3. *Source Code Jupyter Notebook*

```
# Membaca dataset
data = pd.read_csv('data.csv')  # Ganti 'nama_file.csv' dengan
nama file dataset Anda

# Menampilkan lima baris pertama dari dataset
print("Lima baris pertama dari dataset:")
print(data.head())

# Menampilkan jumlah entri dengan sentimen negatif, netral, dan
positif
sentimen_count = data['Human'].value_counts()

# Membuat plot diagram batang
plt.figure(figsize=(8, 6))
sentimen_count.plot(kind='bar', color=['blue', 'red', 'grey'])
plt.title('Jumlah Sentimen positif, negatif, dan netral')
plt.xlabel('Sentimen')
plt.ylabel('Jumlah')
plt.xticks(rotation=0)
plt.show()

# Menghitung jumlah sentimen positif, netral, dan negatif
positif_count = (data['Human'] == 'Positif').sum()
netral_count = (data['Human'] == 'Netral').sum()
negatif_count = (data['Human'] == 'Negatif').sum()

# Menampilkan jumlah sentimen positif, netral, dan negatif
print("Jumlah Sentimen Positif:", positif_count)
print("Jumlah Sentimen Netral:", netral_count)
print("Jumlah Sentimen Negatif:", negatif_count)

# Membaca data dari file CSV
df = pd.read_csv("data.csv")

# Proses Cleansing Data
# Membersihkan data kolom 'Human' dari karakter yang tidak
diinginkan
def clean_text(text):
    # Contoh: Menghilangkan tanda baca dan mengubah teks menjadi
huruf kecil
    text = text.replace(",", "")
    text = text.replace(".", "")
    text = text.replace("!", "")
    text = text.replace("?", "")
    text = text.replace("@", "")
    text = text.replace("#", "")
    # Lanjutkan sesuai kebutuhan membersihkan teks

    return text

# Memanggil fungsi clean_text untuk membersihkan kolom 'Human'
df['Text'] = df['Text'].apply(clean_text)

# PROSES CASE FOLDING
# Proses case folding pada kolom 'Human'
df['Text'] = df['Text'].str.lower()
```

```
# PROSES Stopword

# Inisialisasi stemmer Sastrawi
stemmer = StemmerFactory().create_stemmer()
# Fungsi untuk stemming dengan Sastrawi
def stemming(text):
    return stemmer.stem(text)

df['Text'] = df['Text'].apply(stemming)

# Inisialisasi stopword remover Sastrawi
stopword_remover                                                    =
StopWordRemoverFactory().create_stop_word_remover()

# Fungsi untuk menghapus stopword dengan Sastrawi
def remove_stopwords(text):
    return stopword_remover.remove(text)

df['Text'] = df['Text'].apply(remove_stopwords)


# PROSES Stemming

# Inisialisasi stemmer bahasa Inggris
stemmer = PorterStemmer()

# Fungsi untuk melakukan stemming pada teks
def stemming(text):
    words = text.split()  # Memisahkan teks menjadi kata-kata
    stemmed_words = [stemmer.stem(word) for word in words]
    return ' '.join(stemmed_words)

# Memanggil fungsi stemming untuk melakukan stemming pada kolom
'Human'
df['Text'] = df['Text'].apply(stemming)

# PROSES Tokenizing

# Fungsi untuk melakukan tokenisasi pada teks
def tokenize(text):
    tokens = word_tokenize(text)  # Melakukan tokenisasi kata
    return tokens

# Memanggil fungsi tokenize untuk melakukan tokenisasi pada kolom
'Human'
df['Text'] = df['Text'].apply(tokenize)


# Menampilkan hasil setelah membersihkan data
print(df['Text'])

# Simpan DataFrame ke file CSV
df.to_csv('data.csv', index=False)

# Membaca dataset dari file CSV
df = pd.read_csv('data.csv')

stopword = {
    "yang": "",
```

```
    "nya": "",
    "kok": "",
    "sih": "",
    "ga": "tidak",
    "gak": "tidak",
    "tidakk": "tidak",
    "udah": "sudah",
    "ka": "kak",
    "kakk": "kak",
    "cewe": "cewek",
    "cew": "cewek",
    "cowo": "cowok",
    "cow": "cowok",
}

def normalize_text(text, stopword):
    for word, replacement in stopword.items():
        text = text.replace(word, replacement)
    return text

df['Text']   =   df['Text'].apply(lambda   x:   normalize_text(x,
stopword))

# Membuat word cloud untuk setiap sentimen
for sentiment in df['Human'].unique():
    # Menggabungkan semua teks dalam kolom 'Text' berdasarkan
sentimen
