```python
import asyncio
import json
from datetime import datetime
from fastapi.responses import HTMLResponse
import os
from pydantic import BaseModel
from pymodbus.client import ModbusTcpClient
from pathlib import Path

from starlette import status
from starlette.responses import Response
from fastapi.staticfiles import StaticFiles
import struct
import subprocess
import sys
import time
from fastapi import FastAPI, WebSocket

DEBUG_LOG = True
SIMULATE = False

MODBUS_IP = "192.168.2.231"  # 192.168.2.231
MODBUS_PORT = 502  # 502

MODBUS_REG_BUTTON_ENABLED = 0
MODBUS_REG_BUTTON_SEND = 1
MODBUS_REG_BUTTON_RECEIVED = 2
MODBUS_REG_BUTTON_RESULT = 3
MODBUS_REG_STATE = 4
MODBUS_REG_ALARM = 5
MODBUS_REG_WARNING = 6
MODBUS_REG_STATE_SP_GRILL1 = 7
MODBUS_REG_STATE_SP_GRILL2 = 9
MODBUS_REG_STATE_SP_MIN_GRILL = 11

MODBUS_REG_STATE_STATE = 19
MODBUS_REG_STATE_PV_GRILL1 = 21
MODBUS_REG_STATE_PV_GRILL2 = 23

MODBUS_HMI_BTN_START = 0
MODBUS_HMI_BTN_STOP = 1
MODBUS_HMI_BTN_START_JOG = 2
MODBUS_HMI_BTN_ACK = 3
MODBUS_HMI_BTN_RESET = 4
MODBUS_HMI_BTN_SEL_DESKA = 5
MODBUS_HMI_BTN_SEL_GRIL = 6
MODBUS_HMI_BTN_GRIL1 = 7
MODBUS_HMI_BTN_GRIL2 = 8
MODBUS_LIFEBIT = 15


def float_from_integers(integer1, integer2):
    return struct.unpack('!f', struct.pack('!HH', integer1, integer2))[0]


def integers_from_float(float):
    return struct.unpack('!HH', struct.pack('!f', float))


def write_bit(modbus_client, register, bit, value):
    or_mask = {0: 256,
                1: 512,
                2: 1024,
                3: 2048,
                4: 4096,
                5: 8192,
                6: 16384,
                7: 32768,
                8: 1,
                9: 2,
                10: 4,
                11: 8,
                12: 16,
                13: 32,
                14: 64,
                15: 128}
    and_mask = {0: ~256,
                 1: ~512,
                 2: ~1024,
                 3: ~2048,
                 4: ~4096,
                 5: ~8192,
                 6: ~16384,
                 7: ~32768,
                 8: ~1,
                 9: ~2,
                 10: ~4,
                 11: ~8,
                 12: ~16,
                 13: ~32,
                 14: ~64,
                 15: ~128}
    if SIMULATE:
        registers = [0]
    else:
        response = modbus_client.read_holding_registers(register, 1)
        registers = response.registers
    current_value = registers[0]
    if value == 1:
        new_value = current_value | or_mask[bit]
    else:
        new_value = current_value & and_mask[bit]
    data = [new_value]
    if not SIMULATE:
        modbus_client.write_registers(register, data)
    return data


def read_bit(modbus_client, register, bit):
    and_mask = {0: ~256,
                 1: ~512,
                 2: ~1024,
                 3: ~2048,
                 4: ~4096,
                 5: ~8192,
                 6: ~16384,
                 7: ~32768,
                 8: ~1,
                 9: ~2,
                 10: ~4,
                 11: ~8,
                 12: ~16,
            13: ~32,
            14: ~64,
            15: ~128}
    if SIMULATE:
        registers = [0]
    else:
        response = modbus_client.read_holding_registers(register, 1)
        registers = response.registers
    current_value = registers[0]
    if current_value & and_mask[bit] == current_value:
        return 0
    else:
        return 1


def write_word(modbus_client, register, word):
    data = [word]
    if not SIMULATE:
        modbus_client.write_registers(register, data)
    return word


def read_word(modbus_client, register):
    if SIMULATE:
        registers = [0]
    else:
        response = modbus_client.read_holding_registers(register, 1)
        registers = response.registers
    return registers[0]


def write_float(modbus_client, register, float1):
