Finalni Vodič: Toplik Service + ha-bridge + Alexa (Verzija 4.0)

Ovaj dokument opisuje kompletan i ispravan proces za postavljanje Toplik Service Python servera i ha-bridge (Philips Hue Emulatora) na Raspberry Pi (toplik-server), te njihovo povezivanje sa Amazon Echo (Alexa) uređajima za glasovnu kontrolu.

Ovaj vodič rješava sve početne probleme, uključujući:

1. Ispravnu instalaciju svih zavisnosti.
2. Korištenje ha-bridge umjesto diyHue.
3. Ispravnu konfiguraciju termostata da prihvati glasovne komande **bez riječi "percent"** (npr. "Alexa, set thermostat to 22").
4. Korištenje Alexa Grupa za slanje komandi **bez navođenja broja sobe** (npr. "Alexa, turn on light").

Faza 0: Priprema Sistema (Obavezno)

Prije instalacije, server se mora ažurirati i mora se instalirati Java.

1. **Ažurirajte Listu Paketa:**
   sudo apt update

2. Nadogradite Sistem:
   (Ovo može potrajati nekoliko minuta)
   sudo apt upgrade -y

Faza 1: Instalacija Java Okruženja

ha-bridge-5.4.1-java11.jar [cite: user prompt] specifično zahtijeva Javu 11.

1. **Instalirajte Javu 11:**
   sudo apt install openjdk-11-jre -y

2. **Provjerite Verziju:**
   java -version

   (Izlaz bi trebao pokazati "openjdk version 11...")

Faza 2: Postavljanje Python Servera (Toplik Service)

Ovo je Vaš glavni server koji prima komande i prosljeđuje ih na ESP32 uređaje.

1. Kreirajte Direktorij Servisa:

(Preskočite ako već postoji)
mkdir /home/admin/ToplikService
cd /home/admin/ToplikService

2. Instalirajte Zavisnosti:
   Kreirajte fajl requirements.txt [cite: requirements.txt] sa nano requirements.txt:
   Flask
   requests
   PyJWT
   waitress

   Instalirajte ih:
   pip install -r requirements.txt

3. Kreirajte config_server.json:
   Ovo je Vaš glavni konfiguracijski fajl. [cite: config_server.json]
   nano /home/admin/ToplikService/config_server.json

   Zalijepite Vaš JSON sadržaj unutra.
4. Kreirajte server.py (Finalna Verzija sa Ograničenjem):
   Ovo je kompletan kod za server.py [cite: server.py]. On sadrži ključnu ispravku u api_ifttt_control funkciji koja ograničava (clamping) temperaturu poslanu od Alexe na min/max (18-30), rješavajući problem gdje je "100 percent" slalo "100" [cite: user prompt].
   nano /home/admin/ToplikService/server.py