    text = ' '.join(df[df['Human'] == sentiment]['Text'])

    # Membuat objek WordCloud
    wordcloud      =      WordCloud(width=300,      height=200,
background_color='white').generate(text)

    # Menampilkan word cloud
    plt.figure(figsize=(10, 5))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.title(f'Word Cloud untuk Sentimen {sentiment}')
    plt.axis('off')
    plt.show()

# PROSES Pembobotan Dan Pembagian Data Training Dan Data Testing
Menggunakan TF-IDF

# Membaca data dari file CSV
df = pd.read_csv("data.csv")

# Memisahkan fitur (X) dan label (y)
X = df['Text']
y = df['Human']

# Memisahkan data menjadi data pelatihan (training) dan data
pengujian (testing) dengan rasio 80:20
X_train,  X_test,  y_train,  y_test  =  train_test_split(X,  y,
test_size=0.2, random_state=42)

# Inisialisasi objek TF-IDF Vectorizer
tfidf_vectorizer = TfidfVectorizer()
```

```
# Melakukan pembelajaran (fitting) dan transformasi pada data pelatihan
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)

# Melakukan transformasi pada data pengujian
X_test_tfidf = tfidf_vectorizer.transform(X_test)

# Menampilkan dimensi dari matriks TF-IDF
print("Dimensi matriks TF-IDF untuk data pelatihan:", X_train_tfidf.shape)
print("Dimensi matriks TF-IDF untuk data pengujian:", X_test_tfidf.shape)

# Inisialisasi model regresi logistik multinomial
logreg_model = LogisticRegression(multi_class='multinomial', solver='lbfgs')

# Melatih model regresi logistik menggunakan data pelatihan dan labelnya
logreg_model.fit(X_train_tfidf, y_train)

# Memprediksi label untuk data pengujian
y_pred = logreg_model.predict(X_test_tfidf)
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Memprediksi label untuk data pengujian
y_pred = logreg_model.predict(X_test_tfidf)

# Menghitung akurasi prediksi
accuracy = accuracy_score(y_test, y_pred)
print("Akurasi model regresi logistik multinomial:", accuracy)

# Menghitung presisi prediksi
precision = precision_score(y_test, y_pred, average='weighted', zero_division=1)
print("Presisi model regresi logistik multinomial:", precision)

# Menghitung recall prediksi
recall = recall_score(y_test, y_pred, average='weighted', zero_division=1)
print("Recall model regresi logistik multinomial:", recall)

# Simpan model ke dalam file
joblib.dump(logreg_model, "model100.pkl")
joblib.dump(tfidf_vectorizer, "tfidf_vectorizer.pkl")

# Output pesan konfirmasi
print("Model berhasil disimpan dalam file 'model100.pkl'.")

# Load model yang sudah dilatih
logreg_model = joblib.load("model100.pkl")

# Fungsi untuk melakukan klasifikasi teks
def classify_text(input_text):
    # Membersihkan teks input
    cleaned_text = clean_text(input_text)
    # Mengubah teks input menjadi vektor fitur menggunakan TF-IDF
    input_vector = tfidf_vectorizer.transform([cleaned_text])
```

```
    # Melakukan prediksi menggunakan model
    predicted_label = logreg_model.predict(input_vector)[0]
    if predicted_label == 'Positif':
        return "Kalimat termasuk dalam kategori: Positif"
    elif predicted_label == 'Negatif':
        return "Kalimat termasuk dalam kategori: Negatif"
    else:
        return "Kalimat termasuk dalam kategori: Netral"

# Contoh penggunaan
input_text = "nyesel beli parfum ini kebanyakan alkohol bau nya
tidak enak"
result = classify_text(input_text)
print(result)
```

4. *Source Code Visual Studio Code*

```
import pandas as pd

import streamlit as st

import joblib

import nltk

import sqlite3

from Sastrawi.Stemmer.StemmerFactory import StemmerFactory

from nltk.corpus import stopwords

from nltk.tokenize import word_tokenize

from sklearn.feature_extraction.text import TfidfVectorizer

from datetime import datetime

from io import BytesIO

from st_aggrid import AgGrid, GridOptionsBuilder, JsCode

from st_aggrid.shared import GridUpdateMode


# Membaca model yang sudah dilatih

logreg_model = joblib.load("model100.pkl")


