LAMPIRAN

```
from flask import Flask, render_template, request,
redirect, url_for, session
from models import (
    db,
    configure_database,
    DataPenerima,
    TransformedData,
    HasilClustering,
    User,
)
from flask_login import (
    LoginManager,
    login_user,
    login_required,
    logout_user,
    current_user,
)
import plotly.express as px
import plotly.offline as pyo
from flask import flash
import pandas as pd
import io
import random
from datetime import datetime
import numpy as np
from sklearn.metrics import silhouette_score,
silhouette_samples, pairwise_distances
```

```
from sklearn.cluster import KMeans

app = Flask(__name__)
configure_database(app)

# Konfigurasi Flask-Login
login_manager = LoginManager()
login_manager.init_app(app)
login_manager.login_view = "login"

@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))

@app.route("/", methods=["GET", "POST"])
def login():
    if request.method == "POST":
        username = request.form["username"]
        password = request.form["password"]

        user = User.query.filter_by(username=username,
password=password).first()
        if user:
            login_user(user)
            return redirect(url_for("inputdata"))
        # render_template("dashboard.html") #
Mengarahkan ke dashboard
        else:
```

```
            flash("Invalid username or password",
"error")
            return render_template("login.html")
    return render_template("login.html")


# Tambahkan pengaturan untuk menangani pesan ketika
user belum login
@login_manager.unauthorized_handler
def unauthorized():
    flash("Tolong login terlebih dahulu", "info")  #
Pesan flash khusus
    return redirect(url_for("login"))  # Redirect ke
halaman login jika tidak login


@app.route("/logout")
@login_required
def logout():
    logout_user()
    return redirect(url_for("login"))


@app.route("/dashboard")
@login_required
def dashboard():
    # Total data
    total_data = DataPenerima.query.count()
    total_transformed_data =
TransformedData.query.count()


    transformed_data = TransformedData.query.all()
    _, _, _, best_cluster, _ = hitungcoba()
```

```
    data_points = np.array(
        [
            (
                data.transformed_k_rumah,
                data.transformed_gaji,
                data.transformed_pekerjaan,
            )
            for data in transformed_data
        ]
    )
    kmeans = KMeans(n_clusters=best_cluster,
random_state=0)
    kmeans.fit(data_points)
    labels = kmeans.labels_
    cluster_pie_html =
generate_cluster_member_pie(labels)



    return render_template(
        "dashboard.html",
        graph_pie = cluster_pie_html,
        total_data=total_data,
        total_transformed_data=total_transformed_data,
    )


def generate_cluster_member_pie(labels):
    # Hitung jumlah anggota per klaster
    labels = labels +1
```

```
    label_counts =
pd.Series(labels).value_counts().sort_index()
    df = pd.DataFrame({
        "Cluster": label_counts.index.astype(str),
        "Jumlah Anggota": label_counts.values
    })

    fig = px.pie(
    df,
    names="Cluster",
    values="Jumlah Anggota",
    title="Proporsi Anggota per Klaster",
    color_discrete_sequence=px.colors.qualitative.Set3,
    hole=0.3  # Buat menjadi donut chart
    )
    fig.update_traces(textinfo='percent+label',
pull=[0.05]*len(df))  # efek 'terangkat'
    return fig.to_html(full_html=False)

@app.route("/inputdata", methods=["GET", "POST"])
@login_required
def inputdata():
    if request.method == "POST":
        tanggal = request.form["tanggal"]
        nik = request.form["nik"]
        nama = request.form["nama"]
        k_rumah = request.form["k_rumah"]
        pekerjaan = request.form["pekerjaan"]
        gaji = request.form["gaji"]
```

```
        new_data = DataPenerima(
            tanggal=tanggal,
            nik=nik,
            nama=nama,
            k_rumah=k_rumah,
            pekerjaan=pekerjaan,
            gaji=gaji,
        )