   Zalijepite **cijeli** ovaj kod unutra:

```
# ---------------------------------------------------------------------
#  TOPLIK SERVICE - PYTHON BACKEND SERVER (FAZA 7.0: Admin PIN Kontrola)
# ---------------------------------------------------------------------
import os
import json
import requests
import jwt
from datetime import datetime, timedelta
from flask import Flask, request, jsonify, render_template, redirect, make_response, url_for
from waitress import serve
import re
import threading
import time
import logging
from collections import OrderedDict
```

```
# --- INICIJALIZACIJA APLIKACIJE ---
app = Flask(__name__)
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s -
%(message)s')

# --- GLOBALNE VARIJABLE ---
CONFIG = {}
config_lock = threading.Lock()
# OBAVEZNO promijenite 'SECRET_KEY' u nešto unikatno i tajno
app.config['SECRET_KEY'] = 'VAS-TAJNI-KLJUC-ZA-TOKENE-12345'

# ----------------------------------------------------------------------
# FUNKCIJE ZA RUKOVANJE KONFIGURACIJOM
# ----------------------------------------------------------------------
def load_config():
    global CONFIG
    try:
        with open('config_server.json', 'r', encoding='utf-8') as f:
            original_config = json.load(f, object_pairs_hook=OrderedDict)
            for soba_id, soba_data in original_config.get('sobe', {}).items():
                soba_data['cached_ip'] = None

            with config_lock:
                CONFIG = original_config
            logging.info("Uspješno učitan i inicijaliziran 'config_server.json'.")
    except Exception as e:
        logging.critical(f"!!! GRESKA pri čitanju 'config_server.json': {e}")
        exit(1)

def save_config():
    with config_lock:
        try:
            config_to_save = CONFIG.copy()
            with open('config_server.json', 'w', encoding='utf-8') as f:
                json.dump(config_to_save, f, indent=2, ensure_ascii=False)
            logging.info("CONFIG: Uspješno spremljen config_server.json.")
            return True
        except Exception as e:
            logging.error(f"CONFIG: Nije moguće upisati u config_server.json: {e}")
            return False

# ----------------------------------------------------------------------
```

```
#  PARSER ZA GET_STATUS
# ---------------------------------------------------------------------
def parse_get_status(raw_text):
    main_data = {}
    thermostat_data = {}
    pin_states = {}
    current_section = 'main'
    # Mapa za prevođenje ESP32 ključeva u JSON ključeve
    KEY_MAP = {
        'TCP/IP Address': 'ip_address', 'TCP/IP Port': 'port',
        'WIFI SSID': 'wifi_ssid', 'WIFI PASSWORD': 'wifi_password',
        'Ping Watchdog': 'ping_watchdog', 'Pump/Valve': 'pump_valve',
        'EMA Filter': 'ema_filter', 'TIMERON': 'timer_on', 'TIMEROFF': 'timer_off'
    }
    try:
        lines = raw_text.strip().split('\n')
        line = ""
        for line in lines:
            line = line.strip()
            if not line or line.startswith('----'): continue
            if line.startswith('ESP Thermostat State:'):
                current_section = 'thermostat'
                continue
            elif line.startswith('ESP Pin States:'):
                current_section = 'pins'
                continue
            if '=' not in line: continue
            key, value = line.split('=', 1)
            key = key.strip(); value = value.strip()
            if '*C' in value: value = value.split('*C')[0].strip()
            if 'min' in value: value = value.split('min')[0].strip()
            if key in KEY_MAP:
                js_key = KEY_MAP[key]
            else:
                js_key = key.lower().replace(' ', '_').replace('/', '_')
            if current_section == 'thermostat':
                thermostat_data[js_key] = value
            elif current_section == 'pins':
                if js_key.startswith('gpio'):
                    pin_broj = js_key.replace('gpio', '')
                    pin_states[pin_broj] = value
            else:
                main_data[js_key] = value
```

```
            main_data.update(thermostat_data)
            main_data['pins'] = pin_states
            return main_data
        except Exception as e:
            logging.error(f"GRESKA PARSERA: {e} na liniji: '{line}'")
