LAMPIRAN

```
# Django imports
from django.shortcuts import render, redirect
from django.core.paginator import Paginator
from django.contrib import messages
from django.contrib.auth.hashers import check_password
from django.utils.safestring import mark_safe

# Models & Forms
from .models import Dataset, AdminUser
from .forms import LoginForm, DatasetUploadForm

# Python standard library
import csv
import io
import random
import pickle
import numpy as np
import json
import pandas as pd
from openpyxl import load_workbook

# Scikit-learn
from sklearn.feature_extraction.text import
TfidfVectorizer
from sklearn.metrics import accuracy_score,
confusion_matrix
from sklearn.svm import SVC

from collections import Counter

def login(request):
    if request.method == 'POST':
        form = LoginForm(request.POST)
        if form.is_valid():
            username = form.cleaned_data['username']
            password = form.cleaned_data['password']

            try:
                admin =
AdminUser.objects.get(username=username)
```

```
                if check_password(password,
admin.password):
                    request.session['admin_user'] =
admin.username
                    return redirect('dashboard')
                else:
                    messages.error(request, 'Password
salah.')
            except AdminUser.DoesNotExist:
                messages.error(request, 'Username tidak
ditemukan.')
    else:
        form = LoginForm()

    return render(request, 'login.html', {'form': form})

def logout(request):
    request.session.flush()
    return redirect('login')

def dashboard(request):
    # Jumlah label per kelas (dari data training)
    jumlah_positif = Dataset.objects.filter(
        is_training=True
    ).filter(label__in=['positif', '1']).count()

    jumlah_negatif = Dataset.objects.filter(
        is_training=True
    ).filter(label__in=['negatif', '0']).count()

    jumlah_netral = Dataset.objects.filter(
        is_training=True
    ).filter(label__in=['netral', '-1']).count()

    # Distribusi panjang ulasan (jumlah kata)
    ulasan_list =
Dataset.objects.filter(is_training=True).values_list('full
_text', flat=True)
    word_counts = [len(text.split()) for text in
ulasan_list if text]

    # Binning panjang ulasan per 10 kata
    bins = [f"{i}-{i+9}" for i in range(0, 100, 10)]
    bin_counts = [0] * len(bins)
```

```
    for wc in word_counts:
        index = min(wc // 10, len(bins) - 1)
        bin_counts[index] += 1

    return render(request, 'dashboard.html', {
        'jumlah_positif': jumlah_positif,
        'jumlah_negatif': jumlah_negatif,
        'jumlah_netral': jumlah_netral,
        'bins': bins,
        'bin_counts': bin_counts,
    })


# def upload_dataset(request):
#     if request.method == 'POST' and 'file' in
request.FILES:
#         form = DatasetUploadForm(request.POST,
request.FILES)
#         if form.is_valid():
#             dataset_file = form.cleaned_data['file']
#             decoded_file =
dataset_file.read().decode('utf-8-sig')
#             io_string = io.StringIO(decoded_file)
#             reader = list(csv.DictReader(io_string))

#             # Shuffle & split 80/20
#             random.shuffle(reader)
#             total = len(reader)
#             split_index = int(0.8 * total)
#             training_rows = reader[:split_index]
#             testing_rows = reader[split_index:]

#             inserted = 0
#             for row in training_rows:
#                 content = row.get('full_text',
'').strip()
#                 label = normalize_label(row.get('label',
'').strip())
#                 stemming = row.get('stemming',
'').strip()
#                 if content and label is not None:
#                     Dataset.objects.create(
#                         full_text=content,
#                         preprocessed_text=stemming,
#                         label=label,
```

```
#                           is_training=True
#                       )
#                       inserted += 1

#               for row in testing_rows:
#                   content = row.get('full_text',
'').strip()
#                   label = normalize_label(row.get('label',
'').strip())
#                   stemming = row.get('stemming',
'').strip()
#                   if content and label is not None:
#                       Dataset.objects.create(
#                           full_text=content,
#                           preprocessed_text=stemming,
#                           label=label,
#                           is_training=False
#                       )
#                       inserted += 1

#               messages.success(request, f'{inserted} data
berhasil diunggah dan displit (80% training, 20%
testing).')
