Lampiran 3 Source Code

 # Halaman Admin

```
import streamlit as st
import pandas as pd
from statsmodels.tsa.arima.model import ARIMA
from mlxtend.frequent_patterns import fpgrowth,
association_rules
import joblib
from utils import save_history, save_package,
load_model, save_model, fill_missing_dates,
save_forecasting_history, save_prediction_history
from config import
export_transaksi_produk_to_csv,get_db_connection,create
_package_table,
create_history_table,export_transaksi_produk_to_excel,
fetch_all_packages, fetch_all_history, delete_package,
delete_history, update_package,
create_forecasting_history_table,
create_prediction_history_table,
fetch_forecasting_history,
fetch_prediction_history,insert_transaksi_produk,fetch_
all_transaksi_produk,delete_transaksi_produk,update_tra
nsaksi_produk
import io
import xlsxwriter
import os
from mysql.connector import Error
from datetime import datetime
from sklearn.metrics import mean_absolute_error
import plotly.express as px
import matplotlib.pyplot as plt
from PIL import Image

def predict_and_evaluate(model_file, steps,
actual_data):
    model = load_model(model_file)
    forecast = model.forecast(steps)
    mae = (abs(forecast - actual_data)).mean()
    return forecast, mae

def plot_forecast(train_data, actual_data,
forecast_dates, forecast):
    plt.figure(figsize=(14, 7))
    plt.plot(train_data['Tanggal_Transaksi'],
train_data['Jumlah_Stok'], label='Data Train')
```

```
    plt.plot(forecast_dates,                    forecast,
label='Prediksi')
    plt.plot(forecast_dates,  actual_data,  label='Data
Aktual')
    plt.fill_between(forecast_dates,       actual_data,
forecast, color='gray', alpha=0.2)
    plt.title('Prediksi Stok Produk')
    plt.xlabel('Tanggal')
    plt.ylabel('Jumlah Stok')
    plt.legend()
    plt.grid()
    return plt

def plot_train_data(data, date_col, value_col):
    plt.figure(figsize=(10, 6))
    plt.plot(data[date_col],           data[value_col],
label='Jumlah Stok')
    plt.title('Data Train Jumlah Stok Produk')
    plt.xlabel('Tanggal')
    plt.ylabel('Jumlah Stok')
    plt.legend()
    plt.grid()
    return plt
# Function to save history to the database
def save_history(file_name):
    conn = get_db_connection()
    cursor = conn.cursor()
    query = "INSERT INTO history (file_name) VALUES
(%s)"
    cursor.execute(query, (file_name,))
    conn.commit()
    cursor.close()
    conn.close()

# Function to save package to the database
def save_package(package_name, items, discount):
    conn = get_db_connection()
    cursor = conn.cursor()
    query = "INSERT INTO Package (package_name, items,
discount) VALUES (%s, %s, %s)"
    cursor.execute(query,     (package_name,     items,
discount))
    conn.commit()
    cursor.close()
    conn.close()

def load_data():
    packages_df = fetch_all_packages()
```

```
    prediction_history_df = fetch_prediction_history()
    return packages_df, prediction_history_df

# Function to fetch all users from the database
def fetch_all_users():
    try:
        connection = get_db_connection()
        if connection:
            cursor = connection.cursor(dictionary=True)
            cursor.execute("SELECT * FROM users")
            users = cursor.fetchall()
            cursor.close()
            connection.close()
            return users
    except Error as e:
        st.error(f"Error fetching users: {e}")
        return []

# Function to insert a new user into the database
def insert_user(username, password, name, level_akses):
    try:
        connection = get_db_connection()
        if connection:
            cursor = connection.cursor()
            current_timestamp                                    =
datetime.now().strftime('%Y-%m-%d %H:%M:%S')
            cursor.execute("INSERT      INTO      users
(username,  password,  name,  level_akses,  timestamp)
VALUES (%s, %s, %s, %s, %s)",
                           (username,  password,  name,
level_akses, current_timestamp))
            connection.commit()
            st.success("User added successfully!")
            cursor.close()
            connection.close()
    except Error as e:
        st.error(f"Error inserting user: {e}")

# Function to update an existing user in the database
def  update_user(user_id,  username,  password,  name,
level_akses):
    try:
        connection = get_db_connection()
        if connection:
            cursor = connection.cursor()
            cursor.execute("UPDATE      users      SET
username=%s, password=%s, name=%s, level_akses=%s WHERE
id=%s",
```

```
 (username, password, name, level_akses, user_id))
            connection.commit()
            st.success("User updated successfully!")
            cursor.close()
            connection.close()
    except Error as e:
        st.error(f"Error updating user: {e}")

# Function to delete a user from the database
def delete_user(user_id):
    try:
        connection = get_db_connection()
        if connection:
            cursor = connection.cursor()
            cursor.execute("DELETE FROM users WHERE
id=%s", (user_id,))
            connection.commit()
            st.success("User deleted successfully!")
            cursor.close()
            connection.close()
    except Error as e:
        st.error(f"Error deleting user: {e}")

# Streamlit UI for CRUD operations
def users_crud():
    # Initialize session_state for selected_user_id
    if 'selected_user_id' not in st.session_state:
        st.session_state.selected_user_id = None

    with st.form("User Form"):
        st.header("Manage Users")

        # Display current users
        users_df = pd.DataFrame(fetch_all_users())
        st.dataframe(users_df)

        # Form inputs
        username = st.text_input("Username")
        password = st.text_input("Password",
type="password")
        name = st.text_input("Name")
        level_akses = st.radio("Access Level", ('admin',
'PemilikToko'))

        add_update_button =
st.form_submit_button("Add/Update User")

        if add_update_button:
```

```
            if username and password and name:
                if st.session_state.selected_user_id is
None:
                    insert_user(username,     password,
name, level_akses)
                else:

update_user(st.session_state.selected_user_id,
username, password, name, level_akses)
            else:
                st.warning("Please fill in all fields.")

    # Delete functionality
    st.header("Delete Users")
    selected_rows  =  st.multiselect("Select  users  to
delete", users_df['id'].tolist())

    if st.button("Delete Selected"):
        for user_id in selected_rows:
            delete_user(user_id)

# Initialize session state variables
if 'model_loaded' not in st.session_state:
    st.session_state['model_loaded'] = False
    st.session_state['basket_sets'] = None
    st.session_state['frequent_itemsets'] = None
    st.session_state['rules'] = None
# Load the image
logo_img = Image.open('image/parfum.png')
# Ensure the package and history tables exist
create_package_table()
create_history_table()
st.image(logo_img, width=200)
st.title("Kayyasah Parfum ")
```