        db.session.add(new_data)
        db.session.commit()

        return redirect(url_for("inputdata"))

    data_penerima = DataPenerima.query.all()
    return render_template("inputdata.html",
data_penerima=data_penerima)


@app.route("/editdata/<int:id>", methods=["GET",
"POST"])
@login_required
def editdata(id):
    data = DataPenerima.query.get_or_404(id)
    if request.method == "POST":
        tanggal = request.form["tanggal"]
        data.nik = request.form["nik"]
        data.nama = request.form["nama"]
        data.k_rumah = request.form["k_rumah"]
        data.pekerjaan = request.form["pekerjaan"]
```

```
            data.gaji = request.form["gaji"]


            db.session.commit()
            return redirect(url_for("inputdata"))


    return render_template("inputdata.html",
edit_data=data)


@app.route("/deletedata/<int:id>")
@login_required
def deletedata(id):
    data = DataPenerima.query.get_or_404(id)
    db.session.delete(data)
    db.session.commit()
    return redirect(url_for("inputdata"))


@app.route("/deletealldata")
@login_required
def deletealldata():
    try:
            db.session.query(TransformedData).delete()
            db.session.query(DataPenerima).delete()
            db.session.commit()
            return redirect(url_for("inputdata"))
    except Exception as e:
            db.session.rollback()
            return str(e)


@app.route("/uploadcsv", methods=["POST"])
```

```
@login_required
def uploadcsv():
    if "csv_file" not in request.files:
        return "No file part"

    file = request.files["csv_file"]

    if file.filename == "":
        return "No selected file"

    # Baca CSV dengan pandas
    try:
        data = pd.read_csv(file, dtype={"nik": str})

        # Bersihkan data
        data = data.dropna(
            subset=["tanggal", "nik", "nama", "kondisi
rumah", "pekerjaan", "gaji"]
        )

        # Loop dan masukkan ke database
        for index, row in data.iterrows():
            if (
                pd.isna(row["tanggal"])
                or pd.isna(row["nik"])
                or pd.isna(row["nama"])
                or pd.isna(row["kondisi rumah"])
                or pd.isna(row["pekerjaan"])
                or pd.isna(row["gaji"])
```

```
            ):
                continue
            new_item = DataPenerima(
                tanggal=row["tanggal"],
                nik=row["nik"],
                nama=row["nama"],
                k_rumah=row["kondisi rumah"],
                pekerjaan=row["pekerjaan"],
                gaji=row["gaji"],
            )
            db.session.add(new_item)
        db.session.commit()
        print(data.head())
    except Exception as e:
        return str(e)
    return redirect(url_for("inputdata"))


@app.route("/transformdata", methods=["POST"])
@login_required
def transformdata():
    data_penerima = DataPenerima.query.all()
    for data in data_penerima:
        transformed_k_rumah =
transform_k_rumah(data.k_rumah)
        transformed_gaji = transform_gaji(data.gaji)
        transformed_pekerjaan =
transform_pekerjaan(data.pekerjaan)
```

```
        existing_transformed_data =
TransformedData.query.filter_by(
            id_penerima=data.id_penerima
        ).first()
        if existing_transformed_data:
            existing_transformed_data.transformed_k_rum
ah = transformed_k_rumah
            existing_transformed_data.transformed_gaji
= transformed_gaji
            existing_transformed_data.transformed_peker
jaan = transformed_pekerjaan
        else:
            transformed_data = TransformedData(
                id_penerima=data.id_penerima,
                transformed_k_rumah=transformed_k_rumah
,
                transformed_gaji=transformed_gaji,
                transformed_pekerjaan=transformed_peker
jaan,
            )
            db.session.add(transformed_data)
    db.session.commit()
    return redirect(url_for("praproses"))


def transform_k_rumah(k_rumah):
    mapping = {"Baik": 0, "Cukup": 1, "Kurang": 2}
