Lampiran 3 Sourche Code

Lampiran Halaman Login

```
import streamlit as st
import mysql.connector
from admin import admin_panel
from pemilik_toko import pemilik_toko_panel


# Koneksi ke database
conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="",  # Isi dengan password database Anda
    database="devicozmetic"
)
cursor = conn.cursor()


# Streamlit UI untuk aplikasi utama
def main():
    st.title("Aplikasi Devicozmetics")
     st.write("Aplikasi Ini Memproses Data Penjualan Devicozmetic Lalu Memprediksi Produk Yang Paling Diminati")


    # Cek status login
    if "username" not in st.session_state:
        login_form()


        # Tampilkan form registrasi jika belum login
    else:


        level = st.session_state["level"]
        if level == 'admin':
              admin_panel()  # Panggil fungsi admin_panel() dari admin.py
        elif level == 'Pemilik_toko':
```

```
                    pemilik_toko_panel()
def login_form():
    st.subheader("Login")
    username = st.text_input("Username", key="login_username")
        password = st.text_input("Password", type="password",
key="login_password")

    if st.button("Login"):
                cursor.execute(f"SELECT * FROM users WHERE
username='{username}' AND password='{password}'")
        result = cursor.fetchone()
        if result:
            level = result[3]  # Menggunakan indeks 3 karena level
akses berada di indeks 3 pada result tuple
            st.session_state["username"] = username
            st.session_state["level"] = level
            st.success(f"Logged in as {username}")
            st.write(f"Level Akses: {level}")
            redirect_to_page(level)
        else:
            st.error("Username dan Password Salah")

def registration_form():
    st.subheader("Registration")
    reg_username = st.text_input("Username", key="reg_username")
      reg_password = st.text_input("Password", type="password",
key="reg_password")
        reg_level = st.selectbox("Level Akses", ["admin",
"Pemilik_toko"], key="reg_level")

    if st.button("Register"):
        # Periksa apakah username sudah ada

                cursor.execute(f"SELECT * FROM users WHERE
username='{reg_username}'")

        if cursor.fetchone():
```

```
            st.error("Username already exists")
        else:
            # Tambahkan pengguna baru ke database
            insert_query = f"INSERT INTO users (username, password, level) VALUES ('{reg_username}', '{reg_password}', '{reg_level}')"
            cursor.execute(insert_query)
            conn.commit()
            st.success(f"User {reg_username} registered successfully with level {reg_level}")
def redirect_to_page(level):
    # Redirect halaman berdasarkan level akses
    if level == 'admin':
        admin_panel()
    elif level == 'Pemilik_toko':
        pemilik_toko_panel()
    else:
        st.error("Invalid access level")
# Panggil fungsi main
if __name__ == "__main__":
    main()
```

Lampiran Halaman Admin

```
# admin.py
import streamlit as st
from streamlit_option_menu import option_menu
import urllib.parse
import halaman_utama
import halaman_users
import halaman_prediksi
import halaman_laporan
import halaman_trainingmodel
import halaman_datapenjualan


def admin_panel():


    # Menu utama
```

```
    selected = option_menu(
        menu_title=None,
        options=["Home", "Data Users", "Data Penjualan", "Training
Model", "Data Prediksi", "Laporan", "Logout"],
        icons=["house", "graph-up-arrow", "box", "box", "file-
earmark-bar-graph", "file-earmark-check", "arrow-right-square"],
        menu_icon="cast",
        default_index=0,
        orientation="horizontal",
    )



    # Tampilkan halaman sesuai pilihan di menu utama
    if selected == "Home":
        halaman_utama.show_halaman_utama()

    elif selected == "Data Users":
        halaman_users.show_halaman_users()

    elif selected == "Data Penjualan":
        halaman_datapenjualan.show_halaman_datapenjualan()

    elif selected == "Data Prediksi":
        halaman_prediksi.show_halaman_prediksi()

    elif selected == "Laporan":
        halaman_laporan.show_halaman_laporan()

    elif selected == "Training Model":
        halaman_trainingmodel.show_halaman_trainingmodel()

    elif selected == "Logout":
        st.session_state.clear()  # Hapus semua session state


        st.experimental_rerun()
```

Lampiran Halaman Training Model

```
import streamlit as st

import pandas as pd

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import confusion_matrix,
classification_report

from sklearn.model_selection import train_test_split

import joblib

import mysql.connector

from datetime import date

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import cross_val_score,
StratifiedKFold


# Fungsi untuk menyimpan model ke dalam tabel history_model di
database MySQL

def save_model_to_history(model, model_name, conn):

    try:

        model_filename = f"{model_name}.joblib"

        joblib.dump(model, model_filename)


        with open(model_filename, 'rb') as f:

            model_binary = f.read()


        cursor = conn.cursor()


        today = date.today().isoformat()

        cursor.execute("INSERT INTO history_model (model_name,
training_date, model_binary) VALUES (%s, %s, %s)",

                       (model_name, today, model_binary))

        conn.commit()

        cursor.close()
```

```
        st.success(f"Model {model_name} berhasil disimpan ke dalam
history_model dengan tanggal {today}.")

