DAFTAR LAMPIRAN

Surat Keterangan Bebas Plagiat

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 KOMENTAR INSTAGRAM DAN ULASAN GOOGLE MAPS TERHADAP UNIVERSITAS DARMA PERSADA MENGGUNAKAN METODE NAIVE BAYES
Penulis	: Erga Wanda Afriza
NIM	: **2021230016**
Tgl pemeriksaan	: 25 Juli 2025

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

Demikian Surat Keterangan kami buat, untuk dipergunakan sebagaimana mestinya.

Jakarta, 25 Juli 2025

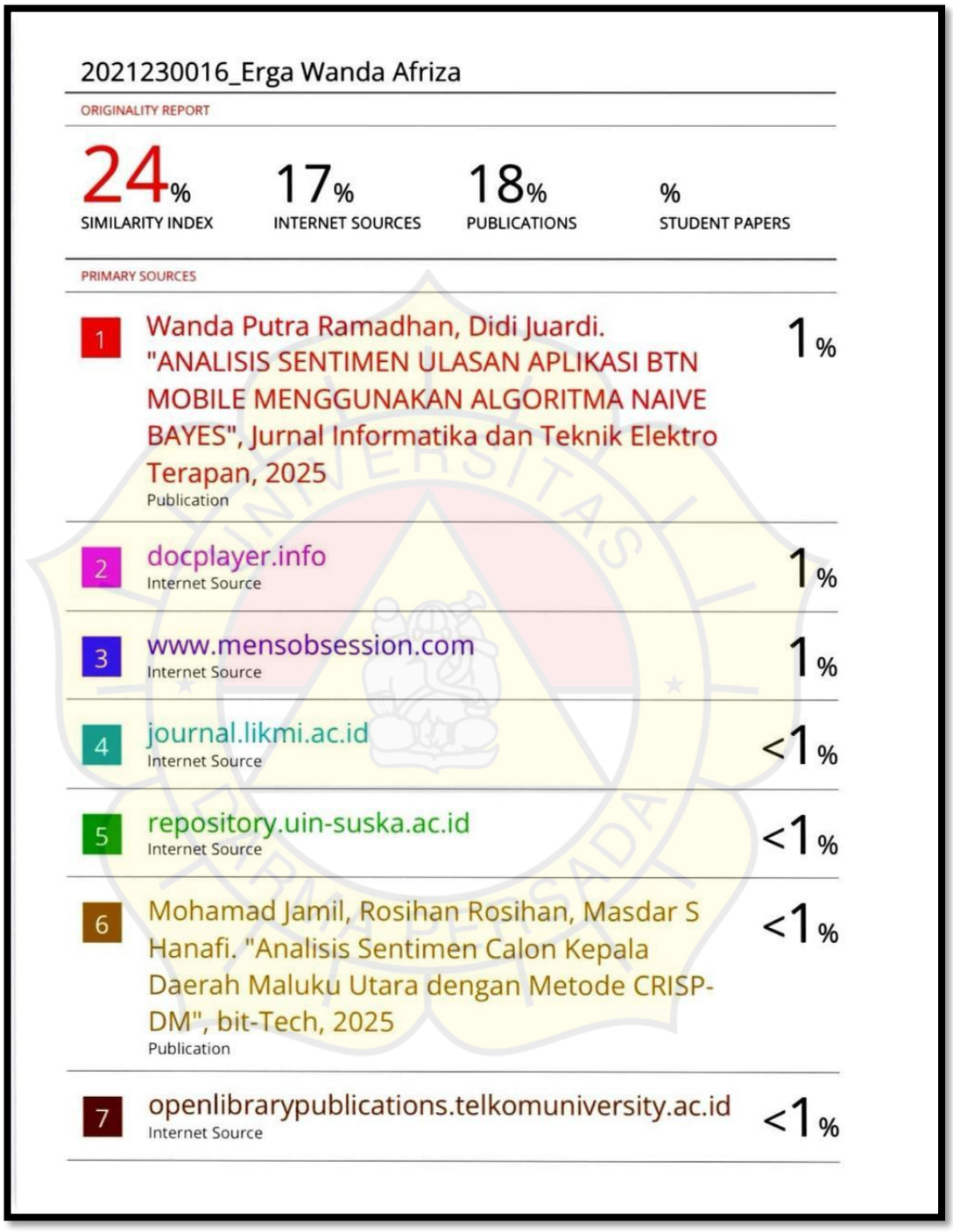
Ka.UPT Perpustakaan Unsada

Yus Rusmiyati, SS., MM

Batas maksimal similarity 30% untuk Fakultas Sastra dan Ekonomi

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

Hasil Turmitin



2021230016_Erga Wanda Afriza

ORIGINALITY REPORT

24%	17%	18%	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Wanda Putra Ramadhan, Didi Juardi. "ANALISIS SENTIMEN ULASAN APLIKASI BTN MOBILE MENGGUNAKAN ALGORITMA NAIVE BAYES", Jurnal Informatika dan Teknik Elektro Terapan, 2025 Publication	1%
2	docplayer.info Internet Source	1%
3	www.mensobsession.com Internet Source	1%
4	journal.likmi.ac.id Internet Source	<1%
5	repository.uin-suska.ac.id Internet Source	<1%
6	Mohamad Jamil, Rosihan Rosihan, Masdar S Hanafi. "Analisis Sentimen Calon Kepala Daerah Maluku Utara dengan Metode CRISP-DM", bit-Tech, 2025 Publication	<1%
7	openlibrarypublications.telkomuniversity.ac.id Internet Source	<1%