    return mapping.get(k_rumah, 0)


def transform_gaji(gaji):
    try:
```

```
            gaji = int(gaji)
        except (ValueError, TypeError):
            return 0
        if 0 <= gaji < 1500000:
            return 1
        elif 1500000 <= gaji <= 2000000:
            return 2
        elif 2000000 <= gaji <= 2500000:
            return 3
        elif 2500000 <= gaji <= 3000000:
            return 4
        elif gaji >= 3000000:
            return 4
        else:
            return None


def transform_pekerjaan(pekerjaan):
    mapping = {"Petani": 1, "Wiraswasta": 2,
"Pedagang": 3, "Serabutan": 4}
    return mapping.get(pekerjaan, 0)


@app.route("/praproses")
@login_required
def praproses():
    transformed_data = TransformedData.query.all()
    return render_template("praproses.html",
transformed_data=transformed_data)


@app.route("/ujicoba", methods=["GET", "POST"])
```

```
@login_required
def ujicoba():
    cluster_counts, silhouette_scores, iterations,
best_cluster, max_score = (
        hitungcoba()
    )
    # Ambil data transformed
    transformed_data = TransformedData.query.all()
    data_points = np.array(
        [
            (
                data.transformed_k_rumah,
                data.transformed_gaji,
                data.transformed_pekerjaan,
            )
            for data in transformed_data
        ]
    )
    kmeans = KMeans(n_clusters=best_cluster,
random_state=0)
    kmeans.fit(data_points)
    labels = kmeans.labels_




    # Pakai function untuk buat graph
    graph_html = generate_graph(cluster_counts,
silhouette_scores)
```

```
    # Buat grafik bar jumlah anggota per klaster
    cluster_bar_html =
generate_cluster_member_plot(labels)


    # Siapkan data results
    results = []
    for c, s, i in zip(cluster_counts,
silhouette_scores, iterations):
        results.append({"cluster": c, "silhouette":
round(s, 4), "iterations": i})


    # Render ke template
    return render_template(
        "ujicoba.html",
        graph_html=graph_html,
        cluster_bar_html=cluster_bar_html,
        results=results,
        best_cluster=best_cluster,
        max_score=round(max_score, 4),
    )


def generate_cluster_member_plot(labels):
    import plotly.express as px
    import pandas as pd
    # Hitung jumlah anggota per klaster
    labels = labels + 1
    label_counts =
pd.Series(labels).value_counts().sort_index()
    df = pd.DataFrame({
        "Cluster": label_counts.index.astype(str),
```

```
        "Jumlah Anggota": label_counts.values
    })

    # Scatter plot: cluster di sumbu X, jumlah anggota
di Y
    fig = px.scatter(
        df,
        x="Cluster",
        y="Jumlah Anggota",
        size="Jumlah Anggota",
        color="Cluster",
        title="Scatter Plot Jumlah Anggota per
Klaster",
        text="Jumlah Anggota"
    )

    fig.update_traces(textposition='top center')
    fig.update_layout(yaxis=dict(tick0=0, dtick=1))

    # label_counts =
pd.Series(labels).value_counts().sort_index()
    # df = pd.DataFrame({
    #     "Cluster": label_counts.index.astype(str),
    #     "Jumlah Anggota": label_counts.values
    # })

    # fig = px.bar(
    #     df,
    #     x="Cluster",
```

```
    #     y="Jumlah Anggota",
    #     color="Cluster",  # Pewarnaan berbeda tiap
klaster
    #     title="Jumlah Anggota per Klaster",
    #     text="Jumlah Anggota"
    # )


    # fig.update_traces(textposition='outside')
    # fig.update_layout(uniformtext_minsize=8,
uniformtext_mode='hide')


    return fig.to_html(full_html=False)


def generate_graph(cluster_counts, silhouette_scores):
    fig = px.line(
        x=cluster_counts,
        y=silhouette_scores,
        labels={"x": "Number of Clusters", "y":
"Silhouette Score"},
        title="",
        markers=True,
        width=800,
        height=400,
    )
    fig.update_layout(
        autosize=True,
        margin=dict(l=20, r=20, t=20, b=20),
    )
```

```
    graph_html = pyo.plot(fig, output_type="div")
