```python
import streamlit as st
import pandas as pd
from auth import get_all_users, get_all_admin_users, delete_user
from data_generator import generate_energy_data
from model import prepare_data, train_model, predict_future, load_model, save_model
import matplotlib.pyplot as plt
import os

def admin_dashboard():
    """Admin dashboard with user management and system overview"""
    st.set_page_config(page_title="EcoWatt Admin Dashboard", page_icon="вљЎ")

    if not st.session_state.get('logged_in') or st.session_state.get('user_type') != 'admin':
        st.error("Access denied. Admin login required.")
        return

    st.title("вљЎ EcoWatt Admin Dashboard")
    st.markdown(f"Welcome, {st.session_state.user['full_name']}!")

    # Sidebar navigation
    page = st.sidebar.selectbox("Navigation", [
        "Dashboard Overview",
        "User Management",
        "System Data",
        "Analytics"
    ])

    if page == "Dashboard Overview":
        dashboard_overview()
    elif page == "User Management":
        user_management()
    elif page == "System Data":
        system_data_management()
    elif page == "Analytics":
        analytics_section()

    # Logout button
    if st.sidebar.button("Logout"):
        for key in list(st.session_state.keys()):
            del st.session_state[key]
        st.rerun()

def dashboard_overview():
    """Main dashboard overview"""
    st.header("Dashboard Overview")

    # System statistics
    col1, col2, col3, col4 = st.columns(4)

    users_df = get_all_users()
    admin_df = get_all_admin_users()

    with col1:
        st.metric("Total Users", len(users_df))
    with col2:
        st.metric("Total Admins", len(admin_df))
    with col3:
        st.metric("System Status", "Active")
    with col4:
        data_exists = os.path.exists('energy_data.csv')
        st.metric("Data Available", "Yes" if data_exists else "No")

    # Recent activity
    st.subheader("Recent Activity")
    st.info("System initialized and running normally")
    st.info(f"Last login: {st.session_state.user.get('username', 'Unknown')}")

def user_management():
    """User management section"""
    st.header("User Management")

    tab1, tab2 = st.tabs(["Regular Users", "Admin Users"])

    with tab1:
        manage_users("user")
    with tab2:
        manage_users("admin")

def manage_users(user_type):
    """Manage users of specific type"""
    if user_type == "admin":
        users_df = get_all_admin_users()
        title = "Admin Users"
    else:
        users_df = get_all_users()
        title = "Regular Users"

    st.subheader(title)

    if not users_df.empty:
        # Display users table
        display_df = users_df[['username', 'email', 'full_name', 'created_at']].copy()
        display_df['created_at'] = pd.to_datetime(display_df['created_at']).dt.strftime('%Y-%m-%d %H:%M')
        st.dataframe(display_df)

        # Delete user section
        st.subheader("Delete User")
        usernames = users_df['username'].tolist()
        username_to_delete = st.selectbox("Select user to delete", usernames, key=f"delete_{user_type}")

        if st.button(f"Delete {user_type.title()}", key=f"btn_delete_{user_type}"):
            if username_to_delete == st.session_state.user['username']:
                st.error("Cannot delete your own account")
            else:
                success, message = delete_user(username_to_delete, user_type)
                if success:
                    st.success(message)
                    st.rerun()
                else:
                    st.error(message)
    else:
        st.info(f"No {user_type} users found")

def system_data_management():
    """System data management"""
    st.header("System Data Management")

    # Data generation
    st.subheader("Generate System Data")
    col1, col2 = st.columns(2)

    with col1:
        periods = st.slider("Number of days", 365, 365*5, 730)
        if st.button("Generate New Data"):
            with st.spinner("Generating data..."):
                data = generate_energy_data(periods=periods)
                data.to_csv('energy_data.csv', index=False)
                st.success("System data generated successfully!")

    with col2:
        if os.path.exists('energy_data.csv'):
            data = pd.read_csv('energy_data.csv')
            st.metric("Current Data Points", len(data))
            st.metric("Date Range", f"{data['date'].min()} to {data['date'].max()}")

    # Model training
    st.subheader("Model Training")
    if os.path.exists('energy_data.csv'):
        if st.button("Train System Model"):
            with st.spinner("Training model..."):
                data = pd.read_csv('energy_data.csv')
                data['date'] = pd.to_datetime(data['date'])
                X, y = prepare_data(data)
                model = train_model(X, y)
                save_model(model)
                st.success("Model trained successfully!")
    else:
        st.warning("No data available. Generate data first.")

def analytics_section():
    """Analytics and insights"""
    st.header("System Analytics")

    if os.path.exists('energy_data.csv'):
        data = pd.read_csv('energy_data.csv')
        data['date'] = pd.to_datetime(data['date'])

        # Basic analytics
        st.subheader("Data Analytics")

        col1, col2, col3 = st.columns(3)
        with col1:
            st.metric("Total Records", len(data))
        with col2:
            st.metric("Avg Consumption", f"{data['consumption_kwh'].mean():.1f} kWh")
        with col3:
            st.metric("Date Range", f"{len(data)} days")

        # Consumption chart
        st.subheader("Consumption Trends")
        fig, ax = plt.subplots(figsize=(10, 4))
        ax.plot(data['date'], data['consumption_kwh'])
        ax.set_xlabel('Date')
        ax.set_ylabel('Consumption (kWh)')
        ax.set_title('System Energy Consumption Trends')
        st.pyplot(fig)

        # Model performance if available
        if os.path.exists('energy_model.pkl'):
            st.subheader("Model Performance")
            model = load_model()
            if model:
                X, y = prepare_data(data)
                if len(X) > 0:
                    split_idx = int(len(X) * 0.8)
                    X_test, y_test = X[split_idx:], y[split_idx:]
                    if len(X_test) > 0:
                        y_pred = model.predict(X_test)
                        mse = ((y_test - y_pred) ** 2).mean()
                        st.metric("Model MSE", f"{mse:.2f}")
                    else:
                        st.info("Not enough data for performance evaluation")
                else:
                    st.info("Model loaded but no test data available")
            else:
                st.warning("Could not load model")
        else:
            st.info("No trained model available")
    else:
        st.warning("No system data available for analytics")

if __name__ == "__main__":
    admin_dashboard()
```