# Memuat TF-IDF Vectorizer yang sudah di-fit

tfidf_vectorizer = joblib.load("tfidf_vectorizer.pkl")
```

```
# Fungsi untuk membersihkan teks

def clean_text(text):

    stop_words = set(stopwords.words('indonesian'))

    factory = StemmerFactory()

    stemmer = factory.create_stemmer()

    text = text.lower()  # Case folding

    words = word_tokenize(text)  # Tokenizing

    cleaned_words = [word for word in words if word not in
stop_words]  # Stopword removal

    stemmed_words = [stemmer.stem(word) for word in
cleaned_words]  # Stemming

    return " ".join(stemmed_words)



# Fungsi untuk melakukan klasifikasi teks

def classify_text(input_text):

    # Membersihkan teks input

    cleaned_text = clean_text(input_text)

    # Mengubah teks input menjadi vektor fitur menggunakan TF-IDF

    input_vector = tfidf_vectorizer.transform([cleaned_text])

    # Melakukan Analisis menggunakan model

    predicted_label = logreg_model.predict(input_vector)[0]

    return predicted_label



# Fungsi untuk memasukkan data ke database

def insert_to_db(text, sentiment):

    conn = sqlite3.connect('db_moris.db')

    cursor = conn.cursor()

    date = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    cursor.execute('''INSERT INTO riwayat (text, sentiment, date)
VALUES (?, ?, ?)''', (text, sentiment, date))
```

```
    conn.commit()

    conn.close()


# Fungsi untuk mengambil data dari database

def fetch_data():

    conn = sqlite3.connect('db_moris.db')

    cursor = conn.cursor()

    cursor.execute('''SELECT rowid AS id, text, sentiment, date
FROM riwayat''')

    rows = cursor.fetchall()

    conn.close()

    return rows


# Fungsi untuk mengonversi DataFrame ke Excel

@st.cache_data

def convert_df_to_excel(df):

    output = BytesIO()

    with pd.ExcelWriter(output, engine='xlsxwriter') as writer:

        df.to_excel(writer, index=False, sheet_name='Sheet1')

        writer.close()  # Gunakan writer.close() untuk menyimpan
file

    processed_data = output.getvalue()

    return processed_data


# Fungsi untuk menjalankan Website

def run():

    st.title("Website Analisis Sentimen Moris Indonesia")


    tab1, tab2 = st.tabs(["Analisis Satu Kalimat", "Analisis
File"])
```

```
    with tab1:
        st.header("Masukkan kalimat untuk analisis sentimen:")
        input_text = st.text_input("Masukkan kalimat")

        if 'data' not in st.session_state:
            st.session_state['data'] = fetch_data()

        if st.button("Analisis"):
                result = classify_text(input_text)
                st.write("Hasil Analisis Sentimen:", result)
                insert_to_db(input_text, result)
                st.session_state['data'] = fetch_data()

        # Menampilkan data dari database sebagai tabel
        data = st.session_state['data']
        if data:
            df = pd.DataFrame(data, columns=['id', 'text',
'sentiment', 'date'])
            df.rename(columns={'text': 'Text', 'sentiment':
'Human'}, inplace=True)

            # Konfigurasi AgGrid
            gb = GridOptionsBuilder.from_dataframe(df)

gb.configure_pagination(paginationAutoPageSize=False,
paginationPageSize=10)
            gb.configure_default_column(resizable=True,
filterable=True, sortable=True)
            grid_options = gb.build()
```

```
            # Tampilkan tabel dengan AgGrid
            AgGrid(
                df,
                gridOptions=grid_options,
                update_mode=GridUpdateMode.SELECTION_CHANGED,
                fit_columns_on_grid_load=True,
                theme="streamlit",
            )

            # Tambahkan tombol unduh
            st.download_button(
                label="Unduh data sebagai Excel",
                data=convert_df_to_excel(df),
                file_name="data_sentimen.xlsx",
                mime="application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet"
            )
        else:
            st.write("Tidak ada data yang tersedia.")