    except Exception as e:

        st.error(f"Error: {e}")


# Fungsi untuk memprediksi jumlah penjualan dan menampilkan produk
paling diminati

def predict_and_show_top_products(svm_model, dt_model,
features_df, product_names):

    try:

        # Prediksi dengan SVM

        svm_predictions = svm_model.predict(features_df)

        product_names['SVM Predicted Sales'] = svm_predictions


        # Prediksi dengan Decision Tree

        dt_predictions = dt_model.predict(features_df)

        product_names['Decision Tree Predicted Sales'] =
dt_predictions



        # Urutkan berdasarkan prediksi dari SVM

        top_products_svm = product_names.sort_values(by='SVM
Predicted Sales', ascending=False)

        st.subheader("Produk Paling Diminati Menurut Model SVM")

        st.write(top_products_svm)


        # Urutkan berdasarkan prediksi dari Decision Tree

        top_products_dt = product_names.sort_values(by='Decision
Tree Predicted Sales', ascending=False)

        st.subheader("Produk Paling Diminati Menurut Model
Decision Tree")

        st.write(top_products_dt)

    except Exception as e:

        st.error(f"Error predicting sales: {e}")


# Fungsi untuk menampilkan confusion matrix

def plot_confusion_matrix(cm, title):
```

```
    fig, ax = plt.subplots()
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=ax)
    ax.set_title(title)
    ax.set_xlabel('Predicted')
    ax.set_ylabel('True')
    st.pyplot(fig)


# Fungsi untuk menampilkan perbandingan performa model
def plot_model_comparison(svm_report, dt_report):
    metrics = ['precision', 'recall', 'f1-score', 'support']
    svm_metrics = [svm_report['macro avg'][metric] * 100 for
metric in metrics]
    dt_metrics = [dt_report['macro avg'][metric] * 100 for metric
in metrics]


    fig, ax = plt.subplots()
    index = range(len(metrics))
    bar_width = 0.35


    bar1 = plt.bar(index, svm_metrics, bar_width, label='SVM')
    bar2 = plt.bar([i + bar_width for i in index], dt_metrics,
bar_width, label='Decision Tree')


    plt.xlabel('Metrics')
    plt.ylabel('Scores (%)')
    plt.title('Perbandingan Performa Model SVM dan Decision Tree')
    plt.xticks([i + bar_width / 2 for i in index], metrics)
    plt.legend()
    st.pyplot(fig)


    # Tampilkan tabel perbandingan
    display_comparison_table(svm_report, dt_report)


# Fungsi untuk menampilkan tabel perbandingan hasil evaluasi
def display_comparison_table(svm_report, dt_report):
```

```
    metrics = ['precision', 'recall', 'f1-score', 'accuracy']
    # Menghitung akurasi dari laporan klasifikasi
    svm_accuracy = svm_report['accuracy'] * 100
    dt_accuracy = dt_report['accuracy'] * 100

    # Mengumpulkan data untuk tabel perbandingan
    comparison_data = {
        "Metric": metrics,
        "SVM (%)": [svm_report['macro avg'][metric] * 100 if
metric != 'accuracy' else svm_accuracy for metric in metrics],
        "Decision Tree (%)": [dt_report['macro avg'][metric] * 100
if metric != 'accuracy' else dt_accuracy for metric in metrics]
    }

    # Membuat DataFrame dari data perbandingan
    comparison_df = pd.DataFrame(comparison_data)

    # Menampilkan tabel perbandingan di Streamlit
    st.subheader("Tabel Perbandingan Hasil Evaluasi Model")
    st.write(comparison_df)

# Fungsi untuk melakukan K-Fold Cross-Validation
def perform_cross_validation(model, X, y):
    cv = StratifiedKFold(n_splits=5, shuffle=True,
random_state=42)
    scores = cross_val_score(model, X, y, cv=cv,
scoring='accuracy')
    return scores

# Fungsi untuk membangun halaman training model dan menyimpan
model ke dalam history_model
def show_halaman_trainingmodel():
    st.title("Training Model")

    uploaded_file = st.file_uploader("Unggah file CSV",
type=['csv'])
```

```
    if uploaded_file is not None:
        df = pd.read_csv(uploaded_file)

        features = ['Jumlah_Penjualan', 'Harga_Awal', 'Total
Diskon', 'Rating', 'Total Harga Produk']
        target = 'Status'

        X = df[features]
        y = df[target]

        st.subheader("Deskripsi Data Training")
        st.write("Berikut adalah contoh dari beberapa baris data
yang digunakan untuk training:")
        st.write(df.head())

        st.write("Fitur yang digunakan untuk training model
adalah:")
        st.write(features)

        st.write("Target yang diprediksi adalah:")
        st.write(target)

        if st.button("Train Model"):
            try:
                X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.2, random_state=42)

                st.subheader("Contoh Sampel Data Latih dan Data
Uji")
                st.write("Sampel data latih (X_train):")
                st.write(X_train.head())
                st.write("Jumlah sampel data latih:",
len(X_train))

                st.write("Sampel data uji (X_test):")
                st.write(X_test)
```

```
                st.write("Jumlah sampel data uji:", len(X_test))


                svm_model = SVC(kernel='linear', probability=True)
                svm_model.fit(X_train, y_train)


                dt_model = DecisionTreeClassifier(random_state=42,
max_depth=5, min_samples_split=10, min_samples_leaf=5)
                dt_model.fit(X_train, y_train)


                svm_pred = svm_model.predict(X_test)
                svm_cm = confusion_matrix(y_test, svm_pred)
                svm_report = classification_report(y_test,
svm_pred, output_dict=True)


                dt_pred = dt_model.predict(X_test)
                dt_cm = confusion_matrix(y_test, dt_pred)
                dt_report = classification_report(y_test, dt_pred,
output_dict=True)



                st.subheader("Evaluasi Model SVM")
                st.write("Confusion Matrix (Matrix Kebingungan)
untuk Model SVM:")
                st.write("Confusion Matrix adalah tabel yang
menunjukkan hasil prediksi model dibandingkan dengan data
sebenarnya. Angka-angka pada tabel menunjukkan jumlah prediksi
yang benar dan salah untuk setiap kelas.")
                plot_confusion_matrix(svm_cm, "Confusion Matrix -
SVM")
                st.write("Classification Report (Laporan
Klasifikasi) untuk Model SVM:")
                st.write("Laporan klasifikasi menampilkan metrik
evaluasi seperti precision, recall, dan f1-score untuk setiap
kelas. Precision adalah rasio prediksi yang benar terhadap total
prediksi positif, recall adalah rasio prediksi yang benar terhadap
total aktual positif, dan f1-score adalah rata-rata harmonis dari
precision dan recall.")
                st.write(pd.DataFrame(svm_report).transpose())


                st.subheader("Evaluasi Model Decision Tree")
```

```
                    st.write("Confusion Matrix (Matrix Kebingungan)
untuk Model Decision Tree:")

                st.write("Confusion Matrix adalah tabel yang
menunjukkan hasil prediksi model dibandingkan dengan data
sebenarnya. Angka-angka pada tabel menunjukkan jumlah prediksi
yang benar dan salah untuk setiap kelas.")

                plot_confusion_matrix(dt_cm, "Confusion Matrix -
Decision Tree")

                st.write("Classification Report (Laporan
Klasifikasi) untuk Model Decision Tree:")

                st.write("Laporan klasifikasi menampilkan metrik
evaluasi seperti precision, recall, dan f1-score untuk setiap
kelas. Precision adalah rasio prediksi yang benar terhadap total
prediksi positif, recall adalah rasio prediksi yang benar terhadap
total aktual positif, dan f1-score adalah rata-rata harmonis dari
precision dan recall.")

                st.write(pd.DataFrame(dt_report).transpose())



                plot_model_comparison(svm_report, dt_report)



                # Tampilkan hasil cross-validation

                svm_cv_scores =
perform_cross_validation(svm_model, X, y)

                dt_cv_scores = perform_cross_validation(dt_model,
X, y)



                st.subheader("Hasil Cross-Validation")

                st.write(f"Akurasi Cross-Validation Model SVM:
{svm_cv_scores.mean() * 100:.2f}%")

                st.write(f"Akurasi Cross-Validation Model Decision
Tree: {dt_cv_scores.mean() * 100:.2f}%")



                conn = mysql.connector.connect(

                    host='localhost',

                    user='root',

                    password='',

                    database='devicozmetic'

                )

                save_model_to_history(svm_model, "SVM_Model",
conn)
```

```
                save_model_to_history(dt_model,
"Decision_Tree_Model", conn)

                conn.close()


                # Prediksi dan tampilkan produk paling diminati

                features_df = df[features]

                product_names = df[['Nama_Produk']]

                predict_and_show_top_products(svm_model, dt_model,
features_df, product_names)

            except Exception as e:

                st.error(f"Error training model: {e}")


show_halaman_trainingmodel()
```