LAMPIRAN

UNIVERSITAS DARMA PERSADA
UPT PERPUSTAKAAN
Gedung Rektorat Lantai 3,
Jl.Taman Malaka Selatan, Pondok Kelapa – Jakarta Timur 13450

SURAT KETERANGAN
HASIL PENGECEKAN TURNITIN

UPT Perpustakaan Universitas Darma Persada menerangkan telah selesai melakukan pemeriksaan duplikasi/*similarity* menggunakan perangkat lunak Turnitin terhadap hasil karya sebagai berikut:

Judul : ANALISIS SENTIMEN MENGGUNAKAN METODE TRANSFORMER TERHADAP ULASAN PENGGUNA APLIKASI TOKOCRYPTO PADA GOOGLE PLAYSTORE

Penulis : **ADI DWI PRIYONO**

NIM : **2019230035**

Tgl pemeriksaan : 11 Februari 2025

Dengan hasil Tingkat Kesamaan (*similarity index*) **24%**

Demikian Surat Keterangan kami buat, untuk dipergunakan sebagaimana mestinya.

Jakarta, 11 Februari 2025

Ka.UPT Perpustakaan Unsada

Yus Rusmiyati, SS., MM

Batas maksimal similarity 30% untuk Fakultas Sastra dan Ekonomi

Batas maksimal similarity 25% untuk Fakultas Teknik, Kelautan dan Pasca Sarjana

24% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Top Sources

22% Internet sources
13% Publications
0% Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

SOURCE CODE

app.py

```
import os
import streamlit as st
from streamlit_cookies_manager import EncryptedCookieManager
from streamlit_js_eval import streamlit_js_eval
from utils import init_db, authenticate_user, register_user

# Initialize the database connection
conn = init_db()

# Initialize cookies manager
cookies = EncryptedCookieManager(
    prefix="myprefix",
    password=os.environ.get("COOKIES_PASSWORD", "mypassword"),
)

# Wait for the component to load and send us current cookies
if not cookies.ready():
    st.stop()
st.session_state.cookies = cookies

# Define pages based on user role
def check_role(cookies_role):
    if cookies_role == "user":
        return {
            f"WELCOME , {authenticated_user}": [
                st.Page("page/prediksi.py", title="Prediksi"),
                st.Page("page/report.py",        title="Analisis
Sentiment"),
                st.Page("page/analisis.py",     title="Sentiment
Analisis"),
                st.Page("page/profile.py", title="Account")
            ]
        }
    elif cookies_role == "admin":
        return {
            f"WELCOME , {authenticated_user}": [
                st.Page("page/prediksi.py", title="Prediksi"),
                st.Page("page/report.py",        title="Analisis
Sentiment"),
                st.Page("page/analisis.py",     title="Sentiment
Analisis"),
                st.Page("page/profile.py", title="Account"),
                st.Page("page/user_management.py",   title="User
Management")
            ]
        }

# Main UI
authenticated_user = cookies.get('authenticated')
authenticated_role = cookies.get('role')

if authenticated_user:
    # Display dashboard based on the user's role
    pages = check_role(authenticated_role)
```

```
    pg = st.navigation(pages)
    pg.run()

    with st.sidebar:
        if st.button("Logout", type="primary"):

            cookies['authenticated'] = ""
            cookies['role'] = ""
            cookies.save()    # Ensure the deletion is saved
immediately
            # Reload the page

streamlit_js_eval(js_expressions="parent.window.location.reload(
)")

else:
    # Show login and register tabs when the user is not
authenticated
    st.sidebar.empty()  # Hide the sidebar during login
    tab_login, tab_register = st.tabs(["Login", "Register"])

    # Login tab
    with tab_login:
        st.subheader("Login")
        login_username        =        st.text_input("Username",
key="login_username")
        login_password        =        st.text_input("Password",
type="password", key="login_password")

        if st.button("Login"):
            auth   =   authenticate_user(conn,   login_username,
login_password)
            if auth:
                # Set authentication cookies
                cookies['authenticated'] = login_username
                cookies['role'] = auth["role"]   # Default role
for the user (can be adjusted based on the role)
                cookies.save()  # Save the cookies
                st.success("Login successful!")
                # Reload the page after login

streamlit_js_eval(js_expressions="parent.window.location.reload(
)")
            else:
                st.error("Invalid username or password.")

    # Register tab
    with tab_register:
        st.subheader("Register")
        register_username       =       st.text_input("Username",
key="register_username")
        register_password       =       st.text_input("Password",
type="password", key="register_password")

        if st.button("Register"):
            if      register_user(conn,      register_username,
register_password):
                st.success("Registration successful! You can now
log in.")
```

```
                # Reload the page after registration

streamlit_js_eval(js_expressions="parent.window.location.reload(
)")
            else:
                st.error("Username already exists. Please choose
a different one.")
```