            return {'error': str(e)}

# ---------------------------------------------------------------------
#  HELPER FUNKCIJE ZA IP RESOLVING I SLANJE
# ---------------------------------------------------------------------
def resolve_and_cache_ip(soba_id):
    with config_lock:
        if soba_id not in CONFIG['sobe']:
            return None
        soba_config = CONFIG['sobe'][soba_id]
        mdns_name = soba_config['mdns']
        port = soba_config['port']
    mdns_url = f"http://{mdns_name}:{port}/sysctrl.cgi"
    params = {'CMD': 'GET_IP_ADDRESS'}
    try:
        logging.info(f"RESOLVING: Tražim IP za {soba_id} ({mdns_name})...")
        r = requests.get(mdns_url, params=params, timeout=5)
        r.raise_for_status()
        match = re.search(r'TCP/IP Address = ([\d\.]+)', r.text)
        if match:
            ip_address = match.group(1).strip()
            logging.info(f"RESOLVED: Soba {soba_id} ({mdns_name}) je na IP {ip_address}")
            with config_lock:
                CONFIG['sobe'][soba_id]['cached_ip'] = ip_address
            return ip_address
        else:
            logging.warning(f"Neuspjelo parsiranje IP adrese za {soba_id}. Odgovor: {r.text}")
            return None
    except requests.exceptions.RequestException as e:
        logging.error(f"RESOLVE FAILED: Neuspješno kontaktiranje {mdns_name}. Greška:
{e}")
        with config_lock:
            if soba_id in CONFIG['sobe']:
                CONFIG['sobe'][soba_id]['cached_ip'] = None
        return None

def send_esp_command(soba_id, params, timeout=1.5):
    with config_lock:
```

```
        soba_config = CONFIG['sobe'].get(soba_id)
        if not soba_config:
            logging.error(f"FATAL: Pokušaj slanja komande na nepostojeću sobu {soba_id}")
            return None, "Greška: Soba nije pronađena u konfiguraciji."

        cached_ip = soba_config.get('cached_ip')
        port = soba_config['port']

    if not cached_ip:
        logging.warning(f"INFO: Komanda za {soba_id} odbijena (cached_ip=None).
Pozadinski task traži adresu.")
        threading.Thread(target=resolve_and_cache_ip, args=(soba_id,),
daemon=True).start()
        return None, "Uređaj se još traži (mDNS). Pokušajte ponovo za 10 sekundi."

    base_url = f"http://{cached_ip}:{port}/sysctrl.cgi"

    try:
        logging.info(f"Šaljem komandu: {base_url}  Params: {params}")
        r = requests.get(base_url, params=params, timeout=timeout)
        r.raise_for_status()

        cmd = params.get('CMD')
        if cmd == 'SET_PASSWORD':
            if "Password set OK" not in r.text:
                logging.warning(f"GREŠKA: Očekivan 'Password set OK', dobiveno:
{r.text.strip()}")
                return None, "Uređaj je odbio komandu za PIN."
        elif cmd == 'ESP_SET_PIN':
            if "PIN set HIGH" not in r.text:
                logging.warning(f"GREŠKA: Očekivan 'PIN set HIGH', dobiveno: {r.text.strip()}")
                return None, "Uređaj je odbio komandu za SET_PIN."
        elif cmd == 'ESP_RESET_PIN':
            if "PIN set LOW" not in r.text:
                logging.warning(f"GREŠKA: Očekivan 'PIN set LOW', dobiveno: {r.text.strip()}")
                return None, "Uređaj je odbio komandu za RESET_PIN."

        logging.info(f"ODGOVOR OK: {r.text.strip()}")
        return r, "ok" # USPJEH

    except requests.exceptions.RequestException as e:
        logging.warning(f"GREŠKA KONEKCIJE za {soba_id} na {base_url}: {e}. Pokrećem
reaktivni re-resolve.")
```

```
        with config_lock:
            if soba_id in CONFIG['sobe']:
                CONFIG['sobe'][soba_id]['cached_ip'] = None

        threading.Thread(target=resolve_and_cache_ip, args=(soba_id,),
daemon=True).start()

        return None, "Greška: Konekcija na uređaj nije uspjela. Adresa se osvježava u
pozadini. Pokušajte ponovo."

def find_soba_id_by_mdns(mdns_name):
    with config_lock:
        for soba_id, data in CONFIG.get('sobe', {}).items():
            if data.get('mdns') == mdns_name:
                return soba_id
    return None

# ---------------------------------------------------------------------
#  KORISNIČKE (GOST) RUTE
# ---------------------------------------------------------------------
@app.route('/')