    f1 = integers_from_float(float1)
    data = [f1[0], f1[1]]
    if not SIMULATE:
        modbus_client.write_registers(register, data)
    return float1


def read_float(modbus_client, register):
    if SIMULATE:
        registers = [0, 0]
    else:
        response = modbus_client.read_holding_registers(register, 2)
        registers = response.registers
    return float_from_integers(registers[0], registers[1])


def decode_machine_state(state):
    if state == 0:
        result = "0 - Stop - Provozní poloha HH - nenatočeno"
    elif state == 1:
        result = "1 - Stop - Mezipoloha H - nenatočeno"
    elif state == 2:
        result = "2 - Stop - Mezipoloha M - nenatočeno"
    elif state == 3:
        result = "3 - Stop - Dolní poloha L"
    elif state == 4:
        result = "4 - Stop - Deska v provozní poloze HH"
    elif state == 5:
        result = "5 - Stop - Deska v mezipoloze H"
    elif state == 6:
        result = "6 - Stop - Deska v mezipoloze M"
    elif state == 7:
        result = "7 - Stop - Deska v dolní poloze L"
    elif state == 8:
        result = "8 - Stop - Gril v provozní poloze HH"
    elif state == 9:
        result = "9 - Stop - Stop - Gril v mezipoloze H"
    elif state == 10:
        result = "10 - Stop - Stop - Gril v mezipoloze M"
    elif state == 11:
        result = "11 - Stop - Gril v dolní poloze L"
    elif state == 21:
        result = "21 - Chod - Gril dolů"
    elif state == 22:
        result = "22 - Chod - Otoč na desku"
    elif state == 23:
        result = "23 - Chod - Deska nahoru"
    elif state == 24:
        result = "24 - Chod - Deska nahoru ruční dojezd"
    elif state == 31:
        result = "31 - Chod - Deska dolů"
    elif state == 32:
        result = "32 - Chod - Otoč na gril"
    elif state == 33:
        result = "33 - Chod - Gril nahoru"
    elif state == 34:
        result = "34 - Chod - Gril nahoru ruční dojezd"
    else:
        result = str(state) + " - neznámý stav"

    return result


def handle_button_click(modbusclient, bit, name):
    if DEBUG_LOG:
        print("zkontroluj jestli je možné použít tlačítko " + name + "...")

    if read_bit(modbusclient, MODBUS_REG_BUTTON_ENABLED, bit) == 0:
        if DEBUG_LOG:
            print("tlačítko " + name + " není povoleno")
        return

    if DEBUG_LOG:
        print("stisknout tlačítko " + name + "...")
    write_bit(modbusclient, MODBUS_REG_BUTTON_SEND, bit, 1)

    if DEBUG_LOG:
        print("čekáme na potvrzení přijetí povelu tlačítka " + name +
"...")
    while read_bit(modbusclient, MODBUS_REG_BUTTON_RECEIVED, bit) == 0:
        time.sleep(0.1)

    if DEBUG_LOG:
        print("přijetí povelu tlačítka " + name + " potvrzeno...")
        print("shodíme povel stisknuto tlačítko " + name + "...")
    write_bit(modbusclient, MODBUS_REG_BUTTON_SEND, bit, 0)

    # vypneme případně aktivovaný dojezd
    if (bit == MODBUS_HMI_BTN_SEL_DESKA) | (bit ==
MODBUS_HMI_BTN_SEL_GRIL):
        if DEBUG_LOG:
            print("shazuji povel tlačítka dojezd...")
        write_bit(modbusclient, MODBUS_REG_BUTTON_SEND,
MODBUS_HMI_BTN_START_JOG, 0)

    return


def handle_button_momentary_click(modbusclient, bit, name):
    if DEBUG_LOG:
        print("zkontroluj jestli je možné použít tlačítko " + name + "...")

    if read_bit(modbusclient, MODBUS_REG_BUTTON_ENABLED, bit) == 0:
        if DEBUG_LOG:
            print("tlačítko " + name + " není povoleno")
        return

    if DEBUG_LOG:
        print("stisknout tlačítko " + name + "...")
    write_bit(modbusclient, MODBUS_REG_BUTTON_SEND, bit, 1)

    if DEBUG_LOG:
        print("čekáme na potvrzení přijetí povelu tlačítka " + name +
"...")
    while read_bit(modbusclient, MODBUS_REG_BUTTON_RECEIVED, bit) == 0:
        time.sleep(0.1)

    if DEBUG_LOG:
        print("přijetí povelu tlačítka " + name + " potvrzeno...")