report.py

```
import streamlit as st
import pandas as pd
import numpy as np
from tqdm import tqdm
from stqdm import stqdm
import nltk
from collections import Counter
from nltk.corpus import stopwords
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
import emoji
import string
from transformers import BertTokenizer,
BertForSequenceClassification
import torch
from sklearn.metrics import classification_report,
confusion_matrix, accuracy_score

# Setup
nltk.download("stopwords")
indonesian_stopwords = stopwords.words("indonesian")
factory = StemmerFactory()
stemmer = factory.create_stemmer()

# Preprocessing Functions
def lower_case(text):
    return text.lower()

def remove_punctuation(text):
    return text.translate(str.maketrans("", "",
string.punctuation))

def remove_emoji(text):
    return emoji.replace_emoji(text, replace="")

def remove_stopwords(text):
    words = text.split()
    filtered_words = [word for word in words if word not in
indonesian_stopwords]
    return " ".join(filtered_words)

def apply_stemming(text):
    words = text.split()
    stemmed_words = [stemmer.stem(word) for word in words]
    return " ".join(stemmed_words)

# Load Model and Tokenizer
# model_dir = "models"
tokenizer =
BertTokenizer.from_pretrained('indobenchmark/indobert-base-p1')
model =
BertForSequenceClassification.from_pretrained("diiadi/sentiment-
gplay")
```

```
device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")
model.to(device)

# Function to preprocess and predict
def preprocess_data(df):
    # Lowercasing
    df["content_lower"] = [lower_case(text) for text in
stqdm(df["content"], desc="Lowercasing Text")]
    st.write("After Lowercasing:")
    st.dataframe(df[["content", "content_lower"]])

    # Removing Punctuation
    df["content_no_punctuation"] = [remove_punctuation(text) for
text in stqdm(df["content_lower"], desc="Removing Punctuation")]
    st.write("After Removing Punctuation:")
    st.dataframe(df[["content", "content_no_punctuation"]])

    # Removing Emojis
    df["content_no_emoji"] = [remove_emoji(text) for text in
stqdm(df["content_no_punctuation"], desc="Removing Emojis")]
    st.write("After Removing Emojis:")
    st.dataframe(df[["content", "content_no_emoji"]])

    # Removing Stopwords
    df["content_no_stopwords"] = [remove_stopwords(text) for
text in stqdm(df["content_no_emoji"], desc="Removing
Stopwords")]
    st.write("After Removing Stopwords:")
    st.dataframe(df[["content", "content_no_stopwords"]])

    # Stemming
    df["content_stemmed"] = [apply_stemming(text) for text in
stqdm(df["content_no_stopwords"], desc="Applying Stemming")]
    st.write("After Stemming:")
    st.dataframe(df[["content", "content_stemmed"]])

    return df

def predict_sentiment(texts):
    model.eval()
    predictions = []
    for text in stqdm(texts, desc="Predicting Sentiments"):
        inputs = tokenizer(text, return_tensors="pt",
padding="max_length", truncation=True,
max_length=128).to(device)
        outputs = model(**inputs)
        logits = outputs.logits
        pred = torch.argmax(logits, dim=1).item()
        predictions.append(pred)
    return predictions

# Fungsi untuk Generate WordCloud dengan Frekuensi, Tabel, dan
Grafik
def generate_wordcloud_with_frequency(data, sentiment, column):
    words = " ".join(data[data["predicted"] ==
sentiment][column])
    word_counts = Counter(words.split())
```

```
    # Generate WordCloud
    wordcloud = WordCloud(width=800, height=400,
background_color="white").generate_from_frequencies(word_counts)
    fig, ax = plt.subplots(figsize=(10, 5))
    ax.imshow(wordcloud, interpolation="bilinear")