#               return redirect('upload_dataset')
#       else:
#           form = DatasetUploadForm()

#       # FILTER & PAGINATION
#       filter_type = request.GET.get('filter')
#       if filter_type == 'training':
#           datasets =
Dataset.objects.filter(is_training=True)
#       elif filter_type == 'testing':
#           datasets =
Dataset.objects.filter(is_training=False)
#       else:
#           datasets = Dataset.objects.all()

#       # Tambahkan urutan default
#       datasets = datasets.order_by('-id')

#       paginator = Paginator(datasets, 10)
#       page_number = request.GET.get('page')
#       page_obj = paginator.get_page(page_number)
```

```
#     return render(request, 'dataset.html', {
#         'form': form,
#         'page_obj': page_obj,
#         'filter_type': filter_type
#     })

def upload_dataset(request):
    if request.method == 'POST' and 'file' in
request.FILES:
        form = DatasetUploadForm(request.POST,
request.FILES)
        if form.is_valid():
            dataset_file = form.cleaned_data['file']
            ext = dataset_file.name.split('.')[-1].lower()

            try:
                records = []

                if ext == 'csv':
                    decoded_file =
io.StringIO(dataset_file.read().decode('utf-8-sig'))
                    reader = csv.DictReader(decoded_file)
                    for row in reader:
                        records.append({
                            'full_text':
row.get('full_text', '').strip(),
                            'label': row.get('label',
'').strip(),
                            'stemming':
row.get('stemming', '').strip()
                        })

                elif ext == 'xlsx':
                    wb =
load_workbook(filename=io.BytesIO(dataset_file.read()),
data_only=True)
                    ws = wb.active

                    # Ambil header
                    headers = [str(cell.value).strip() if
cell.value else '' for cell in
next(ws.iter_rows(min_row=1, max_row=1))]

                    # Pastikan kolom penting ada
```

```
                    required_cols = ['full_text', 'label',
'stemming']
                    for col in required_cols:
                        if col not in headers:
                            messages.error(request,
f'Kolom "{col}" tidak ditemukan di file Excel.')
                            return
redirect('upload_dataset')

                    # Ambil data
                    for row in ws.iter_rows(min_row=2,
values_only=True):
                        row_dict = dict(zip(headers,
[str(v).strip() if v is not None else '' for v in row]))
                        records.append({
                            'full_text':
row_dict.get('full_text', ''),
                            'label': row_dict.get('label',
''),
                            'stemming':
row_dict.get('stemming', '')
                        })
                else:
                    messages.error(request, 'Format file
tidak didukung. Gunakan CSV atau XLSX.')
                    return redirect('upload_dataset')

                # Shuffle & split
                random.seed(42)
                random.shuffle(records)
                total = len(records)
                split_index = int(0.8 * total)
                training_rows = records[:split_index]
                testing_rows = records[split_index:]

                inserted = 0
                for row in training_rows:
                    if row['full_text'] and row['label']:
                        Dataset.objects.create(
                            full_text=row['full_text'],

preprocessed_text=row['stemming'],

label=normalize_label(row['label']),
                            is_training=True
```

```
                        )
                        inserted += 1

                for row in testing_rows:
                    if row['full_text'] and row['label']:
                        Dataset.objects.create(
                            full_text=row['full_text'],
preprocessed_text=row['stemming'],
label=normalize_label(row['label']),
                            is_training=False
                        )
                        inserted += 1

                messages.success(
                    request,
                    f'{inserted} data berhasil diunggah
dan dibagi (80% training, 20% testing).'
                )
                return redirect('upload_dataset')

            except Exception as e:
                messages.error(request, f'Gagal memproses
file: {e}')
                return redirect('upload_dataset')
    else:
        form = DatasetUploadForm()

    # Filter & Pagination
    filter_type = request.GET.get('filter')
    if filter_type == 'training':
        datasets =
Dataset.objects.filter(is_training=True)
    elif filter_type == 'testing':
        datasets =
Dataset.objects.filter(is_training=False)
    else:
        datasets = Dataset.objects.all()

    datasets = datasets.order_by('-id')
    paginator = Paginator(datasets, 10)
    page_number = request.GET.get('page')
    page_obj = paginator.get_page(page_number)
```

```
    return render(request, 'dataset.html', {
        'form': form,
        'page_obj': page_obj,
        'filter_type': filter_type
    })

def normalize_label(label):
    """
    Mengubah label menjadi salah satu dari:
    '1' → positif
    '0' → negatif
    '-1' → netral
    """
    if label.lower() in ['positif', '1']:
        return 'positif'
    elif label.lower() in ['negatif', '0']:
        return 'negatif'
    elif label.lower() in ['netral', '-1']:
        return 'netral'
    return None


def reset_dataset(request):
    if request.method == 'POST':
        Dataset.objects.all().delete()
        messages.success(request, "Seluruh dataset
berhasil dihapus.")