    return graph_html


def hitungcoba():
    # Ambil data transformed
    transformed_data = TransformedData.query.all()

    data_points = np.array(
        [
            (
                data.transformed_k_rumah,
                data.transformed_gaji,
                data.transformed_pekerjaan,
            )
            for data in transformed_data
        ]
    )

    cluster_counts = []
    silhouette_scores = []
    iterations = []

    for num_clusters in range(2, 11):
        kmeans = KMeans(n_clusters=num_clusters,
random_state=0)
        labels = kmeans.fit_predict(data_points)
        score = silhouette_score(data_points, labels)
        cluster_counts.append(num_clusters)
        silhouette_scores.append(score)
```

```
        iterations.append(kmeans.n_iter_)


    # Tentukan cluster terbaik
    max_score = max(silhouette_scores)  # Skor terbaik
    best_index = silhouette_scores.index(max_score)  #
Index dari skor terbaik
    best_cluster = cluster_counts[best_index]  #
Cluster terbaik berdasarkan skor


    return cluster_counts, silhouette_scores,
iterations, best_cluster, max_score


@app.route("/2cluster")
@login_required
def duacluster():
    # Ambil data dari database
    transformed_data = TransformedData.query.all()


    if not transformed_data:
        return "No data available", 400  # Tangani jika
tidak ada data


    # Konversi data ke numpy array untuk clustering
    data_points = np.array(
        [
            (
                data.transformed_k_rumah,
                data.transformed_gaji,
                data.transformed_pekerjaan,
            )
```

```
            for data in transformed_data
        ]
    )

    # Jalankan algoritma K-Means dengan 2 cluster
    num_clusters = 2
    kmeans = KMeans(n_clusters=num_clusters,
random_state=0)
    kmeans.fit(data_points)
    labels = kmeans.labels_  # Hasil cluster untuk tiap
data

    # Hapus data sebelumnya di tabel Hasilduacluster
    db.session.query(Hasilduacluster).delete()

    # Simpan hasil clustering ke tabel Hasilduacluster
    for data, cluster in zip(transformed_data, labels):
        hasil_duacluster = Hasilduacluster(
            id_penerima=data.id_penerima,  # Ambil
id_penerima dari relasi
            cluster=int(cluster) + 1,  # Cluster ID
dimulai dari 1
        )
        db.session.add(hasil_duacluster)
    db.session.commit()

    # Hitung statistik untuk setiap cluster
    cluster_statsdua = []
    for cluster_id in range(num_clusters):
        cluster_data = data_points[
```

```
            labels == cluster_id
        ]  # Ambil data sesuai label cluster
        cluster_statsdua.append(
            {
                "cluster_id": cluster_id + 1,  # 
Cluster ID dimulai dari 1
                "min": (
                    cluster_data.min(axis=0).tolist()
                    if len(cluster_data) > 0
                    else [None, None, None]
                ),
                "max": (
                    cluster_data.max(axis=0).tolist()
                    if len(cluster_data) > 0
                    else [None, None, None]
                ),
                "mean": (
                    cluster_data.mean(axis=0).tolist()
                    if len(cluster_data) > 0
                    else [None, None, None]
                ),
                "total": len(cluster_data),
            }
        )

    # Hitung silhouette coefficient
    silhouette_avg = silhouette_score(data_points, 
labels)
```

```
    silhouette_values = silhouette_samples(data_points,
labels)


    # Simpan hasil perhitungan ke session
    session["silhouette_avg"] = silhouette_avg
    session["silhouette_values"] =
silhouette_values.tolist()
    session["cluster_assignments"] = labels.tolist()


    # Ambil data hasil clustering untuk ditampilkan
    hasil_duacluster = Hasilduacluster.query.all()


    # Kirim data dan hasil ke template
    return render_template(
        "2cluster.html",
        data=transformed_data,
        labels=labels,