Source Code

App.py

```
1. # app.py
2. from flask import Flask, render_template, request, redirect,
   url_for, flash, session, send_file, jsonify
3. from functools import wraps
4. import os
5. import pandas as pd
6. import datetime
7. from sklearn.metrics import confusion_matrix
8. from wordcloud import WordCloud
9. import matplotlib
10.      matplotlib.use('Agg')
11.      import matplotlib.pyplot as plt
12.      import io
13.      import base64
14.      import numpy as np
15.      import pickle
16.      import uuid
17.      import re
18.      import seaborn as sns
19.      from sklearn.cluster import KMeans
20.      from sklearn.model_selection import train_test_split
21.      import nltk
22.      from nltk.corpus import stopwords
23.      import openpyxl # Tambahkan import openpyxl


24.      os.environ['OPENBLAS_NUM_THREADS'] = '1'
25.      os.environ['OMP_NUM_THREADS'] = '1'
26.      os.environ['MKL_NUM_THREADS'] = '1'
27.      from config import Config
28.      from models import db, User, Comment, AnalysisLog,
   UserInput


29.      app = Flask(__name__)
30.      app.config.from_object(Config)
31.      db.init_app(app)


32.      if not os.path.exists(os.path.join(app.root_path,
   app.config['UPLOAD_FOLDER'])):
33.      os.makedirs(os.path.join(app.root_path,
   app.config['UPLOAD_FOLDER']))
```

```
34.      try:
35.      list_stopwords = set(stopwords.words('indonesian'))
36.      print("DEBUG_TERMINAL: NLTK stopwords berhasil
   dimuat.")
37.      except LookupError:
38.      print("\n-----------------------------------------
   ----------------")
39.      print("WARNING: NLTK stopwords belum diunduh.")
40.      print("Silakan jalankan perintah ini di terminal
   server:")
41.      print(">>> source <path_ke_venv>/bin/activate")
42.      print(">>> python -c \"import nltk;
   nltk.download('stopwords')\"")
43.      print("-----------------------------------------
   --------------\n")
44.      list_stopwords = set()


45.      def get_ml_models():
46.      model_path = os.path.join(app.root_path,
   'model_naive_bayes.pkl')
47.      vectorizer_path = os.path.join(app.root_path,
   'tfidf_vectorizer.pkl')
48.      if not os.path.exists(model_path) or not
   os.path.exists(vectorizer_path):
49.      print(f"DEBUG_TERMINAL: File model atau vectorizer
   tidak ditemukan.")
50.      return None, None
51.      try:
52.      with open(model_path, 'rb') as model_file,
   open(vectorizer_path, 'rb') as vectorizer_file:
53.      loaded_model_nb = pickle.load(model_file)
54.      loaded_tfidf_vectorizer = pickle.load(vectorizer_file)
55.      print("DEBUG_TERMINAL: Model Naive Bayes dan TF-IDF
   Vectorizer berhasil dimuat.")
56.      return loaded_model_nb, loaded_tfidf_vectorizer
57.      except Exception as e:
58.      print(f"DEBUG_TERMINAL: Gagal memuat model dari file
   .pkl: {e}")
59.      return None, None


60.      def preprocess_text(text):
61.      text = str(text)
62.      text = re.sub(r'[\U0001F600-\U0001F64F\U0001F300-
   \U0001F5FF\U0001F680-\U0001F6FF\U0001F700-
   \U0001F77F\U0001F800-\U0001F8FF\U0001F900-
   \U0001F9FF\U0001FA00-\U0001FA6F\U0001FA70-
```

```
   \U0001FAFF\U00002702-\U000027B0\U000024C2-\U0001F251]+', '
   ', text)
63.      text = text.lower()
64.      text = re.sub(r'@[A-Za-z0-9_]+', ' ', text)
65.      text = re.sub(r'#(\w+)', ' ', text)
66.      text = re.sub(r'\w*\d\w*', ' ', text)
67.      text = re.sub(r'https?:\/\/\S+', ' ', text)
68.      text = re.sub(r'[^\w\s]', ' ', text)