    with tab2:
        st.header("Unggah file untuk Analisis Sentimen")
        uploaded_file = st.file_uploader("Unggah file Excel",
type=["xlsx"], key="file_uploader")

        if uploaded_file is not None:
            # Baca file Excel
            df = pd.read_excel(uploaded_file)
```

```
            # Periksa apakah kolom 'Text' ada di file yang
diunggah

            if 'Text' in df.columns:

                #    Inisialisasi    TF-IDF    Vectorizer    dan
fit_transform pada data teks

                X = df['Text'].apply(clean_text)

                X_tfidf = tfidf_vectorizer.transform(X)



                # Lakukan Analisis

                df['Human'] = logreg_model.predict(X_tfidf)



                # Tampilkan Analisis

                st.write(df)



                # Buat tombol unduh

                st.download_button(

                    label="Unduh file dengan analisis",

                    data=convert_df_to_excel(df),

                    file_name="Analisis_sentimen.xlsx",

                    mime="application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet"

                )

            else:

                st.error("File harus memiliki kolom 'Text'.")



if __name__ == "__main__":

    run()

from fpdf import FPDF

import io
```

```
def create_pdf(df):

    pdf = FPDF()

    pdf.add_page()

    pdf.set_font("Arial", size=12)



    # Header

    pdf.cell(200,   10,   txt="Produk   Terpopuler",   ln=True,
align='C')



    # Column titles

    pdf.cell(100, 10, txt="Kata", border=1)

    pdf.cell(100, 10, txt="Jumlah", border=1, ln=True)



    # Data rows

    for index, row in df.iterrows():

        pdf.cell(100, 10, txt=str(row['Kata']), border=1)

        pdf.cell(100,   10,   txt=str(row['Jumlah']),   border=1,
ln=True)



    # Save to BytesIO and return as bytes

    pdf_output = io.BytesIO()

    pdf_output.write(pdf.output(dest='S').encode('latin1'))    #
'S' is for string mode, to get PDF as bytes

    pdf_output.seek(0)  # Go to the start of the BytesIO object

    return pdf_output.getvalue()  # Return the bytes

import streamlit as st

import sqlite3



def create_connection():

    conn = sqlite3.connect('db_moris.db')
```

```
    return conn


def get_user_data(username, password):

    conn = create_connection()

    cursor = conn.cursor()

    query = "SELECT * FROM user WHERE username = ? AND password
= ?"

    cursor.execute(query, (username, password))

    result = cursor.fetchone()

    conn.close()

    return result


def    update_user_data(user_id,    new_username,    new_email,
new_password):

    conn = create_connection()

    cursor = conn.cursor()

    query = "UPDATE user SET username = ?, email = ?, password =
? WHERE id = ?"

    cursor.execute(query, (new_username, new_email, new_password,
user_id))

    conn.commit()

    conn.close()


def show_dialog(user_data):

    user_id, email, username, password = user_data

    st.session_state['edit_username'] = username

    st.session_state['edit_email'] = email

    st.session_state['edit_password'] = password


    @st.experimental_dialog("Edit User Data")
```

```
    def dialog():

        st.write("Edit Data Anda Di Bawah Ini:")

        new_email       =       st.text_input("New       Email",
value=st.session_state['edit_email'])

        new_username     =     st.text_input("New     Username",
value=st.session_state['edit_username'], disabled=True)

        new_password     =     st.text_input("New     Password",
value=st.session_state['edit_password'], type="password")



        if st.button("Save"):

            if not new_email or not new_password:

                st.error("Tolong  masukan  email  atau  password
terlebih dahulu")

            elif "@gmail.com" not in new_email:

                st.error("Email Harus Menggunakan '@gmail.com'")

            else:

                update_user_data(user_id,          new_username,
new_email, new_password)

                st.success("Edit Data Berhasil")

                st.experimental_rerun()



    dialog()


def run():

    st.title("Edit User Data")



    # Tambahkan tombol Logout

    if st.button("Logout"):

        st.session_state['logged_in'] = False

        st.experimental_rerun()
```

```