def login_page():
    return render_template('login.html')

@app.route('/soba')
def soba_page():
    return render_template('soba.html')

@app.route('/api/login', methods=['POST'])
def api_login():
    data = request.json
    pin = data.get('pin')
    if not pin:
        return jsonify({'success': False, 'message': 'PIN nije poslan'}), 400

    try:
        with config_lock:
            for soba_id, soba_data in CONFIG.get('sobe', {}).items():
                if soba_data.get('guest_pin') == pin:
                    token_payload = {
                        'soba_id': soba_id,
                        'guest_pin': pin,
```

```
                'mdns': soba_data['mdns'],
                'port': soba_data['port'],
                'tip': 'gost',
                'exp': datetime.utcnow() + timedelta(hours=24)
            }
            token = jwt.encode(token_payload, app.config['SECRET_KEY'],
algorithm='HS256')
            response = make_response(jsonify({'success': True}))
            response.set_cookie('token', token, httponly=True, samesite='Strict',
max_age=86400)
            logging.info(f"USPJEŠAN LOGIN: Gost se prijavio za sobu {soba_id} (Ime:
{soba_data['ime']})")
            return response

        logging.warning(f"NEUSPJEŠAN LOGIN: Pogrešan PIN unesen: {pin}")
        return jsonify({'success': False, 'message': 'Pogrešan PIN'}), 401

    except Exception as e:
        logging.error(f"Greška u /api/login: {e}")
        return jsonify({'success': False, 'message': 'Greška servera'}), 500

@app.route('/api/control', methods=['POST'])
def api_control():
    token = request.cookies.get('token')
    if not token:
        return jsonify({'success': False, 'message': 'Niste prijavljeni'}), 401
    try:
        soba_data = jwt.decode(token, app.config['SECRET_KEY'], algorithms=['HS256'])
        if soba_data.get('tip') != 'gost':
            return jsonify({'success': False, 'message': 'Neispravan token'}), 401

        soba_id = soba_data.get('soba_id')
        pin_iz_tokena = soba_data.get('guest_pin')
        if not pin_iz_tokena:
            return jsonify({'success': False, 'message': 'Neispravan token (nedostaje pin).
Prijavite se ponovo.'}), 401

        with config_lock:
            if not soba_id or soba_id not in CONFIG['sobe']:
                return jsonify({'success': False, 'message': 'Konfiguracija sobe nije
pronađena.'}), 500
            trenutni_guest_pin = CONFIG['sobe'][soba_id].get('guest_pin')
            if trenutni_guest_pin != pin_iz_tokena:
```

```
            logging.warning(f"ODBIJENO: Token za sobu {soba_id} je nevažeći (PIN promijenjen). Korisnik izbačen.")
            return jsonify({'success': False, 'message': 'PIN za sobu je promijenjen. Molimo prijavite se ponovo.'}), 401

        data = request.json
        uredjaj = data.get('uredjaj'); vrijednost = data.get('vrijednost')

        with config_lock:
            soba_config = CONFIG['sobe'][soba_id]
            if not uredjaj or uredjaj not in soba_config.get('uredjaji', {}):
                return jsonify({'success': False, 'message': f'Uređaj "{uredjaj}" nije definiran'}), 400
            device_config = soba_config['uredjaji'][uredjaj]

        params = device_config.copy()
        komanda = params.get('CMD')

        if komanda == 'SET_PIN': params['VALUE'] = '1' if vrijednost else '0'
        elif komanda == 'SET_ROOM_TEMP': params['VALUE'] = str(int(round(float(vrijednost))))
        elif komanda in ['SET_THST_ON', 'SET_THST_OFF', 'SET_THST_HEATING', 'SET_THST_COOLING']:
            pass
        else: return jsonify({'success': False, 'message': f'Komanda {komanda} nije podržana'}), 500

        logging.info(f"GOST KONTROLA ({soba_id}): Uređaj '{uredjaj}' -> {params}")
        response, message = send_esp_command(soba_id, params, timeout=1.5)

        if response and response.ok:
            return jsonify({'success': True, 'message': 'Komanda poslana'})
        else:
            return jsonify({'success': False, 'message': message}), 500

    except jwt.ExpiredSignatureError:
        logging.info("GOST KONTROLA: Sesija istekla (ExpiredSignatureError)")
        return jsonify({'success': False, 'message': 'Sesija istekla, prijavite se ponovo'}), 401