    ax.set_title(f"WordCloud untuk Sentimen {sentiment} (dengan
Frekuensi)")
    st.pyplot(fig)

    # Tampilkan Tabel Frekuensi Kata
    st.subheader(f"Frekuensi Kata untuk Sentimen {sentiment}")
    most_common = word_counts.most_common(20)  # Ambil 20 kata
paling umum
    freq_df = pd.DataFrame(most_common, columns=["Kata",
"Frekuensi"])
    st.dataframe(freq_df)  # Tampilkan dalam tabel Streamlit

    # Tampilkan Grafik Frekuensi Kata (Seaborn)
    st.subheader(f"Grafik Frekuensi Kata untuk Sentimen
{sentiment}")
    plt.figure(figsize=(10, 6))  # Adjust figure size as needed
    sns.barplot(x='Kata', y='Frekuensi', data=freq_df)
    plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels
for readability
    plt.title(f"Top 20 Kata - Sentimen {sentiment}")
    st.pyplot(plt.gcf())  # Use st.pyplot to display the
Matplotlib chart

# Streamlit UI
st.title("Sentiment Analysis Playstore")

uploaded_file = st.file_uploader("Upload CSV File",
type=["csv"])

if uploaded_file:
    df = pd.read_csv(uploaded_file)
    st.write("Original Data:")
    st.dataframe(df)

    if "content" not in df.columns:
        st.error("The uploaded file must contain a 'content'
column.")
    else:
        # Preprocessing
        st.subheader("Preprocessing Data")
        df = preprocess_data(df)

        # Predict Sentiments
        st.subheader("Predicting Sentiments")
        df["predicted"] =
predict_sentiment(df["content_stemmed"])

        # Assume labels are included in dataset for evaluation
        if "label" in df.columns:
            # Map string labels to integers
            label_mapping = {"negative": 0, "positive": 1}
            y_true = df["label"].map(label_mapping)

            # Calculate Metrics
```

```
            y_pred = df["predicted"]
            accuracy = accuracy_score(y_true, y_pred)
            class_report = classification_report(y_true, y_pred,
target_names=["negative", "positive"], output_dict=True)
            confusion = confusion_matrix(y_true, y_pred)

            # Display Accuracy
            st.subheader("Accuracy Score")
            st.write(f"Accuracy: **{accuracy:.2f}**")

            # Display Classification Report
            st.subheader("Classification Report")
            st.table(pd.DataFrame(class_report).transpose())

            # Display Confusion Matrix
            st.subheader("Confusion Matrix")
            fig, ax = plt.subplots(figsize=(5, 4))
            sns.heatmap(confusion, annot=True, fmt="d",
cmap="Blues", xticklabels=["negative", "positive"],
yticklabels=["negative", "positive"])
            ax.set_xlabel("Predicted Labels")
            ax.set_ylabel("True Labels")
            st.pyplot(fig)

        # Display Pie Chart
        st.subheader("Sentiment Distribution")
        sentiment_counts = df["predicted"].value_counts()
        fig, ax = plt.subplots(figsize=(5, 5))
        sentiment_counts.plot.pie(
            autopct="%1.1f%%",
            labels=["negative", "positive"],
            ax=ax,
            ylabel=""  # Remove default ylabel
        )
        st.pyplot(fig)

        # Tampilkan WordCloud dengan Frekuensi, Tabel, dan
Grafik
        st.subheader("WordCloud dengan Frekuensi, Tabel, dan
Grafik")
        st.write("Sentimen Positif:")
        generate_wordcloud_with_frequency(df, 1,
"content_stemmed")

        st.write("Sentimen Negatif:")
        generate_wordcloud_with_frequency(df, 0,
"content_stemmed")

        # Downloadable CSV
        st.subheader("Download Results")
        csv = df.to_csv(index=False)
        st.download_button("Download CSV", data=csv,
file_name="predicted_results.csv", mime="text/csv")
```