    return redirect('upload_dataset')


def klasifikasi(request):
    # Ambil semua data training
    training_data = Dataset.objects.filter(
        is_training=True,
        preprocessed_text__isnull=False

).exclude(preprocessed_text__exact='').exclude(label__exac
t='').exclude(label__isnull=True)

    # Ambil semua data testing
    testing_data = Dataset.objects.filter(
        is_training=False,
        preprocessed_text__isnull=False
    ).exclude(preprocessed_text__exact='')
```

```
    X_train = [d.preprocessed_text for d in training_data]
    y_train = [normalize_label(d.label) for d in
training_data]
    X_test = [d.preprocessed_text for d in testing_data]

    if not X_train or not X_test:
        return render(request, 'klasifikasi.html', {
            'error': 'Data training atau testing belum
tersedia atau belum valid.'
        })

    # TF-IDF
    vectorizer = TfidfVectorizer()
    X_train_vec = vectorizer.fit_transform(X_train)
    X_test_vec = vectorizer.transform(X_test)

    # SVM Model
    model = SVC(kernel='linear', probability=True,
random_state=42)
    model.fit(X_train_vec, y_train)

    # Prediksi
    predicted_test = model.predict(X_test_vec)
    predicted_test_probs = model.predict_proba(X_test_vec)

    predicted_train = model.predict(X_train_vec)
    predicted_train_probs =
model.predict_proba(X_train_vec)

    # Index kelas sesuai model.classes_
    class_indices = {label: idx for idx, label in
enumerate(model.classes_)}

    # Simpan hasil prediksi (testing)
    for i, data in enumerate(testing_data):
        data.predicted_sentiment = predicted_test[i]
        data.positive_probability =
float(predicted_test_probs[i][class_indices.get('positif',
0)])
        data.neutral_probability =
float(predicted_test_probs[i][class_indices.get('netral',
0)])
        data.negative_probability =
float(predicted_test_probs[i][class_indices.get('negatif',
0)])
```

```
        data.save()

    # Simpan hasil prediksi (training)
    for i, data in enumerate(training_data):
        data.predicted_sentiment = predicted_train[i]
        data.positive_probability =
float(predicted_train_probs[i][class_indices.get('positif'
, 0)])
        data.neutral_probability =
float(predicted_train_probs[i][class_indices.get('netral',
0)])
        data.negative_probability =
float(predicted_train_probs[i][class_indices.get('negatif'
, 0)])
        data.save()

    # === Tambahan: Hitung akurasi training & testing ===
    y_test = [normalize_label(d.label) for d in
testing_data if d.label]
    akurasi_test = accuracy_score(y_test, predicted_test)
* 100 if y_test and len(y_test) == len(predicted_test)
else None

    akurasi_train = accuracy_score(y_train,
predicted_train) * 100 if y_train and len(y_train) ==
len(predicted_train) else None

    # Opsional: rata-rata keseluruhan
    akurasi_overall = None
    if akurasi_train is not None and akurasi_test is not
None:
        akurasi_overall = (akurasi_train + akurasi_test) /
2

    # Filter
    filter_type = request.GET.get('filter')
    if filter_type == 'training':
        hasil_queryset =
Dataset.objects.filter(is_training=True)
    elif filter_type == 'testing':
        hasil_queryset =
Dataset.objects.filter(is_training=False)
    else:
        hasil_queryset = Dataset.objects.all()
```

```
    hasil_queryset = hasil_queryset.order_by('-id')
    paginator = Paginator(hasil_queryset, 10)
    page = request.GET.get('page')
    hasil_klasifikasi = paginator.get_page(page)

    return render(request, 'klasifikasi.html', {
        'akurasi': akurasi_overall,       # Keseluruhan
(opsional)
        'akurasi_train': akurasi_train,   # Training
        'akurasi_test': akurasi_test,     # Testing
        'hasil_klasifikasi': hasil_klasifikasi,
        'filter_type': filter_type,
    })


def visualisasi(request):
    # === Jumlah data training & testing ===
    jumlah_training =
Dataset.objects.filter(is_training=True).count()
    jumlah_testing =
Dataset.objects.filter(is_training=False).count()

    # === Jumlah label per kelas (aktual data) ===
    jumlah_positif =
Dataset.objects.filter(label__iexact='positif').count()
    jumlah_negatif =
Dataset.objects.filter(label__iexact='negatif').count()
    jumlah_netral =
Dataset.objects.filter(label__iexact='netral').count()

    # === Ambil data training ===
    training_data = Dataset.objects.filter(
        is_training=True,
        preprocessed_text__isnull=False,