69.      text = re.sub(r'\s+', ' ', text).strip()
70.      tokens = text.split()
71.      tokens = [word for word in tokens if word not in
   list_stopwords]
72.      return ' '.join(tokens)


73.      def login_required(f):
74.      @wraps(f)
75.      def decorated_function(*args, **kwargs):
76.      if 'user_id' not in session:
77.      flash('Anda harus login terlebih dahulu.', 'danger')
78.      return redirect(url_for('login'))
79.      return f(*args, **kwargs)
80.      return decorated_function


81.      def admin_required(f):
82.      @wraps(f)
83.      def decorated_function(*args, **kwargs):
84.      if 'user_id' not in session:
85.      flash('Anda harus login terlebih dahulu.', 'danger')
86.      return redirect(url_for('login'))
87.      user = User.query.get(session['user_id'])
88.      if user and user.role != 'admin':
89.      flash('Akses ditolak. Hanya admin yang dapat mengakses
   halaman ini.', 'danger')
90.      return redirect(url_for('landing'))
91.      return f(*args, **kwargs)
92.      return decorated_function


93.      def allowed_file(filename):
94.      return '.' in filename and \
95.      filename.rsplit('.', 1)[1].lower() in
   app.config['ALLOWED_EXTENSIONS']


96.      def log_event(description, user_id=None):
97.      with app.app_context():
```

```
98.      try:
99.      new_log = AnalysisLog(event_description=description,
   user_id=user_id)
100.     db.session.add(new_log)
101.     db.session.commit()
102.     except Exception as e:
103.     print(f"DEBUG_TERMINAL: Gagal menyimpan log: {e}")
104.     db.session.rollback()


105.     def generate_wordcloud_image(text_data):
106.     if not text_data: return None
107.     wordcloud = WordCloud(width=800, height=400,
   background_color='white',
   collocations=False).generate(text_data)
108.     img = io.BytesIO()
109.     plt.figure(figsize=(10, 5))
110.     plt.imshow(wordcloud, interpolation='bilinear')
111.     plt.axis('off')
112.     plt.savefig(img, format='png'); plt.close()
113.     img.seek(0)
114.     return base64.b64encode(img.getvalue()).decode()


115.     def analyze_and_store_comments(filepath,
   current_user_id=None):
116.     with app.app_context():
117.     def log_progress(message, progress):
118.     print(f"TASK_PROGRESS: {message} ({progress}%)")
119.     log_event(f"TASK_PROGRESS: {message} ({progress}%)",
   current_user_id)
120.     try:
121.     log_progress("Memulai proses analisis data...", 5)
122.     loaded_model_nb, loaded_tfidf_vectorizer =
   get_ml_models()
123.     if not loaded_model_nb or not loaded_tfidf_vectorizer:
124.     log_progress("Analisis gagal: Model ML belum dimuat.",
   100)
125.     return {'success': False, 'message': 'Analisis gagal:
   Model ML belum dimuat. Pastikan file model .pkl ada.'}
126.     log_progress("Membaca file data...", 10)
127.     if filepath.endswith('.xlsx'): df =
   pd.read_excel(filepath)
128.     else: df = pd.read_csv(filepath)
129.     log_progress(f"File dibaca. Jumlah baris awal:
   {len(df)}", 15)
130.     if 'text' in df.columns: df.rename(columns={'text':
   'raw_text'}, inplace=True)
131.     elif 'raw_text' not in df.columns:
```

```
132.     log_progress('Kolom "text" atau "raw_text" tidak
   ditemukan.', 100)
133.     return {'success': False, 'message': 'Kolom "text"
   atau "raw_text" tidak ditemukan di file yang diunggah.'}
134.     df['raw_text'] = df['raw_text'].astype(str)
135.     df.drop_duplicates(subset=['raw_text'], inplace=True)