Python Code

```python
import pandas as pd
import os
import hashlib
import streamlit as st
from datetime import datetime

# File paths
USERS_FILE = 'users.xlsx'
ADMIN_USERS_FILE = 'admin_users.xlsx'

def hash_password(password):
    """Hash password using SHA-256"""
    return hashlib.sha256(password.encode()).hexdigest()

def init_user_files():
    """Initialize user data files if they don't exist"""
    # Regular users file
    if not os.path.exists(USERS_FILE):
        users_df = pd.DataFrame(columns=['username', 'password', 'email', 'full_nam
e', 'role', 'created_at'])
        # Add sample users
        sample_users = [
            {
                'username': 'user1',
                'password': hash_password('password123'),
                'email': 'user1@example.com',
                'full_name': 'John Doe',
                'role': 'user',
                'created_at': datetime.now().strftime('%Y-%m-%d %H:%M:%S')
            },
            {
                'username': 'user2',
                'password': hash_password('password123'),
                'email': 'user2@example.com',
                'full_name': 'Jane Smith',
                'role': 'user',
                'created_at': datetime.now().strftime('%Y-%m-%d %H:%M:%S')
            }
        ]
        users_df = pd.DataFrame(sample_users)
        users_df.to_excel(USERS_FILE, index=False)

    # Admin users file
    if not os.path.exists(ADMIN_USERS_FILE):
        admin_df = pd.DataFrame(columns=['username', 'password', 'email', 'full_nam
e', 'role', 'created_at'])
        # Add sample admin
        sample_admin = [{
            'username': 'admin',
            'password': hash_password('admin123'),
            'email': 'admin@ecowatt.com',
            'full_name': 'System Administrator',
                'role': 'admin',
                'created_at': datetime.now().strftime('%Y-%m-%d %H:%M:%S')
            }]
            admin_df = pd.DataFrame(sample_admin)
            admin_df.to_excel(ADMIN_USERS_FILE, index=False)

def load_users():
    """Load regular users from Excel file"""
    if os.path.exists(USERS_FILE):
        return pd.read_excel(USERS_FILE)
    return pd.DataFrame(columns=['username', 'password', 'email', 'full_name', 'role', 'created_at'])

def load_admin_users():
    """Load admin users from Excel file"""
    if os.path.exists(ADMIN_USERS_FILE):
        return pd.read_excel(ADMIN_USERS_FILE)
    return pd.DataFrame(columns=['username', 'password', 'email', 'full_name', 'role', 'created_at'])

def save_users(users_df):
    """Save users to Excel file"""
    users_df.to_excel(USERS_FILE, index=False)

def save_admin_users(admin_df):
    """Save admin users to Excel file"""
    admin_df.to_excel(ADMIN_USERS_FILE, index=False)

def authenticate_user(username, password, user_type='user'):
    """Authenticate user login"""
    hashed_password = hash_password(password)

    if user_type == 'admin':
        users_df = load_admin_users()
    else:
        users_df = load_users()

    user = users_df[(users_df['username'] == username) & (users_df['password'] == hashed_password)]
    if not user.empty:
        return user.iloc[0].to_dict()
    return None

def register_user(username, password, email, full_name, user_type='user'):
    """Register a new user"""
    if user_type == 'admin':
        users_df = load_admin_users()
    else:
        users_df = load_users()

    # Check if username already exists
    if username in users_df['username'].values:
        return False, "Username already exists"

    # Check if email already exists
    if email in users_df['email'].values:
        return False, "Email already exists"

    # Add new user
    new_user = {
        'username': username,
        'password': hash_password(password),
        'email': email,
        'full_name': full_name,
        'role': user_type,
        'created_at': datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    }

    users_df = pd.concat([users_df, pd.DataFrame([new_user])], ignore_index=True)

    if user_type == 'admin':
        save_admin_users(users_df)
    else:
        save_users(users_df)

    return True, "Registration successful"

def get_all_users():
    """Get all regular users for admin view"""
    return load_users()

def get_all_admin_users():
    """Get all admin users"""
    return load_admin_users()

def delete_user(username, user_type='user'):
    """Delete a user"""
    if user_type == 'admin':
        users_df = load_admin_users()
    else:
        users_df = load_users()

    users_df = users_df[users_df['username'] != username]

    if user_type == 'admin':
        save_admin_users(users_df)
    else:
        save_users(users_df)

    return True, "User deleted successfully"