        hasil_duacluster=hasil_duacluster,
        cluster_statsdua=cluster_statsdua,
        silhouette_avg=silhouette_avg,
        enumerate=enumerate,
    )


@app.route("/3cluster")
@login_required
def tigacluster():
    # Ambil data dari database
    transformed_data = TransformedData.query.all()
```

```
    # Konversi data ke numpy array
    data_points = np.array(
        [
            (
                data.transformed_k_rumah,
                data.transformed_gaji,
                data.transformed_pekerjaan,
            )
            for data in transformed_data
        ]
    )

    # Jalankan algoritma K-Means dengan 3 cluster
    kmeans = KMeans(n_clusters=3, random_state=0)
    kmeans.fit(data_points)
    labels = kmeans.labels_  # Hasil clustering

    # Hapus data hasil clustering sebelumnya
    db.session.query(Hasiltigacluster).delete()

    # Simpan hasil clustering ke database
    for data, cluster in zip(transformed_data, labels):
        hasil_tigacluster = Hasiltigacluster(
            id_penerima=data.id_penerima,  # Pastikan
atribut ini ada
            cluster=int(cluster) + 1,  # Cluster ID
dimulai dari 1
        )
        db.session.add(hasil_tigacluster)
```

```
    db.session.commit()

    # Hitung statistik untuk setiap cluster
    cluster_statstiga = []
    for cluster_id in range(3):
        cluster_data = data_points[
            labels == cluster_id
        ]  # Ambil data sesuai label cluster
        cluster_statstiga.append(
            {
                "cluster_id": cluster_id + 1,  # Cluster ID dimulai dari 1
                "min": (
                    cluster_data.min(axis=0).tolist()
                    if len(cluster_data) > 0
                    else [None, None, None]
                ),
                "max": (
                    cluster_data.max(axis=0).tolist()
                    if len(cluster_data) > 0
                    else [None, None, None]
                ),
                "mean": (
                    cluster_data.mean(axis=0).tolist()
                    if len(cluster_data) > 0
                    else [None, None, None]
                ),
                "total": len(cluster_data),
            }
```

```
        )


    # Hitung silhouette coefficient
    silhouette_avg = silhouette_score(data_points,
labels)
    silhouette_values = silhouette_samples(data_points,
labels)


    # Simpan hasil perhitungan ke session
    session["silhouette_avg"] = silhouette_avg
    session["silhouette_values"] =
silhouette_values.tolist()
    session["cluster_assignments"] = labels.tolist()


    # Ambil hasil clustering untuk ditampilkan
    hasil_tigacluster = Hasiltigacluster.query.all()


    # Kirim data ke template
    return render_template(
        "3cluster.html",
        data=transformed_data,
        labels=labels,
        hasil_tigacluster=hasil_tigacluster,
        cluster_statstiga=cluster_statstiga,
        silhouette_avg=silhouette_avg,
        enumerate=enumerate,
    )


@app.route("/pengujian", methods=["GET","POST"])
```

```
@login_required
def pengujian():
    # Ambil data dari database
    transformed_data = TransformedData.query.all()

    # Konversi data ke numpy array
    data_points = np.array(
        [
            (
                data.transformed_k_rumah,
                data.transformed_gaji,
                data.transformed_pekerjaan,
            )
            for data in transformed_data
        ]
    )

    # Jalankan algoritma K-Means dengan cluster terbaik
    _, _, _, best_cluster, _ = hitungcoba()
    kmeans = KMeans(n_clusters=best_cluster,
random_state=0)
    kmeans.fit(data_points)
    labels = kmeans.labels_  # Hasil clustering

    if len(data_points) == 0 or len(labels) == 0:
        return "Clustering belum dilakukan", 400

    # Hitung silhouette coefficient
```

```
    silhouette_avg = silhouette_score(data_points,
labels)
    silhouette_values = silhouette_samples(data_points,
labels)


    # Hitung a(i), b(i), dan s(i) untuk setiap data