    return


def handle_button_momentary_click_end(modbusclient, bit, name):
    if DEBUG_LOG:
        print("uvolnit tlačítko " + name + "...")
    write_bit(modbusclient, MODBUS_REG_BUTTON_SEND, bit, 0)

    return


def button_start(modbusclient):
    if DEBUG_LOG:
        print("voláno button_start()...")
    handle_button_click(modbusclient, MODBUS_HMI_BTN_START, "Start")
    return


def button_stop(modbusclient):
    if DEBUG_LOG:
        print("voláno button_stop()...")
    handle_button_click(modbusclient, MODBUS_HMI_BTN_STOP, "Stop")
    return


def button_start_jog(modbusclient):
    if DEBUG_LOG:
        print("voláno button_start_jog()...")
    handle_button_momentary_click(modbusclient, MODBUS_HMI_BTN_START_JOG,
"Dojezd")
    return
```

```
def button_stop_jog(modbusclient):
    if DEBUG_LOG:
        print("voláno button_stop_jog()...")
    handle_button_momentary_click_end(modbusclient,
MODBUS_HMI_BTN_START_JOG, "Dojezd")
    return


def button_ack(modbusclient):
    if DEBUG_LOG:
        print("voláno button_ack()...")
    handle_button_click(modbusclient, MODBUS_HMI_BTN_ACK, "Ack")
    return


def button_reset(modbusclient):
    if DEBUG_LOG:
        print("voláno button_reset()...")
    handle_button_click(modbusclient, MODBUS_HMI_BTN_RESET, "Reset")
    return


def button_deska(modbusclient):
    if DEBUG_LOG:
        print("voláno button_deska()...")
    handle_button_click(modbusclient, MODBUS_HMI_BTN_SEL_DESKA, "Select
deska")
    return


def button_gril(modbusclient):
    if DEBUG_LOG:
        print("voláno button_gril()...")
    handle_button_click(modbusclient, MODBUS_HMI_BTN_SEL_GRIL, "Select
gril")
    return


def button_toggle_gril1(modbusclient):
    if DEBUG_LOG:
        print("voláno button_toggle_gril1()...")
    handle_button_click(modbusclient, MODBUS_HMI_BTN_GRIL1, "Toggle Gril1")
    return


def button_toggle_gril2(modbusclient):
    if DEBUG_LOG:
        print("voláno button_toggle_gril2()...")
    handle_button_click(modbusclient, MODBUS_HMI_BTN_GRIL2, "Toggle Gril2")
    return


def save_wifi_and_connect(ssid, password, wpaconfig, tempfile):
    try:
        lines = subprocess.check_output(['sudo', 'cat', wpaconfig])
    except subprocess.CalledProcessError as err:
        print(err)
        sys.exit('Error: Get configed wifi netword list failed!')