    except Exception as e:
        logging.error(f"Greška u /api/control: {e}")
        return jsonify({'success': False, 'message': 'Greška servera'}), 500

# ------------------------------------------------------------------
```

```
#  ADMIN RUTE
# ------------------------------------------------------------------------
def provjeri_admin_token(token):
    if not token: return False
    try:
        data = jwt.decode(token, app.config['SECRET_KEY'], algorithms=['HS256'])
        return data.get('tip') == 'admin'
    except: return False

@app.route('/admin')
def admin_login_page():
    return render_template('admin_login.html')

@app.route('/admin/dashboard')
def admin_dashboard():
    if not provjeri_admin_token(request.cookies.get('admin_token')):
        return redirect(url_for('admin_login_page'))

    with config_lock:
        sobe_za_admina = {}
        for soba_id, data in CONFIG.get('sobe', {}).items():
            uredjaji = data.get('uredjaji', {})
            sobe_za_admina[soba_id] = {
                "ime": data.get('ime'),
                "mdns": data.get('mdns'),
                "port": data.get('port'),
                "guest_pin": data.get('guest_pin', 'N/A'),
                "termostat_id": uredjaji.get('termostat_set', {}).get('ID', 'N/A'),
                "pin_controller_id": uredjaji.get('pin_controller', {}).get('ID', 'N/A')
            }
    return render_template('admin.html', sobe=sobe_za_admina)

@app.route('/api/admin/login', methods=['POST'])
def api_admin_login():
    data = request.json
    password = data.get('password')
    if not password:
        return jsonify({'success': False, 'message': 'Lozinka nije poslana'}), 400
    if password == CONFIG.get('admin_password'):
        token_payload = { 'tip': 'admin', 'exp': datetime.utcnow() + timedelta(hours=8) }
        token = jwt.encode(token_payload, app.config['SECRET_KEY'], algorithm='HS256')
        response = make_response(jsonify({'success': True}))
        response.set_cookie('admin_token', token, httponly=True, samesite='Strict',
```

```
max_age=28800)
        logging.info("USPJEŠAN LOGIN: Administrator se prijavio.")
        return response
    else:
        logging.warning("NEUSPJEŠAN LOGIN: Pogrešna admin lozinka.")
        return jsonify({'success': False, 'message': 'Pogrešna lozinka'}), 401

@app.route('/api/admin/get_status', methods=['POST'])
def api_admin_get_status():
    if not provjeri_admin_token(request.cookies.get('admin_token')):
        return jsonify({'success': False, 'message': 'Niste prijavljeni'}), 401

    data = request.json
    mdns_name = data.get('mdns')
    if not mdns_name:
        logging.error("ADMIN GRESKA: /api/admin/get_status pozvan bez 'mdns'
parametra.")
        return jsonify({'success': False, 'message': 'Greška: Nedostaje mdns u zahtjevu.'}),
400

    soba_id = find_soba_id_by_mdns(mdns_name)

    if not soba_id:
        logging.warning(f"ADMIN: Primljen zahtjev za nepoznat mDNS: {mdns_name}")
        return jsonify({'success': False, 'message': f'Greška: mDNS ime {mdns_name} nije
pronađeno u config_server.json.'}), 400

    params = {'CMD': 'GET_STATUS'}
    logging.info(f"ADMIN: Tražim GET_STATUS za {soba_id} (mDNS: {mdns_name})")

    response, message = send_esp_command(soba_id, params, timeout=5)

    if response and response.ok:
        parsed_data = parse_get_status(response.text)
        logging.info(f"ADMIN: Uspješno parsiran status za {soba_id}")
        return jsonify({'success': True, 'status': parsed_data})
    else:
        logging.error(f"ADMIN GRESKA: Dohvaćanje statusa za {soba_id} nije uspjelo.")
        return jsonify({'success': False, 'message': message}), 500

@app.route('/api/admin/set_settings', methods=['POST'])
def api_admin_set_settings():
    if not provjeri_admin_token(request.cookies.get('admin_token')):
```

```
        return jsonify({'success': False, 'message': 'Niste prijavljeni'}), 401

    data = request.json
    mdns_name = data.get('mdns')
    termostat_id = data.get('termostat_id')
    settings = data.get('settings')

    if not mdns_name:
         return jsonify({'success': False, 'message': 'Greška: Nedostaje mdns.'}), 400

    soba_id = find_soba_id_by_mdns(mdns_name)

    if not all([soba_id, settings]):