point
    results = []
    for i in range(len(data_points)):
        # a(i): Rata-rata jarak ke data point lain
dalam cluster yang sama
        cluster = labels[i]
        mask = labels == cluster
        a_i = np.mean(np.linalg.norm(data_points[mask]
- data_points[i], axis=1))


        # b(i): Rata-rata jarak ke data point dalam
cluster terdekat
        b_i = np.inf
        for other_cluster in range(3):
            if other_cluster != cluster:
                mask_other = labels == other_cluster
                distance = np.mean(
                    np.linalg.norm(data_points[mask_oth
er] - data_points[i], axis=1)
                )
                if distance < b_i:
                    b_i = distance


        # s(i): Nilai silhouette untuk data point ini
        s_i = silhouette_values[i]
```

```
        # Simpan hasil perhitungan
        results.append(
            {"data_point": data_points[i].tolist(),
"a_i": a_i, "b_i": b_i, "s_i": s_i}
        )

    # Kirim hasil ke template
    return render_template(
        "pengujian.html",
silhouette_avg=silhouette_avg, results=results
    )

@app.route("/clusterterbaik", methods=["GET","POST"])
@login_required
def clusterterbaik():
    transformed_data = TransformedData.query.all()

    data_points = np.array(
        [
            (
                data.transformed_k_rumah,
                data.transformed_gaji,
                data.transformed_pekerjaan,
            )
            for data in transformed_data
        ]
    )
```

```
    # Jalankan algoritma K-Means dengan cluster terbaik
    _, _, _, best_cluster, _ = hitungcoba()
    kmeans = KMeans(n_clusters=best_cluster,
random_state=0)
    kmeans.fit(data_points)
    labels = kmeans.labels_

    # Hitung statistik untuk setiap cluster
    cluster_statstiga = []
    for cluster_id in range(3):
        cluster_data = data_points[
            labels == cluster_id
        ]  # Ambil data sesuai label cluster
        cluster_statstiga.append(
            {
                "cluster_id": cluster_id + 1,  #
Cluster ID dimulai dari 1
                "min": (
                    cluster_data.min(axis=0).tolist()
                    if len(cluster_data) > 0
                    else [None, None, None]
                ),
                "max": (
                    cluster_data.max(axis=0).tolist()
                    if len(cluster_data) > 0
                    else [None, None, None]
                ),
                "mean": (
                    cluster_data.mean(axis=0).tolist()
```

```
                    if len(cluster_data) > 0
                    else [None, None, None]
                ),
                "total": len(cluster_data),
            }
        )

    # Hitung silhouette coefficient
    silhouette_avg = silhouette_score(data_points,
labels)
    silhouette_values = silhouette_samples(data_points,
labels)

    # Simpan hasil perhitungan ke session
    session["silhouette_avg"] = silhouette_avg
    session["silhouette_values"] =
silhouette_values.tolist()
    session["cluster_assignments"] = labels.tolist()

    datapenerima = DataPenerima.query.all()
    # Gabungkan data penerima dengan label cluster

    data_penerima_with_cluster = []
    for i, penerima in enumerate(datapenerima):
        data_penerima_with_cluster.append(
        {
            "nama": penerima.nama,
            "nik": penerima.nik,
            "k_rumah": penerima.k_rumah,
```

```
                "pekerjaan": penerima.pekerjaan,
                "gaji": penerima.gaji,
                "cluster": labels[i] + 1,  # Tambah 1
supaya cluster dimulai dari 1
            }
        )

        return render_template(
            "clusterterbaik.html",
            data_penerima=data_penerima_with_cluster,
            best_cluster = best_cluster,
            data=transformed_data,
            cluster_statstiga=cluster_statstiga,
            silhouette_avg=silhouette_avg,
        )

if __name__ == "__main__":
    app.run(debug=True)
```