```
from flask import Flask, render_template, jsonify, request, redirect, url_for,
session, flash
import mysql.connector
from mysql.connector import Error
from werkzeug.security import generate_password_hash, check_password_hash
from datetime import datetime, timedelta
import logging

app = Flask(__name__)
app.secret_key = 'your_secure_secret_key_here'  # Replace with a secure key

# Configure logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# Database configuration
DB_CONFIG = {
    'host': 'localhost',
    'user': 'root',
    'password': 'Kiku@999',
    'database': 'iot_farming'
}

# API key for ESP32
API_KEY = 'IOTg5_key'

# Helper functions
def get_db_connection():
    try:
        conn = mysql.connector.connect(**DB_CONFIG)
        if conn.is_connected():
            return conn
    except Error as e:
        logger.error(f"Database connection failed: {e}")
    return None

def verify_table_schema():
    conn = get_db_connection()
    if not conn:
        logger.error("Cannot verify table schema: No database connection")
        return False

    try:
        cursor = conn.cursor()
        cursor.execute("SHOW COLUMNS FROM alerts")
        columns = [col[0] for col in cursor.fetchall()]
        required_columns = ['id', 'type', 'message', 'severity', 'timestamp']
        missing_columns = [col for col in required_columns if col not in columns]

        if missing_columns:
            logger.error(f"Missing columns in alerts table: {missing_columns}")
            return False
        return True
    except Error as e:
        logger.error(f"Error verifying table schema: {e}")
        return False
    finally:
        if conn and conn.is_connected():
            conn.close()
```

```
def init_db():
    conn = get_db_connection()
    if not conn:
        logger.error("Failed to connect to database for initialization")
        return False

    try:
        cursor = conn.cursor()

        # Create users table
        cursor.execute("""
        CREATE TABLE IF NOT EXISTS users (
            id INT AUTO_INCREMENT PRIMARY KEY,
            username VARCHAR(50) UNIQUE NOT NULL,
            password VARCHAR(255) NOT NULL,
            email VARCHAR(100) UNIQUE NOT NULL,
            created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
        )
        """)

        # Create sensor_data table
        cursor.execute("""
        CREATE TABLE IF NOT EXISTS sensor_data (
            id INT AUTO_INCREMENT PRIMARY KEY,
            device_id VARCHAR(50) NOT NULL,
            temperature FLOAT,
            humidity FLOAT,
            soil_moisture INT,
            water_level INT,
            light_temt INT,
            light_mh INT,
            ambient_light INT,
            proximity INT,
            motion BOOLEAN,
            timestamp DATETIME DEFAULT CURRENT_TIMESTAMP
        )
        """)

        # Drop and recreate alerts table to ensure correct schema
        cursor.execute("DROP TABLE IF EXISTS alerts")
        cursor.execute("""
        CREATE TABLE alerts (
            id INT AUTO_INCREMENT PRIMARY KEY,
            type VARCHAR(50) NOT NULL,
            message TEXT NOT NULL,
            severity VARCHAR(20) NOT NULL,
            timestamp DATETIME DEFAULT CURRENT_TIMESTAMP
        )
        """)

        conn.commit()
        logger.info("Database initialized successfully")
        return True
    except Error as e:
        logger.error(f"Database initialization failed: {e}")
        return False
    finally:
        if conn and conn.is_connected():
```

```
            conn.close()

# Fetch motion-based alerts
def fetch_weather_alerts():
    conn = get_db_connection()
    if not conn:
        logger.error("Database connection failed for fetching alerts")
        return 'No Alerts'

    try:
        cursor = conn.cursor()
        cursor.execute("""
            SELECT motion FROM sensor_data
            ORDER BY timestamp DESC
            LIMIT 1
        """)
        result = cursor.fetchone()

        if result and result[0]:  # motion is True (YES)
            return 'Motion Detected'
        return 'No Alerts'
    except Error as e:
        logger.error(f"Error fetching motion alert: {e}")
        return 'No Alerts'
    finally:
        if conn and conn.is_connected():
            conn.close()

# Fetch hardcoded 7-day weather forecast for Kansas City (May 6-12, 2025)
def fetch_weather_forecast():
    forecast = [
        {'date': '2025-05-06', 'temp_high': 22, 'temp_low': 12, 'condition':
'Clear'},
        {'date': '2025-05-07', 'temp_high': 20, 'temp_low': 10, 'condition':
'Clouds'},
        {'date': '2025-05-08', 'temp_high': 19, 'temp_low': 9, 'condition':
'Rain'},
        {'date': '2025-05-09', 'temp_high': 23, 'temp_low': 13, 'condition':
'Clear'},
        {'date': '2025-05-10', 'temp_high': 21, 'temp_low': 11, 'condition':
'Partly Cloudy'},
        {'date': '2025-05-11', 'temp_high': 24, 'temp_low': 14, 'condition':
'Clear'},
        {'date': '2025-05-12', 'temp_high': 20, 'temp_low': 10, 'condition':
'Rain'}
    ]
