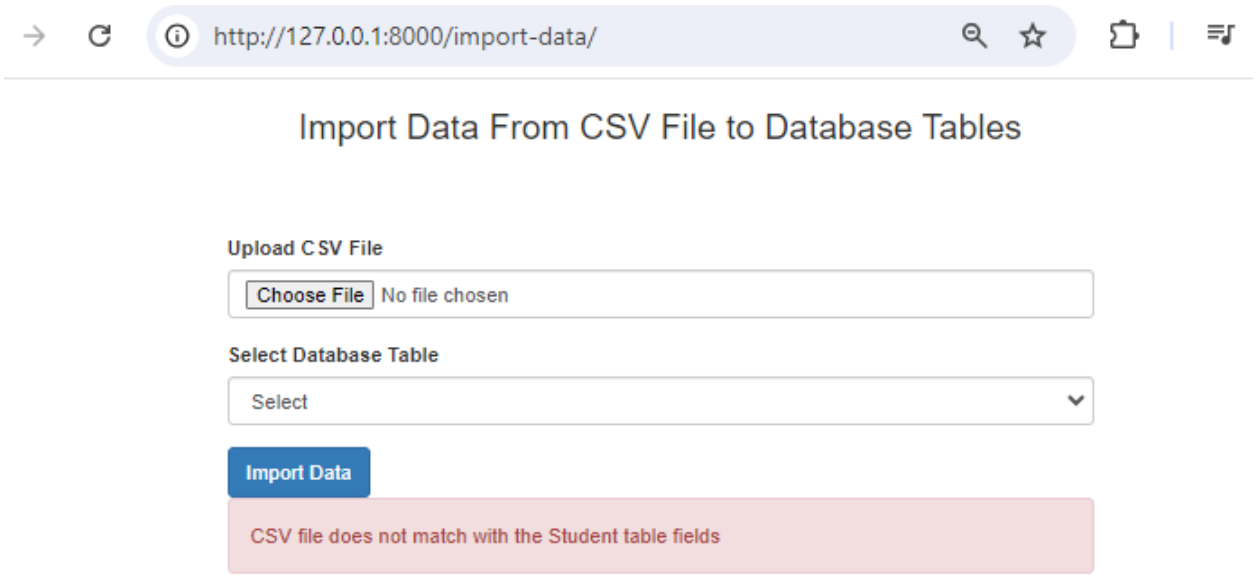
6. We also updated our DATAENTRY\VIEWS.PY to be able to call our error-handling function, check_for_csv,

```python
from django.shortcuts import render, redirect
from .utils import get_all_custom_models, check_csv_error
from uploads.models import Upload
from django.conf import settings
from django.contrib import messages
# imports our celery function
from .tasks import import_data_task
```

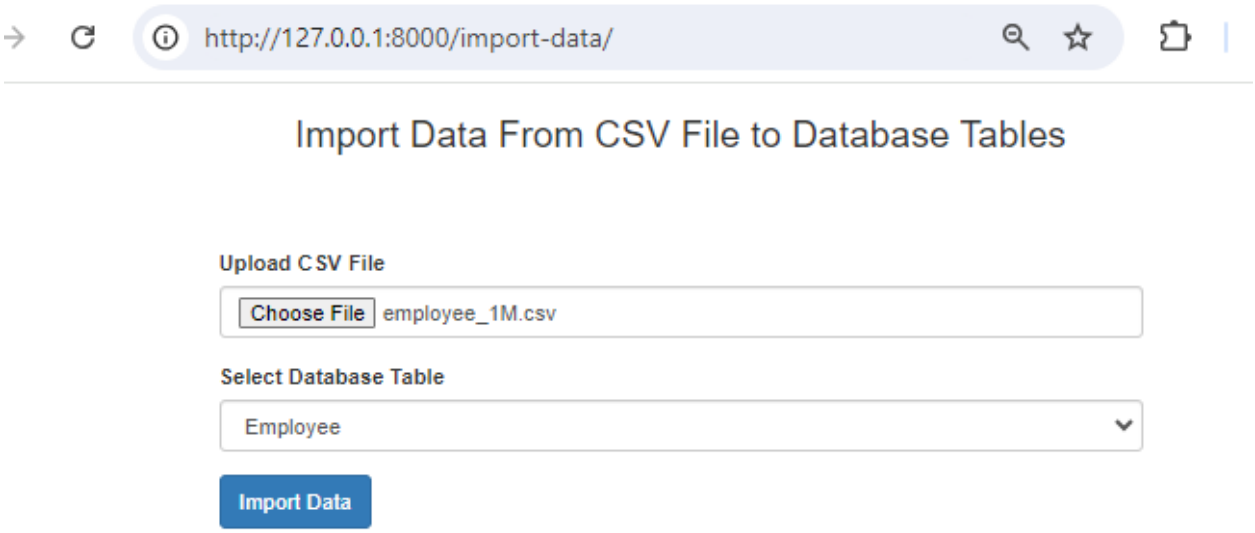


```python
def import_data(request):
    if request.method == 'POST':
        # gets the user-submitted file using its path
        file_path = request.FILES.get('file_path')
        # gets the user-submitted MODEL name
        model_name = request.POST.get('model_name')

        # 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)

        file_path = base_url + relative_path  # absolute path of the file

        # check for the CSV errors, if there is an error, we wont call our celery function for import
        try:
            # calls the check_csv_error function in dataentry\utils.property
            check_csv_error(file_path, model_name)
        except Exception as e:
            messages.error(request, str(e))
            return redirect('import_data')

        # if no error, call the Celery function for data import from dataentry\tasks.py
        import_data_task.delay(file_path, model_name)

        # show the message the user
        messages.success(
            request, 'Your data is being imported, you will be notified once this ia done.')
        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)
```

7. Run your Django-server (use $ python manage.py runserver), Redis (use $ redis-cli ping), Celery (use $ celery -A autocommontasks_main worker --loglevel=info --pool=solo) and upload Customer to Student Model, we get this user friendly message instead.



8. Now run this with your large data like EMPLOYEE.CSV to employee model. You should be able to see a user-friendly message instead of your webpage just loading for so long.



BEFORE UPLOADING:

AFTER UPLOADING: Your celery must be able to receive the task request but now the user can do other things since celery handles the time-consuming data importing in the background.