```

Python Code

```python
import pandas as pd
import numpy as np
from datetime import datetime, timedelta

def generate_energy_data(start_date='2020-01-01', periods=365*2, freq='D'):
    """
    Generate synthetic energy consumption data.
    """
    date_range = pd.date_range(start=start_date, periods=periods, freq=freq)
    np.random.seed(42)  # For reproducibility

    # Base consumption with seasonal and daily patterns
    base_consumption = 100  # kWh
    seasonal_amplitude = 20
    daily_amplitude = 10

    # Seasonal component (yearly cycle)
    seasonal = seasonal_amplitude * np.sin(2 * np.pi * np.arange(periods) / 365)

    # Daily component (weekly cycle)
    daily = daily_amplitude * np.sin(2 * np.pi * np.arange(periods) / 7)

    # Random noise
    noise = np.random.normal(0, 5, periods)

    # Trend (slight increase over time)
    trend = 0.01 * np.arange(periods)

    consumption = base_consumption + seasonal + daily + noise + trend

    # Ensure non-negative values
    consumption = np.maximum(consumption, 0)

    df = pd.DataFrame({
        'date': date_range,
        'consumption_kwh': consumption
    })

    return df

if __name__ == "__main__":
    data = generate_energy_data()
    data.to_csv('energy_data.csv', index=False)
    print("Synthetic energy data generated and saved to energy_data.csv")