    if 'logged_in' in st.session_state and st.session_state['logged_in']:

        if 'username' not in st.session_state:

            st.error("Data pengguna tidak di temukan")

            return



        username = st.text_input("Username", value=st.session_state['username'], disabled=True)

        password = st.text_input("Password", value=st.session_state['password'], disabled=True, type="password")



        if st.button("Edit Data"):

            user_data = get_user_data(username, st.session_state['password'])

            if user_data:

                show_dialog(user_data)

            else:

                st.error("Edit data gagal")

    else:

        st.error("Anda harus login terlebih dahulu")



if __name__ == "__main__":

    run()

import pandas as pd

import streamlit as st

import nltk

import matplotlib.pyplot as plt

import plotly.express as px

from replace_words import replace_words_negatif, replace_words_netral, replace_words_positif
```

```
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory

from nltk.corpus import stopwords

from nltk.tokenize import word_tokenize

from wordcloud import WordCloud

from collections import Counter

from create_pdf import create_pdf

from fuzzywuzzy import process


# Fungsi untuk mengunduh resource NLTK secara senyap

def download_nltk_resources():

    nltk.download('stopwords', quiet=True)

    nltk.download('punkt', quiet=True)


# Panggil fungsi unduh NLTK resource di awal

download_nltk_resources()


# Fungsi untuk mengganti atau menghapus kata-kata tertentu dalam
teks

def replace_and_remove_words(text, replace_words):

    words = text.split()

    replaced_words = [replace_words.get(word, word) for word in
words]

    return " ".join(replaced_words)


# Fungsi untuk membuat word cloud

def create_word_cloud(text, title):

    wordcloud = WordCloud(width=300, height=200,
background_color='white').generate(text)

    fig, ax = plt.subplots()

    ax.imshow(wordcloud, interpolation='bilinear')
```

```
    ax.axis('off')

    ax.set_title(title)

    st.pyplot(fig)



# Fungsi untuk membersihkan teks

def clean_text(text, replace_words):

    stop_words = set(stopwords.words('indonesian'))

    factory = StemmerFactory()

    stemmer = factory.create_stemmer()

    text = text.lower()  # Case folding

    words = word_tokenize(text)  # Tokenizing

    cleaned_words = [word for word in words if word not in
stop_words]  # Stopword removal

    stemmed_words = [stemmer.stem(word) for word in
cleaned_words]  # Stemming

    replaced_text = replace_and_remove_words("
".join(stemmed_words), replace_words)  # Replace and remove words

    return replaced_text



# Fungsi untuk menghitung kemunculan kata-kata tertentu

def count_specific_words(text, words_to_count):

    word_list = text.split()

    word_counts = Counter(word_list)

    specific_word_counts = {word: word_counts[word] for word in
words_to_count}

    sorted_word_counts =
dict(sorted(specific_word_counts.items(), key=lambda item:
item[1], reverse=True))

    return sorted_word_counts



# Fungsi untuk koreksi ejaan kata
```

```
def correct_spelling(word, word_list):

    # Cari kata yang paling mirip dari word_list

    corrected_word, _ = process.extractOne(word, word_list)

    return corrected_word


# Fungsi untuk mempersiapkan words_to_count dengan kata yang benar

def prepare_words_to_count(words_to_count):

    unique_words = set(words_to_count)

    corrected_words = {word: correct_spelling(word, unique_words)
for word in unique_words}

    return list(set(corrected_words.values()))   # Menghilangkan
duplikat



# Fungsi untuk menjalankan Website

def run():

    st.title("Website Analisis Sentimen Moris Indonesia")



    st.header("Unggah file untuk Grafik dan Word Cloud Sentimen")

    uploaded_excel = st.file_uploader("Unggah file Excel",
type=["xlsx"], key="file_uploader_analysis")



    if uploaded_excel is not None:

        # Baca file Excel

        df_excel = pd.read_excel(uploaded_excel)



        # Periksa apakah kolom 'Human' ada di file yang diunggah

        if 'Human' in df_excel.columns:

            # Periksa apakah kolom 'Human' kosong

            if df_excel['Human'].isnull().all():

                st.error("Kolom 'Human' tidak boleh kosong.")