136.     if 'platform' not in df.columns: df['platform'] =
   'uploaded_file'
137.     log_progress("Kolom platform disiapkan.", 25)
138.     log_progress("Timestamp disiapkan.", 30)
139.     if 'timestamp' in df.columns:
140.     df['timestamp'] = pd.to_datetime(df['timestamp'],
   errors='coerce', utc=True)
141.     df['timestamp'] =
   df['timestamp'].dt.tz_convert('Asia/Jakarta').dt.tz_localize
   (None)
142.     df = df.dropna(subset=['timestamp'])
143.     else:
144.     df['timestamp'] = datetime.datetime.now()
145.     log_progress("Memulai pra-pemrosesan teks...", 40)
146.     df['processed_text'] =
   df['raw_text'].apply(preprocess_text)
147.     df = df[df['processed_text'].str.strip() != '']
148.     log_progress(f"Pra-pemrosesan teks selesai. Baris
   setelah pra-proses: {len(df)}", 50)
149.     if df.empty:
150.     log_progress("Analisis dibatalkan: Tidak ada komentar
   tersisa setelah pra-pemrosesan.", 100)
151.     return {'success': False, 'message': "Analisis
   dibatalkan: Tidak ada komentar tersisa setelah pra-
   pemrosesan."}
152.     log_progress("Memulai prediksi sentimen...", 60)
153.     df['predicted_sentiment'] = None
154.     non_empty_mask = df['processed_text'].str.strip() !=
   ''
155.     if non_empty_mask.any():
156.     X_to_predict =
   loaded_tfidf_vectorizer.transform(df.loc[non_empty_mask,
   'processed_text'])
157.     df.loc[non_empty_mask, 'predicted_sentiment'] =
   loaded_model_nb.predict(X_to_predict)
158.     df.loc[~non_empty_mask, 'predicted_sentiment'] =
   'NETRAL'
159.     df['predicted_sentiment'] =
   df['predicted_sentiment'].astype(str).str.strip().str.upper(
   ).replace('POSSITIF', 'POSITIF')
160.     log_progress("Prediksi sentimen selesai.", 70)
161.     log_progress("Memulai proses clustering...", 75)
162.     df['cluster_id'] = np.nan
```

```
163.     target_sentiment_labels = ['POSITIF', 'NETRAL',
   'NEGATIF']
164.     for sentiment_label in target_sentiment_labels:
165.     subset_df = df[df['predicted_sentiment'] ==
   sentiment_label].copy()
166.     if not subset_df.empty and subset_df.shape[0] >= 2:
167.     subset_df_non_empty_processed =
   subset_df[subset_df['processed_text'].str.strip() !=
   ''].copy()
168.     if not subset_df_non_empty_processed.empty:
169.     subset_vec =
   loaded_tfidf_vectorizer.transform(subset_df_non_empty_proces
   sed['processed_text'])
170.     if subset_vec.shape[1] > 0 and subset_vec.sum() > 0:
171.     num_clusters = min(subset_vec.shape[0], 3)
172.     if num_clusters >= 2:
173.     try:
174.     kmeans = KMeans(n_clusters=num_clusters,
   random_state=42, n_init=10)
175.     clusters = kmeans.fit_predict(subset_vec)
176.     df.loc[subset_df_non_empty_processed.index,
   'cluster_id'] = clusters.astype(int)
177.     except Exception as cluster_e:
178.     print(f"DEBUG_TERMINAL: CRITICAL ERROR Clustering
   Sentiment '{sentiment_label}': {str(cluster_e)}")
179.     log_progress("Clustering selesai.", 85)
180.     log_progress("Menyimpan hasil analisis ke
   database...", 90)
181.     db.session.query(Comment).delete();
   db.session.commit()
182.     for index, row in df.iterrows():
183.     try:
184.     new_comment = Comment(timestamp=row['timestamp'],
   platform=row['platform'], raw_text=row['raw_text'],
   processed_text=row['processed_text'],
   predicted_sentiment=row['predicted_sentiment'],
   cluster_id=row['cluster_id'] if pd.notna(row['cluster_id'])