    return forecast

# Authentication routes
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        email = request.form['email']

        conn = get_db_connection()
        if not conn:
            flash('Database connection error', 'danger')
```

```
            return redirect(url_for('register'))

        try:
            cursor = conn.cursor()
            hashed_password = generate_password_hash(password)
            cursor.execute(
                "INSERT INTO users (username, password, email) VALUES (%s, %s,
%s)",
                (username, hashed_password, email)
            )
            conn.commit()
            flash('Registration successful! Please login.', 'success')
            return redirect(url_for('login'))
        except Error as e:
            logger.error(f"Registration error: {e}")
            flash('Username or email already exists', 'danger')
        finally:
            if conn and conn.is_connected():
                conn.close()

    return render_template('register.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        conn = get_db_connection()
        if not conn:
            flash('Database connection error', 'danger')
            return redirect(url_for('login'))

        try:
            cursor = conn.cursor(dictionary=True)
            cursor.execute("SELECT * FROM users WHERE username = %s", (username,))
            user = cursor.fetchone()

            if user and check_password_hash(user['password'], password):
                session['user_id'] = user['id']
                session['username'] = user['username']
                flash('Login successful!', 'success')
                return redirect(url_for('dashboard'))
            else:
                flash('Invalid username or password', 'danger')
        except Error as e:
            logger.error(f"Login error: {e}")
            flash('Login error', 'danger')
        finally:
            if conn and conn.is_connected():
                conn.close()

    return render_template('login.html')

@app.route('/logout')
def logout():
    session.clear()
    flash('You have been logged out', 'info')
    return redirect(url_for('login'))
```

```
# API endpoints
@app.route('/api/logout', methods=['POST'])
def api_logout():
    session.clear()
    return jsonify({'status': 'success', 'message': 'Logged out'})

@app.route('/api/sensor-data', methods=['POST'])
def receive_sensor_data():
    # Check API key
    api_key = request.headers.get('X-API-Key')
    if api_key != API_KEY:
        logger.warning("Unauthorized sensor data request")
        return jsonify({'error': 'Unauthorized'}), 401

    data = request.get_json()
    if not data or 'device_id' not in data:
        logger.error("Invalid sensor data received")
        return jsonify({'error': 'Invalid data'}), 400

    # Verify table schema
    if not verify_table_schema():
        logger.error("Invalid alerts table schema")
        return jsonify({'error': 'Database schema error'}), 500

    conn = get_db_connection()
    if not conn:
        logger.error("Database connection failed for sensor data")
        return jsonify({'error': 'Database connection failed'}), 500

    try:
        cursor = conn.cursor()

        # Insert sensor data
        cursor.execute("""
            INSERT INTO sensor_data (
                device_id, temperature, humidity, soil_moisture, water_level,
                light_temt, light_mh, ambient_light, proximity, motion
            ) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
        """, (
            data['device_id'],
            data.get('temperature'),
            data.get('humidity'),
            data.get('soil_moisture'),
            data.get('water_level'),
            data.get('light_temt'),
            data.get('light_mh'),
            data.get('ambient_light'),
            data.get('proximity'),
            data.get('motion')
        ))

        # Generate alerts
        alerts = []
        if data.get('water_level', 0) < 2000:
            alerts.append(('low_water', 'Low water level detected. Refill
recommended.', 'warning'))
        if data.get('temperature', 0) > 30:
            alerts.append(('high_temperature', 'High temperature detected. Consider
```

```
cooling measures.', 'danger'))
        if data.get('soil_moisture', 100) < 10:
            alerts.append(('low_soil_moisture', 'Low soil moisture detected.
Immediate irrigation recommended.', 'danger'))
        if data.get('soil_moisture', 0) > 90:
            alerts.append(('flooding', 'Flooding detected. Check drainage
systems.', 'danger'))

        # Check for persistent motion (10+ consecutive readings)
        cursor.execute("""
            SELECT motion FROM sensor_data
            WHERE device_id = %s
            ORDER BY timestamp DESC
            LIMIT 10
        """, (data['device_id'],))
        motion_readings = [row[0] for row in cursor.fetchall()]
        if len(motion_readings) >= 10 and all(motion_readings):
            alerts.append(('persistent_motion', 'Persistent motion detected.