        logging.error(f"ADMIN SET GRESKA: Poziv bez 'settings' ili 'soba_id' nije pronađen
(mdns: {mdns_name})")
        return jsonify({'success': False, 'message': 'Nedostaju ključni podaci (settings) ili
mDNS nije pronađen.'}), 400

    commands_to_send = []

    if 'mode' in settings:
        mode = settings['mode'].upper()
        if mode == 'HEATING': commands_to_send.append({'CMD': 'TH_HEATING'})
        elif mode == 'COOLING': commands_to_send.append({'CMD': 'TH_COOLING'})
        elif mode == 'ON': commands_to_send.append({'CMD': 'TH_ON'})
        elif mode == 'OFF': commands_to_send.append({'CMD': 'TH_OFF'})
    if 'mdns' in settings:
        commands_to_send.append({'CMD': 'SET_MDNS_NAME', 'MDNS': settings['mdns']})
    if 'port' in settings:
        commands_to_send.append({'CMD': 'SET_TCPIP_PORT', 'PORT': settings['port']})
    if termostat_id and termostat_id != 'N/A' and 'setpoint' in settings:
        setpoint = str(int(round(float(settings['setpoint']))))
        commands_to_send.append({'CMD': 'SET_ROOM_TEMP', 'ID': termostat_id, 'VALUE':
setpoint})
    if 'wifi_ssid' in settings and 'wifi_password' in settings:
        cmd = {'CMD': 'SET_SSID_PSWRD', 'SSID': settings['wifi_ssid']}
        if settings['wifi_password']: cmd['PSWRD'] = settings['wifi_password']
        commands_to_send.append(cmd)
    if 'ping_watchdog' in settings:
        if settings['ping_watchdog']: commands_to_send.append({'CMD': 'PINGWDG_ON'})
        else: commands_to_send.append({'CMD': 'PINGWDG_OFF'})
    if 'diff' in settings and settings['diff']:
        try:
```

```
            diff_value = int(float(settings['diff']) * 10)
            commands_to_send.append({'CMD': 'TH_DIFF', 'VALUE': diff_value})
        except ValueError:
            logging.warning(f"ADMIN SET: Pogrešna 'diff' vrijednost: {settings['diff']}")
    if 'ema_filter' in settings and settings['ema_filter']:
        try:
            ema_value = int(float(settings['ema_filter']) * 10)
            commands_to_send.append({'CMD': 'TH_EMA', 'VALUE': ema_value})
        except ValueError:
             logging.warning(f"ADMIN SET: Pogrešna 'ema_filter' vrijednost:
{settings['ema_filter']}")
    if 'timer_on' in settings and 'timer_off' in settings:
         commands_to_send.append({'CMD': 'SET_TIMER', 'TIMERON': settings['timer_on'],
'TIMEROFF': settings['timer_off']})



    success_count = 0
    errors = []
    logging.info(f"ADMIN SET: Soba {soba_id} - Počinjem slanje {len(commands_to_send)}
komandi...")
    for params in commands_to_send:
        logging.info(f"ADMIN SET ({soba_id}): Šaljem {params}...")
        response, message = send_esp_command(soba_id, params)
        if response and response.ok:
            success_count += 1
        else:
            errors.append(f"Komanda {params.get('CMD')} nije uspjela: {message}")

    if success_count == len(commands_to_send):
        logging.info(f"ADMIN SET ({soba_id}): Uspješno poslane sve komande.")
        return jsonify({'success': True, 'message': f'Sve postavke ({success_count}) uspješno
poslane.'})
    else:
        logging.warning(f"ADMIN SET ({soba_id}): Neke komande nisu uspjele. Poslano
{success_count}/{len(commands_to_send)}.")
        return jsonify({'success': False, 'message': f'Neke komande nisu uspjele. Poslano
{success_count}/{len(commands_to_send)}. Prva greška: {errors[0]}'})

# ---------------------------------------------------------------------
#  IFTTT / ALEXA WEBHOOK RUTA
# ---------------------------------------------------------------------
@app.route('/api/ifttt/control', methods=['POST'])
def api_ifttt_control():
```

```
    data = request.json

    # 1. Sigurnosna Provjera Ključa
    secret_key = data.get('api_key')
    with config_lock:
        # Koristimo "external_api_key" kako je definirano u Vašem config_server.json
        if secret_key != CONFIG.get('external_api_key'):
            logging.warning("IFTTT: Odbijen neispravan API ključ.")
            return jsonify({'success': False, 'message': 'Neispravan API ključ'}), 401

    # 2. Dohvat Komande (ha-bridge šalje room_id, uredjaj i vrijednost)
    room_id = data.get('room_id')
    uredjaj = data.get('uredjaj')
    vrijednost_str = str(data.get('vrijednost', 'off')).lower() # 'on'/'off' ili broj