   else None)
185.     db.session.add(new_comment)
186.     except Exception as row_e:
187.     log_progress(f"Gagal menambahkan baris komentar ke DB:
   {str(row_e)}", 100); db.session.rollback()
188.     return {'success': False, 'message': 'Terjadi
   kesalahan saat menyimpan komentar.'}
189.     db.session.commit()
190.     log_progress(f"Analisis berhasil dilakukan dan
   {len(df)} komentar disimpan.", 100)
191.     return {'success': True, 'message': 'Analisis
   berhasil.'}
```

```
192.     except Exception as e:
193.     error_message = f'Terjadi kesalahan fatal saat
   analisis: {str(e)}.'
194.     safe_error_message = re.sub(r'[^\x00-\x7F]+', ' ',
   error_message)
195.     log_progress(safe_error_message, 100);
   db.session.rollback()
196.     return {'success': False, 'message':
   safe_error_message}


197.     @app.route('/')
198.     def index(): return redirect(url_for('landing'))
199.     @app.route('/login', methods=['GET', 'POST'])
200.     def login():
201.     if 'user_id' in session: return
   redirect(url_for('landing'))
202.     if request.method == 'POST':
203.     username, password = request.form['username'],
   request.form['password']
204.     user = User.query.filter_by(username=username).first()
205.     if user and user.check_password(password):
206.     session['user_id'], session['role'] = user.id,
   user.role; flash('Login berhasil!', 'success');
   log_event(f"User '{username}' berhasil login.", user.id)
207.     return redirect(url_for('landing'))
208.     else: flash('Username atau password salah.',
   'danger'); return redirect(url_for('login'))
209.     return render_template('login.html')


210.     @app.route('/register', methods=['GET', 'POST'])
211.     def register():
212.     if 'user_id' in session: return
   redirect(url_for('landing'))
213.     if request.method == 'POST':
214.     username, password, confirm_password =
   request.form['username'], request.form['password'],
   request.form['confirm_password']
215.     if not all([username, password, confirm_password]):
   flash('Semua kolom harus diisi.', 'danger'); return
   render_template('register.html', username=username)
216.     if password != confirm_password: flash('Konfirmasi
   password tidak cocok.', 'danger'); return
   render_template('register.html', username=username)
217.     if User.query.filter_by(username=username).first():
   flash('Username sudah terdaftar.', 'danger'); return
   render_template('register.html', username=username)
218.     try:
```

```
219.     new_user = User(username=username, role='user');
   new_user.set_password(password); db.session.add(new_user);
   db.session.commit()
220.     flash('Registrasi berhasil! Silakan login.',
   'success'); log_event(f"User baru '{username}' berhasil
   mendaftar.", new_user.id); return redirect(url_for('login'))
221.     except Exception as e:
222.     db.session.rollback(); flash(f'Terjadi kesalahan:
   {str(e)}.', 'danger'); log_event(f"Error registrasi user
   '{username}': {str(e)}"); return
   render_template('register.html', username=username)
223.     return render_template('register.html')


224.     @app.route('/logout')
225.     @login_required
226.     def logout():
227.     user_id = session.pop('user_id', None); username =
   User.query.get(user_id).username if user_id else "unknown"
228.     log_event(f"User '{username}' berhasil logout.",
   user_id); session.pop('role', None); flash('Anda telah
   logout.', 'info')