        label__isnull=False

).exclude(preprocessed_text__exact='').exclude(label__exac
t='')

    X_train = [d.preprocessed_text for d in training_data]
    y_train = [normalize_label(d.label) for d in
training_data]  # string

    # === Ambil data testing ===
    testing_data = Dataset.objects.filter(
```

```
        is_training=False,
        preprocessed_text__isnull=False,
        label__isnull=False

).exclude(preprocessed_text__exact='').exclude(label__exac
t='')

    X_test = [d.preprocessed_text for d in testing_data]
    y_test = [normalize_label(d.label) for d in
testing_data]  # string

    akurasi_model = None
    predicts_counts = {"positif": 0, "negatif": 0,
"netral": 0}
    confusion_data_test, confusion_data_train = None, None
    confusion_rows_test, confusion_rows_train = [], []

    if X_train and X_test and y_test:
        # === TF-IDF ===
        vectorizer = TfidfVectorizer()
        X_train_vec = vectorizer.fit_transform(X_train)
        X_test_vec = vectorizer.transform(X_test)

        # === Train Model ===
        model = SVC(kernel='linear', probability=True,
random_state=42)
        model.fit(X_train_vec, y_train)

        # === Prediksi ===
        prediksi_test = model.predict(X_test_vec)
        prediksi_train = model.predict(X_train_vec)

        # Akurasi
        akurasi_model = round(accuracy_score(y_test,
prediksi_test) * 100, 2)

        # === Distribusi prediksi (semua data) ===
        all_data = Dataset.objects.filter(
            preprocessed_text__isnull=False,
            label__isnull=False

).exclude(preprocessed_text__exact='').exclude(label__exac
t='')

        X_all = [d.preprocessed_text for d in all_data]
```

```
        X_all_vec = vectorizer.transform(X_all)
        predict_all = model.predict(X_all_vec)

        counter_pred = Counter(predict_all)
        predicts_counts = {
            "positif": counter_pred.get("positif", 0),
            "negatif": counter_pred.get("negatif", 0),
            "netral": counter_pred.get("netral", 0),
        }

        # === Confusion Matrix Testing ===
        labels_order = ["positif", "negatif", "netral"]
        cm_test = confusion_matrix(y_test, prediksi_test,
labels=labels_order)
        confusion_data_test = {"labels": labels_order,
"matrix": cm_test.tolist()}
        confusion_rows_test =
list(zip(confusion_data_test["labels"],
confusion_data_test["matrix"]))

        # === Confusion Matrix Training ===
        cm_train = confusion_matrix(y_train,
prediksi_train, labels=labels_order)
        confusion_data_train = {"labels": labels_order,
"matrix": cm_train.tolist()}
        confusion_rows_train =
list(zip(confusion_data_train["labels"],
confusion_data_train["matrix"]))

    # === Distribusi panjang ulasan (hanya training) ===
    ulasan_list =
Dataset.objects.filter(is_training=True).values_list('full
_text', flat=True)
    word_counts = [len(text.split()) for text in
ulasan_list if text]

    bins = [f"{i}-{i+9}" for i in range(0, 100, 10)]
    bin_counts = [0 for _ in bins]

    for wc in word_counts:
        index = min(wc // 10, len(bin_counts) - 1)
        bin_counts[index] += 1

    # === Context ===
    context = {
```

```
        'jumlah_training': jumlah_training,
        'jumlah_testing': jumlah_testing,
        'jumlah_positif': jumlah_positif,
        'jumlah_negatif': jumlah_negatif,
        'jumlah_netral': jumlah_netral,
        'akurasi_model': akurasi_model,
        'bins': bins,
        'bin_counts': bin_counts,
        'prediksi_positif': predicts_counts["positif"],
        'prediksi_negatif': predicts_counts["negatif"],
        'prediksi_netral': predicts_counts["netral"],
        'confusion_data_test':
mark_safe(json.dumps(confusion_data_test)),
        'confusion_rows_test': confusion_rows_test,
        'confusion_labels_test':
confusion_data_test["labels"] if confusion_data_test else
[],
        'confusion_data_train':
mark_safe(json.dumps(confusion_data_train)),
        'confusion_rows_train': confusion_rows_train,
        'confusion_labels_train':
confusion_data_train["labels"] if confusion_data_train
else [],
    }
    return render(request, 'visualisasi.html', context)

# Mapping label dipakai seragam (string)
def label_mapping(label):
    if str(label).lower() in ['positif', '1']:
        return "positif"
    elif str(label).lower() in ['netral', '-1']:
        return "netral"
    else:
        return "negatif"
```