Possible intrusion.', 'warning'))

        for alert_type, message, severity in alerts:
            cursor.execute("""
                INSERT INTO alerts (type, message, severity)
                VALUES (%s, %s, %s)
            """, (alert_type, message, severity))

        conn.commit()
        logger.info(f"Sensor data received and stored from device
{data['device_id']}")
        return jsonify({'status': 'success', 'message': 'Data received'})
    except Error as e:
        logger.error(f"Error storing sensor data: {e}")
        return jsonify({'error': str(e)}), 500
    finally:
        if conn and conn.is_connected():
            conn.close()

@app.route('/api/current-data')
def get_current_data():
    if 'user_id' not in session:
        return jsonify({'error': 'Unauthorized'}), 401

    conn = get_db_connection()
    if not conn:
        logger.error("Database connection failed for current data")
        return jsonify({'error': 'Database connection failed'}), 500

    try:
        cursor = conn.cursor(dictionary=True)

        # Get latest sensor readings
        cursor.execute("""
            SELECT * FROM sensor_data
            ORDER BY timestamp DESC
            LIMIT 1
        """)
        current_data = cursor.fetchone()

        if not current_data:
```

```
            logger.warning("No sensor data available")
            return jsonify({'error': 'No sensor data available'}), 404

        # Get historical data for charts
        cursor.execute("""
            SELECT
                temperature, humidity, soil_moisture,
                light_temt AS light_intensity,
                DATE_FORMAT(timestamp, '%H:%i') AS time
            FROM sensor_data
            WHERE timestamp > NOW() - INTERVAL 24 HOUR
            ORDER BY timestamp
        """)
        historical_data = cursor.fetchall()

        # Process for charts
        chart_data = {
            'times': [row['time'] for row in historical_data],
            'temperature': [row['temperature'] for row in historical_data],
            'humidity': [row['humidity'] for row in historical_data],
            'light': [row['light_intensity'] for row in historical_data],
            'soil': [row['soil_moisture'] for row in historical_data]
        }

        # AI recommendations with enhanced insights
        soil_moisture = current_data['soil_moisture']
        humidity = current_data['humidity']
        temperature = current_data['temperature']

        # Crop yield prediction (simplified linear model)
        yield_score = (soil_moisture * 0.4 + humidity * 0.3 + (30 - abs(temperature
- 25)) * 0.3)
        yield_prediction = 'High' if yield_score > 60 else 'Moderate' if
yield_score > 40 else 'Low'

        # Irrigation efficiency
        irrigation_efficiency = 'Optimal' if 40 <= soil_moisture <= 80 else
'Suboptimal'

        # Pest risk
        pest_risk = 'High' if humidity > 80 and temperature > 25 else 'Low'

        ai_recommendations = {
            'crop': 'Tomatoes' if soil_moisture > 40 else 'Lettuce',
            'irrigation': '8:00 AM' if soil_moisture < 70 else 'Skip',
            'weather_alert': fetch_weather_alerts(),
            'soil_moisture': soil_moisture,
            'yield_prediction': yield_prediction,
            'irrigation_efficiency': irrigation_efficiency,
            'pest_risk': pest_risk
        }

        # Weather forecast
        weather_forecast = fetch_weather_forecast()

        return jsonify({
            'current': current_data,
            'charts': chart_data,
            'ai_recommendations': ai_recommendations,
```

```
            'weather_forecast': weather_forecast,
            'status': 'success'
        })

    except Error as e:
        logger.error(f"Error fetching current data: {e}")
        return jsonify({'error': str(e)}), 500
    finally:
        if conn and conn.is_connected():
            conn.close()

@app.route('/api/alerts')
def get_alerts():
    if 'user_id' not in session:
        return jsonify({'error': 'Unauthorized'}), 401

    conn = get_db_connection()
    if not conn:
        logger.error("Database connection failed for alerts")
        return jsonify({'error': 'Database connection failed'}), 500

    try:
        cursor = conn.cursor(dictionary=True)
        cursor.execute("""
            SELECT id, type, message, severity, timestamp
            FROM alerts
            WHERE timestamp > NOW() - INTERVAL 24 HOUR
            ORDER BY timestamp DESC
            LIMIT 10
        """)
        alerts = cursor.fetchall()
        return jsonify({'alerts': alerts, 'status': 'success'})
    except Error as e:
        logger.error(f"Error fetching alerts: {e}")
        return jsonify({'error': str(e)}), 500
    finally:
        if conn and conn.is_connected():
            conn.close()

# Dashboard route
@app.route('/')
def home():
    if 'user_id' not in session:
        return redirect(url_for('login'))
    return redirect(url_for('dashboard'))

@app.route('/dashboard')
def dashboard():
    if 'user_id' not in session:
        return redirect(url_for('login'))
    return render_template('dashboard.html', username=session.get('username'))

if __name__ == '__main__':
    if init_db():
        app.run(host='0.0.0.0', port=5000, debug=True)
    else:
        logger.error("Failed to initialize database")
```