229.     return redirect(url_for('login'))


230.     @app.route('/landing')
231.     @login_required
232.     def landing(): return render_template('landing.html',
   user_role=session.get('role'))


233.     @app.route('/upload', methods=['GET', 'POST'])
234.     @admin_required
235.     def upload_file():
236.     if request.method == 'POST':
237.     if 'file' not in request.files or not
   request.files['file']: flash('Tidak ada file yang
   diterima.', 'danger'); return redirect(request.url)
238.     file = request.files['file']
239.     if file.filename == '' or not
   allowed_file(file.filename): flash('Tipe file tidak
   diizinkan atau tidak ada file terpilih.', 'danger'); return
   redirect(request.url)
240.     filename =
   datetime.datetime.now().strftime('%Y%m%d%H%M%S') + '_' +
   file.filename
241.     filepath = os.path.join(app.root_path,
   app.config['UPLOAD_FOLDER'], filename)
```

```
242.     try:
243.     file.save(filepath)
244.     result = analyze_and_store_comments(filepath,
   session.get('user_id'))
245.     if result['success']: flash('Analisis berhasil
   dilakukan!', 'success'); return redirect(url_for('results'))
246.     else: flash(result['message'], 'danger'); return
   redirect(url_for('upload_file'))
247.     except Exception as e: flash(f"Terjadi kesalahan saat
   menyimpan file atau analisis: {str(e)}", 'danger'); return
   redirect(url_for('upload_file'))
248.     user_inputs_data =
   UserInput.query.order_by(UserInput.input_date.desc()).all()
249.     return render_template('upload.html',
   user_inputs=user_inputs_data)


250.     @app.route('/instant_analysis', methods=['GET',
   'POST'])
251.     def instant_analysis():
252.     if request.method == 'POST':
253.     user_name, input_text =
   request.form.get('user_name_input', 'Anonim'),
   request.form['user_comment_input']
254.     if not input_text.strip(): flash('Komentar tidak boleh
   kosong.', 'danger'); return redirect(request.url)
255.     try:
256.     loaded_model_nb, loaded_tfidf_vectorizer =
   get_ml_models()
257.     if not loaded_model_nb or not loaded_tfidf_vectorizer:
   flash('Analisis gagal: Model ML belum dimuat.', 'danger');
   log_event("Analisis instan gagal: Model ML belum dimuat.");
   return redirect(request.url)
258.     processed_text = preprocess_text(input_text)
259.     if not processed_text.strip(): predicted_sentiment =
   'NETRAL'
260.     else: predicted_sentiment =
   loaded_model_nb.predict(loaded_tfidf_vectorizer.transform([p
   rocessed_text]))[0]
261.     predicted_sentiment =
   predicted_sentiment.upper().replace('POSSITIF',
   'POSITIF').strip()
262.     new_user_input = UserInput(user_name=user_name,
   input_text=input_text, processed_text=processed_text,
   predicted_sentiment=predicted_sentiment)
263.     db.session.add(new_user_input); db.session.commit()
264.     flash(f'Komentar berhasil dianalisis: Sentimen Anda
   adalah **{predicted_sentiment.capitalize()}**! Komentar Anda
   telah direkam.', 'success'); log_event(f"Komentar instan
```

```
   diterima: '{input_text[:50]}...' -> {predicted_sentiment}");
   return redirect(url_for('instant_analysis'))
265.     except Exception as e:
266.     db.session.rollback(); flash(f'Terjadi kesalahan saat
   menganalisis komentar: {str(e)}.', 'danger');
   log_event(f"Error menganalisis komentar instan: {str(e)}");
   return redirect(url_for('instant_analysis'))
267.     return render_template('instant_analysis.html')
```