```

Python Code

```python
import streamlit as st
import pandas as pd
from auth import authenticate_user, register_user, init_user_files
import re

def is_valid_email(email):
    """Validate email format"""
    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    return re.match(pattern, email) is not None

def login_page():
    """Main login page with user/admin selection"""
    st.set_page_config(page_title="EcoWatt Login", page_icon="⚡")

    # Initialize user files
    init_user_files()

    st.title("⚡ EcoWatt Login")

    # User type selection
    user_type = st.radio("Select User Type", ["User", "Admin"], horizontal=True)

    tab1, tab2 = st.tabs(["Login", "Register"])

    with tab1:
        login_section(user_type.lower())

    with tab2:
        register_section(user_type.lower())

def login_section(user_type):
    """Login section"""
    st.subheader(f"{user_type.title()} Login")

    with st.form(f"{user_type}_login_form"):
        username = st.text_input("Username")
        password = st.text_input("Password", type="password")

        submitted = st.form_submit_button("Login")

        if submitted:
            if not username or not password:
                st.error("Please fill in all fields")
            else:
                user = authenticate_user(username, password, user_type)
                if user:
                    st.session_state.logged_in = True
                    st.session_state.user = user
                    st.session_state.user_type = user_type
                    st.session_state.page = 'admin_dashboard' if user_type == 'admin' else 'user_dashboard'
                    st.success(f"Welcome back, {user['full_name']}!")
                    st.rerun()
                else:
                    st.error("Invalid username or password")

def register_section(user_type):
    """Registration section"""
    st.subheader(f"{user_type.title()} Registration")

    with st.form(f"{user_type}_register_form"):
        username = st.text_input("Username")
        email = st.text_input("Email")
        full_name = st.text_input("Full Name")
        password = st.text_input("Password", type="password")
        confirm_password = st.text_input("Confirm Password", type="password")

        submitted = st.form_submit_button("Register")

        if submitted:
            # Validation
            if not all([username, email, full_name, password, confirm_password]):
                st.error("Please fill in all fields")
            elif password != confirm_password:
                st.error("Passwords do not match")
            elif len(password) < 6:
                st.error("Password must be at least 6 characters long")
            elif not is_valid_email(email):
                st.error("Please enter a valid email address")
            elif len(username) < 3:
                st.error("Username must be at least 3 characters long")
            else:
                success, message = register_user(username, password, email, full_nam
e, user_type)
                if success:
                    st.success(message)
                    st.info("You can now login with your credentials")
                else:
                    st.error(message)

if __name__ == "__main__":
    login_page()
```

Python Code

```python
import streamlit as st

# Main application entry point
def main():
    # Initialize session state
    if 'logged_in' not in st.session_state:
        st.session_state.logged_in = False
        st.session_state.page = 'login'

    # Route to appropriate page
    if not st.session_state.logged_in or st.session_state.page == 'login':
        from login import login_page
        login_page()
    elif st.session_state.page == 'user_dashboard':
        # Import and run user dashboard
        from app import user_dashboard
        user_dashboard()
    elif st.session_state.page == 'admin_dashboard':
        from admin_dashboard import admin_dashboard
        admin_dashboard()

if __name__ == "__main__":
    main()

```

Python Code

```python
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import joblib
import os

def prepare_data(df, lag_days=7):
    """
    Prepare data for time series forecasting by creating lag features.
    """
    df = df.copy()
    df['date'] = pd.to_datetime(df['date'])
    df = df.set_index('date')

    # Create lag features
    for i in range(1, lag_days + 1):
        df[f'lag_{i}'] = df['consumption_kwh'].shift(i)

    # Drop rows with NaN values
    df = df.dropna()

    # Features and target
    X = df[[f'lag_{i}' for i in range(1, lag_days + 1)]]
    y = df['consumption_kwh']

    return X, y

def train_model(X_train, y_train):
    """
    Train a linear regression model.
    """
    model = LinearRegression()
    model.fit(X_train, y_train)
    return model

def predict_future(model, last_known_data, days_ahead=30, lag_days=7):
    """
    Predict future energy consumption.
    """
    predictions = []
    current_data = last_known_data.copy()

    for _ in range(days_ahead):
        # Prepare features for prediction
        features = np.array([current_data[-lag_days:]])
        pred = model.predict(features)[0]
        predictions.append(pred)

        # Update current data with prediction
        current_data = np.append(current_data[1:], pred)

    return predictions

def save_model(model, filename='energy_model.pkl'):
    """
    Save the trained model.
    """
    joblib.dump(model, filename)

def load_model(filename='energy_model.pkl'):
    """
    Load a trained model.
    """
    if os.path.exists(filename):
        return joblib.load(filename)
    else:
        return None

if __name__ == "__main__":
    # Load data
    data = pd.read_csv('energy_data.csv')

    # Prepare data
    X, y = prepare_data(data)

    # Split into train and test (80-20)
    split_idx = int(len(X) * 0.8)
    X_train, X_test = X[:split_idx], X[split_idx:]
    y_train, y_test = y[:split_idx], y[split_idx:]

    # Train model
    model = train_model(X_train, y_train)

    # Evaluate
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    print(f"Mean Squared Error: {mse:.2f}")

    # Save model
    save_model(model)
    print("Model trained and saved.")

```