    lines2 = bytes.decode(lines, 'utf-8')

    x = lines2.split('\n')

    # writing to file
    output_file = open(tempfile, 'w')

    before_separator = True
    count = 0
    for line in x:
        count += 1
        if before_separator:
            output_file.write(line)
            output_file.write("\n")
        if line.strip() == '# everything after this line will be discarded
on app config write':
            before_separator = False

    output_file.write("\n")
    output_file.write("network={\n")
    output_file.write("        ssid=\"" + ssid + "\"\n")
    output_file.write("        psk=\"" + password + "\"\n")
    output_file.write("        key_mgmt=WPA-PSK\n")
    output_file.write("}\n")

    output_file.close()

    # move tmp file into the finas location
    os.system('sudo mv ' + tempfile + ' ' + wpaconfig)

    # adjust config file permissions
    os.system('sudo chown root:root ' + wpaconfig)
    os.system('sudo chmod g-r,g-w,o-r,o-w ' + wpaconfig)

    # restart WiFi
    os.system('wpa_cli -i wlan0 reconfigure')

    return

modbusclient = ModbusTcpClient(MODBUS_IP, port=MODBUS_PORT)
modbusclient.connect()
# write_bit(modbusclient, 0, 0, 1)
# write_bit(modbusclient, 0, 1, 0)
# write_bit(modbusclient, 0, 2, 1)
lifebit = 1

app = FastAPI()
app.mount("/css", StaticFiles(directory="static/css"), name="css")
app.mount("/js", StaticFiles(directory="static/js"), name="js")
app.mount("/img", StaticFiles(directory="static/img"), name="img")


@app.get("/")
async def get():
    html = Path("templates/index.html").read_text()
    now = datetime.now()
    html = html.replace('DATETIMEPLACEHOLDER',
now.strftime("%Y%m%d%H%M%S"))
    return HTMLResponse(html)
```

```
@app.get("/button/start")
async def get(response: Response):
    try:
        button_start(modbusclient)
    except Exception as err:
        print(f"Unexpected {err=}, {type(err)=}")
        response.status_code = status.HTTP_500_INTERNAL_SERVER_ERROR
        return 'Error'
    return


@app.get("/button/stop")
async def get(response: Response):
    try:
        button_stop(modbusclient)
    except Exception as err:
        print(f"Unexpected {err=}, {type(err)=}")
        response.status_code = status.HTTP_500_INTERNAL_SERVER_ERROR
        return 'Error'
    return


@app.get("/button/start-jog")
async def get():
    button_start_jog(modbusclient)
    return


@app.get("/button/stop-jog")
async def get():
    button_stop_jog(modbusclient)
    return


@app.get("/button/ack")
async def get(response: Response):
    try:
        button_ack(modbusclient)
    except Exception as err:
        print(f"Unexpected {err=}, {type(err)=}")
        response.status_code = status.HTTP_500_INTERNAL_SERVER_ERROR
        return 'Error'
    return


@app.get("/button/reset")
async def get(response: Response):
    try:
        button_reset(modbusclient)
    except Exception as err:
        print(f"Unexpected {err=}, {type(err)=}")
        response.status_code = status.HTTP_500_INTERNAL_SERVER_ERROR
        return 'Error'
    return


@app.get("/button/sel-deska")
async def get(response: Response):
    try:
        button_deska(modbusclient)
    except Exception as err:
        print(f"Unexpected {err=}, {type(err)=}")
        response.status_code = status.HTTP_500_INTERNAL_SERVER_ERROR
        return 'Error'
    return


@app.get("/button/sel-gril")
async def get(response: Response):
    try:
        button_gril(modbusclient)
    except Exception as err:
        print(f"Unexpected {err=}, {type(err)=}")
        response.status_code = status.HTTP_500_INTERNAL_SERVER_ERROR
        return 'Error'
    return


@app.get("/button/toggle-gril1")
async def get(response: Response):
    try:
        button_toggle_gril1(modbusclient)
    except Exception as err:
        print(f"Unexpected {err=}, {type(err)=}")
        response.status_code = status.HTTP_500_INTERNAL_SERVER_ERROR
        return 'Error'
    return


@app.get("/button/toggle-gril2")
async def get(response: Response):
    try:
        button_toggle_gril2(modbusclient)
    except Exception as err:
        print(f"Unexpected {err=}, {type(err)=}")
        response.status_code = status.HTTP_500_INTERNAL_SERVER_ERROR
        return 'Error'
    return


class SP_Gril(BaseModel):
    gril1: float
    gril2: float


@app.post("/update-sp-gril")
async def post(sp: SP_Gril):
    if DEBUG_LOG:
        print("SP_Gril1: " + str(sp.gril1))
        print("SP_Gril2: " + str(sp.gril2))
    write_float(modbusclient, MODBUS_REG_STATE_SP_GRILL1, sp.gril1)
    write_float(modbusclient, MODBUS_REG_STATE_SP_GRILL2, sp.gril2)

    return


class Settings(BaseModel):
    ssid: str
    password: str


@app.post("/settings")
```

```
async def post(settings: Settings):

    if DEBUG_LOG:
        print("voláno /settings...")