```

```
            else:

                # Hitung kemunculan setiap sentimen

                sentiment_counts                                    =
df_excel['Human'].value_counts().reset_index()

                sentiment_counts.columns = ['Sentiment', 'Count']



                # Buat diagram batang menggunakan Plotly

                color_discrete_map = {

                    'Negatif': 'red',

                    'Netral': 'gray',

                    'Positif': 'green'

                }



                fig  =  px.bar(sentiment_counts,  x='Sentiment',
y='Count', color='Sentiment',

                               labels={'Sentiment':    'Sentimen',
'Count': 'Jumlah'},

                               title='Distribusi Sentimen',

                               text='Count',


color_discrete_map=color_discrete_map)



                fig.update_traces(texttemplate='%{text}',
textposition='outside')

                fig.update_layout(uniformtext_minsize=8,
uniformtext_mode='hide')



                st.plotly_chart(fig)



                # Kata-kata yang ingin dihitung kemunculannya
```

```
                words_to_count = ["metropolis", "freedom",
"vintage", "advent", "creative", "independent", "woody",
"aquatic", "oud", "amber"]

                corrected_words_to_count =
prepare_words_to_count(words_to_count)



                all_sentiments_text_cleaned = ""



                # Hasilkan kata untuk setiap sentimen

                sentiments = df_excel['Human'].unique()

                for sentiment in sentiments:

                    sentiment_text = "
".join(df_excel[df_excel['Human'] == sentiment]['Text'])



                    # Pilih daftar kata yang akan diganti atau
dihapus berdasarkan sentimen

                    if sentiment == 'Negatif':

                        replace_words = replace_words_negatif

                    elif sentiment == 'Netral':

                        replace_words = replace_words_netral

                    elif sentiment == 'Positif':

                        replace_words = replace_words_positif

                    else:

                        replace_words = {}



                    sentiment_text_cleaned =
clean_text(sentiment_text, replace_words)

                    all_sentiments_text_cleaned += " " +
sentiment_text_cleaned

                    create_word_cloud(sentiment_text_cleaned,
f'Word Cloud untuk Sentimen {sentiment}')
```

```
                # Hitung kemunculan kata-kata tertentu dari semua
teks yang telah dibersihkan

                word_counts                                          =
count_specific_words(all_sentiments_text_cleaned,
corrected_words_to_count)

                word_counts_df                                       =
pd.DataFrame(word_counts.items(), columns=['Kata', 'Jumlah'])



                st.subheader("Kemunculan  Kata-Kata  Tertentu  di
Semua Sentimen")

                st.dataframe(word_counts_df)



                # Tombol untuk mengunduh tabel sebagai PDF

                pdf_data = create_pdf(word_counts_df)

                st.download_button(

                    label="Unduh PDF",

                    data=pdf_data,

                    file_name="word_counts.pdf",

                    mime="application/pdf"

                )

        else:

            st.error("File harus memiliki kolom 'Human'.")


if __name__ == "__main__":

    run()

import streamlit as st

import sqlite3


def create_connection():

    conn = sqlite3.connect('db_moris.db')

    return conn
```

```
def validate_login(username, password, table):
    conn = create_connection()
    cursor = conn.cursor()
    query = f"SELECT * FROM {table} WHERE username = ? AND password
= ?"
    cursor.execute(query, (username, password))
    result = cursor.fetchone()
    conn.close()
    return result


def login():
    st.title("Login")
    username = st.text_input("Username")
    password = st.text_input("Password", type="password")


    if st.button("Login"):
        if not username or not password :
            st.error("Tolong masukan username atau password
terlebih dahulu")
        elif validate_login(username, password, 'admin'):
            st.session_state['logged_in'] = True
            st.session_state['role'] = 'admin'
            st.session_state['username'] = username
            st.session_state['password'] = password
            st.experimental_rerun()
        elif validate_login(username, password, 'user'):
            st.session_state['logged_in'] = True
            st.session_state['role'] = 'user'
```

```
            st.session_state['username'] = username
            st.session_state['password'] = password
            st.experimental_rerun()
        else:
            st.error("Username atau password salah")
import streamlit as st
import app
import laporan
import login
import manage_accounts
import edit

ADMIN_PAGES = {
    "Analisis Sentimen": app,
    "Laporan Analisis Sentimen": laporan,
    "Mengelola Akun": manage_accounts,
}

USER_PAGES = {
    "Analisis Sentimen": app,
    "Laporan Analisis Sentimen": laporan,
    "Edit Akun": edit,
}

if 'logged_in' not in st.session_state:
    st.session_state['logged_in'] = False

if 'role' not in st.session_state:
```

```
    st.session_state['role'] = None


if not st.session_state['logged_in']:
    login.login()
else:
    role = st.session_state['role']


    if role == 'admin':
        PAGES = ADMIN_PAGES
    elif role == 'user':
        PAGES = USER_PAGES


    st.sidebar.title('Menu')
    selection = st.sidebar.radio("Silahkan Memilih Menu",
list(PAGES.keys()))
    page = PAGES[selection]
    page.run()
import streamlit as st
import sqlite3


# Fungsi untuk membuat koneksi ke database SQLite
def create_connection():
    conn = sqlite3.connect('db_moris.db')
    return conn


# Fungsi untuk menambahkan pengguna baru ke dalam database
def create_user(email, username, password):
    conn = create_connection()
    cursor = conn.cursor()
```

```
    cursor.execute("INSERT INTO user (email, username, password)
VALUES (?, ?, ?)", (email, username, password))

    conn.commit()

    conn.close()



# Fungsi untuk membaca semua pengguna dari database

def read_users():

    conn = create_connection()

    cursor = conn.cursor()

    cursor.execute("SELECT * FROM user")

    rows = cursor.fetchall()

    conn.close()

    return rows



# Fungsi untuk memperbarui data pengguna berdasarkan ID

def update_user(user_id, email, username, password):

    conn = create_connection()

    cursor = conn.cursor()

    cursor.execute("UPDATE user SET email = ?, username = ?,
password = ? WHERE id = ?", (email, username, password, user_id))

    conn.commit()

    conn.close()



# Fungsi untuk menghapus pengguna berdasarkan ID

def delete_user(user_id):

    conn = create_connection()

    cursor = conn.cursor()

    cursor.execute("DELETE FROM user WHERE id = ?", (user_id,))

    conn.commit()
```

```
    conn.close()


# Fungsi utama untuk mengelola akun pengguna
def manage_accounts():
    st.title("Mengelola Akun")


    # Tambahkan tombol Logout
    if st.button("Logout"):
        st.session_state['logged_in'] = False
        st.experimental_rerun()


    # Menampilkan form untuk menambah pengguna baru
    st.subheader("Tambah Pengguna Baru")
    email = st.text_input("Email")
    username = st.text_input("Username")
    password = st.text_input("Password", type="password")


    if st.button("Tambah Pengguna"):
        if not email or not username or not password:
            st.error("Tolong masukan email username atau password
terlebih dahulu")
        elif "@gmail.com" not in email:
            st.error("Email Harus Menggunakan '@gmail.com'")
        else:
            cek_users=read_users()
            usernames=[user[2]for user in cek_users]


            if username in usernames:
                st.error("Username sudah di pakai")
```

```
            else:
                create_user(email, username, password)
                st.success("Pengguna berhasil ditambahkan")

    # Menampilkan daftar pengguna
    st.subheader("Daftar Pengguna")
    users = read_users()

    # Membuat header tabel tanpa indeks
    col1, col2, col3, col4, col5, col6 = st.columns([2, 2, 2, 2,
2, 2])
    col1.write("ID")
    col2.write("Email")
    col3.write("Username")
    col4.write("Password")
    col5.write("Action")

    for user in users:
        col1, col2, col3, col4, col5, col6 = st.columns([2, 3, 2,
2, 2, 2])
        col1.write(user[0])
        col2.write(user[1])
        col3.write(user[2])
        col4.write(user[3])
        with col5:
            if st.button("Update", key=f"update_{user[0]}"):
                update_modal(user)
        with col6:
            if st.button("Delete", key=f"delete_{user[0]}"):
```

```
                delete_modal(user)


# Fungsi untuk menampilkan modal update pengguna menggunakan experimental dialog

@st.experimental_dialog("Update Pengguna")

def update_modal(user):

    user_id, email, username, password = user

    email = st.text_input("Email", value=email)

    username    =    st.text_input("Username",    value=username, disabled=True)

    password    =    st.text_input("Password",    value=password, type="password")



    if st.button("Simpan Perubahan"):

        if not email or not password:

            st.error("Tolong masukan email atau password terlebih dahulu")

        elif "@gmail.com" not in email:

            st.error("Email Harus Menggunakan '@gmail.com'")

        else:

            update_user(user_id, email, username, password)

            st.success(f"Pengguna      {username}      berhasil diperbarui")

            st.experimental_rerun()



# Fungsi untuk menampilkan modal konfirmasi hapus pengguna menggunakan experimental dialog

@st.experimental_dialog("Konfirmasi Hapus Pengguna")

def delete_modal(user):

    user_id, email, username, password = user

    st.warning(f"Apakah Anda yakin ingin menghapus pengguna {username}?")
```

```
    if st.button("Ya, Hapus"):
        delete_user(user_id)
        st.success(f"Pengguna {username} berhasil dihapus")
        st.experimental_rerun()


# Fungsi utama untuk menjalankan aplikasi
def run():
    manage_accounts()


# Menjalankan fungsi run() untuk memulai aplikasi
if __name__ == "__main__":
    run()
```