        print("ssid: " + str(settings.ssid))
        print("password: " + str(settings.password))

    save_wifi_and_connect(settings.ssid, settings.password,
'/etc/wpa_supplicant/wpa_supplicant.conf', '/tmp/wpa_supplicant.conf')

    return


@app.websocket("/ws")
async def websocket_endpoint(websocket: WebSocket):
    global lifebit
    await websocket.accept()
    while True:
        try:
            buttons_enable = read_word(modbusclient,
MODBUS_REG_BUTTON_ENABLED)
            state = read_word(modbusclient, MODBUS_REG_STATE)
            alarm = read_word(modbusclient, MODBUS_REG_ALARM)
            warning = read_word(modbusclient, MODBUS_REG_WARNING)
            state_state = read_word(modbusclient, MODBUS_REG_STATE_STATE +
1)  # TODO ugly solution for double int
            result = read_word(modbusclient, MODBUS_REG_BUTTON_RESULT)
            data = {
                'life_bit': lifebit,
                'result': result,
                'button_start': (buttons_enable & 256) != 0,
                'button_stop': (buttons_enable & 512) != 0,
                'button_jog': (buttons_enable & 1024) != 0,
                'button_ack': (buttons_enable & 2048) != 0,
                'button_reset': (buttons_enable & 4096) != 0,
                'button_deska': (buttons_enable & 8192) != 0,
                'button_grill': (buttons_enable & 16384) != 0,
                'button_grill1': (buttons_enable & 32768) != 0,
                'button_grill2': (buttons_enable & 1) != 0,
                'state': state,
                'alarm': alarm,
                'warning': warning,
                'state_state': state_state,
                'state_text': decode_machine_state(state_state),
                'SP_Gril1': read_float(modbusclient,
MODBUS_REG_STATE_SP_GRILL1),
                'SP_Gril2': read_float(modbusclient,
MODBUS_REG_STATE_SP_GRILL2),
                'SP_min_Gril': read_float(modbusclient,
MODBUS_REG_STATE_SP_MIN_GRILL),
                'PV_Gril1': read_float(modbusclient,
MODBUS_REG_STATE_PV_GRILL1),
                'PV_Gril2': read_float(modbusclient,
MODBUS_REG_STATE_PV_GRILL2),
            }
        except Exception as err:
            print(f"Unexpected {err=}, {type(err)=}")
            data = {
                'life_bit': 0,
                'result': 0,
                'button_start': False,
                'button_stop': False,
                'button_jog': False,
                'button_ack': True,
                'button_reset': False,
                'button_deska': False,
                'button_grill': False,
                'button_grill1': False,
                'button_grill2': False,
                'state': 0,  # TODO lepší by bylo posílat null hodnotu a na
frontendu detekovat a zobrazit např. '-'
                'alarm': 32,  # TODO rozlišit chybu komunikace od PLC nebo
vlastní
                'warning': 0,
                'state_state': 0,
                'state_text': '',
                'SP_Gril1': 0,
                'SP_Gril2': 0,
                'SP_min_Gril': 0,
                'PV_Gril1': 0,
                'PV_Gril2': 0,
            }
        await websocket.send_json(data)
        try:
            write_bit(modbusclient, MODBUS_REG_BUTTON_SEND, MODBUS_LIFEBIT,
lifebit)
        except Exception as err:
            print(f"Unexpected {err=}, {type(err)=}")
            data = {
                'life_bit': 0,
                'result': 0,
                'button_start': False,
                'button_stop': False,
                'button_jog': False,
                'button_ack': True,
                'button_reset': False,
                'button_deska': False,
                'button_grill': False,
                'button_grill1': False,
                'button_grill2': False,
                'state': 0,  # TODO lepší by bylo posílat null hodnotu a na
frontendu detekovat a zobrazit např. '-'
                'alarm': 32,  # TODO rozlišit chybu komunikace od PLC nebo
vlastní
                'warning': 0,
                'state_state': 0,
                'state_text': '',
                'SP_Gril1': 0,
                'SP_Gril2': 0,
                'SP_min_Gril': 0,
                'PV_Gril1': 0,
                'PV_Gril2': 0,
            }

        if lifebit == 0:
            lifebit = 1
        else:
            lifebit = 0

        await asyncio.sleep(0.5)
```

```
# TODO implementovat dword čtení
# TODO or_mask dát do konstanty
# TODO and_mask dát do konstanty
# TODO teoreticky by mohly konstanty jako např. MODBUS_HMI_BTN_START mít
přímo hodnotu masky a vynechala by se tabulka pro and a or masky
# TODO rozdělit do více menších souborů
# TODO refactoring
```