    # 3. Validacija Komande
    if not all([room_id, uredjaj]):
        return jsonify({'success': False, 'message': 'Nedostaje room_id ili uredjaj.'}), 400

    # Pretvori string 'on'/'off' u boolean True/False ili float
    if vrijednost_str == 'on':
        vrijednost = True
    elif vrijednost_str == 'off':
        vrijednost = False
    else:
        try:
            vrijednost = float(vrijednost_str)
        except ValueError:
            return jsonify({'success': False, 'message': 'Vrijednost nije valjana (očekivano
on/off/broj).'}), 400

    # 4. Izvrši Komandu
    try:
        with config_lock:
            soba_config = CONFIG['sobe'].get(room_id)
            if not soba_config or uredjaj not in soba_config.get('uredjaji', {}):
                return jsonify({'success': False, 'message': f'Uređaj "{uredjaj}" nije definiran za
sobu {room_id}.'}), 400

            device_config = soba_config['uredjaji'][uredjaj]
            params = device_config.copy()
            komanda = params.get('CMD')
```

```
    # ------------------------------------------------------------------
    # POČETAK ISPRAVLJENOG BLOKA (OGRANIČAVANJE TEMPERATURE)
    # ------------------------------------------------------------------
    if komanda == 'SET_PIN':
        params['VALUE'] = '1' if vrijednost else '0'

    elif komanda == 'SET_ROOM_TEMP':
        try:
            # Vrijednost stiže od Alexe kao broj (npr. 22 ili 100).
            temp_trazena = float(vrijednost)

            # --- OVDJE JE OGRANIČAVANJE (Clamping) ---
            MIN_TEMP = 18  # Minimalna dozvoljena temperatura
            MAX_TEMP = 30  # Maksimalna dozvoljena temperatura

            final_value = int(round(temp_trazena))

            if final_value < MIN_TEMP:
                final_value = MIN_TEMP
                logging.info(f"IFTTT/ALEXA: Vrijednost {temp_trazena} je ISPOD minimuma.
Postavljam na {final_value}°C")
            elif final_value > MAX_TEMP:
                final_value = MAX_TEMP
                logging.info(f"IFTTT/ALEXA: Vrijednost {temp_trazena} je IZNAD maksimuma.
Postavljam na {final_value}°C")
            else:
                logging.info(f"IFTTT/ALEXA: Postavljam temperaturu na {final_value}°C")

            params['VALUE'] = str(final_value)

        except ValueError:
            logging.error(f"IFTTT: Nije moguće pretvoriti vrijednost '{vrijednost}' u broj za
termostat.")
            params['VALUE'] = str(MIN_TEMP) # Vrati na minimum ako je greška

    else:
        return jsonify({'success': False, 'message': f'Komanda {komanda} nije podržana
preko IFTTT.'}), 500
    # ------------------------------------------------------------------
    # KRAJ ISPRAVLJENOG BLOKA
    # ------------------------------------------------------------------

    logging.info(f"IFTTT KONTROLA ({room_id}): Uređaj '{uredjaj}' -> {params}")
```

```
        response, message = send_esp_command(room_id, params, timeout=3)

        if response and response.ok:
            return jsonify({'success': True, 'message': 'Komanda poslana IFTTT-om'})
        else:
            return jsonify({'success': False, 'message': message}), 500

    except Exception as e:
        logging.error(f"Greška u /api/ifttt/control: {e}")
        return jsonify({'success': False, 'message': 'Greška servera'}), 500

# ---------------------------------------------------------------------
#  API RUTA ZA DOHVAT STATUSA ZA GOSTA
# ---------------------------------------------------------------------
def parse_get_pins_response(response_text):
    match = re.search(r'Pins States = ([\d]+)', response_text)
    if match:
        return match.group(1).strip()
    return None

@app.route('/api/status')
def api_get_guest_status():
    token = request.cookies.get('token')
    if not token:
        return jsonify({'success': False, 'message': 'Niste prijavljeni'}), 401
    try:
        soba_data = jwt.decode(token, app.config['SECRET_KEY'], algorithms=['HS256'])
        soba_id = soba_data.get('soba_id')

        # (Ponovljena sigurnosna provjera pina iz tokena)
        pin_iz_tokena = soba_data.get('guest_pin')
        with config_lock:
            if not soba_id or soba_id not in CONFIG['sobe']:
                return jsonify({'success': False, 'message': 'Konfiguracija sobe nije
pronađena.'}), 500
            trenutni_guest_pin = CONFIG['sobe'][soba_id].get('guest_pin')
            if trenutni_guest_pin != pin_iz_tokena:
                return jsonify({'success': False, 'message': 'PIN za sobu je promijenjen. Molimo
prijavite se ponovo.'}), 401

            soba_config = CONFIG['sobe'][soba_id]
            termostat_id = soba_config.get('uredjaji', {}).get('termostat_set', {}).get('ID')
```

```
        pin_controller_id = soba_config.get('uredjaji', {}).get('pin_controller', {}).get('ID')

    if not termostat_id or not pin_controller_id:
        logging.error(f"GRESKA STATUSA ({soba_id}): 'termostat_set' ili 'pin_controller' nije definiran u config_server.json.")
        return jsonify({'success': False, 'message': 'Greška: Uređaji u sobi nisu ispravno mapirani.'}), 500

    # 1. Dohvati status termostata
    params_temp = {'CMD': 'GET_THST_STATUS', 'ID': termostat_id}
    response_temp, msg_temp = send_esp_command(soba_id, params_temp, timeout=3)

    # 2. Dohvati status pinova
    params_pins = {'CMD': 'ESP_GET_PINS', 'ID': pin_controller_id}
    response_pins, msg_pins = send_esp_command(soba_id, params_pins, timeout=3)

    if not response_temp or not response_pins:
        logging.warning(f"GRESKA STATUSA ({soba_id}): Nije dobiven odgovor od oba uređaja. Temp: {msg_temp}. Pin: {msg_pins}")
        return jsonify({'success': False, 'message': 'Greška: Nije moguće dohvatiti status sa uređaja (offline?).'}), 500

    # 3. Parsiraj oba odgovora
    try:
        match_temp = re.search(r'Room Temp = ([\d\.]+)', response_temp.text)
        match_setpoint = re.search(r'Setpoint = ([\d\.]+)', response_temp.text)
        match_mode = re.search(r'Mode = (\w+)', response_temp.text)

        status_sobe = {
            'room_temp': float(match_temp.group(1)) if match_temp else 0,
            'setpoint': float(match_setpoint.group(1)) if match_setpoint else 0,
            'mode': match_mode.group(1).upper() if match_mode else 'OFF'
        }

        pin_states_str = parse_get_pins_response(response_pins.text)
        if not pin_states_str:
            raise Exception(f"Nije moguće parsirati ESP_GET_PINS odgovor: {response_pins.text}")

        status_pinova = {}
        with config_lock:
            uredjaji_sobe = CONFIG['sobe'][soba_id].get('uredjaji', {})
```

```
            for uredjaj_key, uredjaj_data in uredjaji_sobe.items():
                if uredjaj_data.get('CMD') == 'SET_PIN':
                    try:
                        pin_index = int(uredjaj_data.get('PIN'))
                        if 0 < pin_index <= len(pin_states_str):
                            status_pinova[uredjaj_key] = (pin_states_str[pin_index-1] == '1')
                    except Exception as e_pin:
                        logging.warning(f"GRESKA STATUSA ({soba_id}): Neuspješno čitanje pina {uredjaj_data.get('PIN')} za {uredjaj_key}. Greška: {e_pin}")

            final_status = {
                'success': True,
                'soba': status_sobe,
                'uredjaji': status_pinova
            }
            return jsonify(final_status)

        except Exception as e:
            logging.error(f"GRESKA PARSIRANJA STATUSA ({soba_id}): {e}. Temp: [{response_temp.text}]. Pins: [{response_pins.text}]")
            return jsonify({'success': False, 'message': 'Greška: Server nije mogao razumjeti odgovor uređaja.'}), 500

    except jwt.ExpiredSignatureError:
        return jsonify({'success': False, 'message': 'Sesija istekla, prijavite se ponovo'}), 401
    except Exception as e:
        logging.error(f"Generalna greška u /api/status: {e}")
        return jsonify({'success': False, 'message': f'Greška servera: {e}'}), 500

# ---------------------------------------------------------------------
#  POZADINSKI PROCESI (Osvježavanje IP adresa)
# ---------------------------------------------------------------------
def background_resolver_task():
    """(POZADINSKI TASK) Svakih 5 minuta provjerava sve IP adrese."""
    logging.info("POZADINSKI RESOLVER: Pokrenut.")
    time.sleep(15)
    while True:
        try:
            logging.info("PROAKTIVNI RESOLVER: Počinjem provjeru IP adresa...")
            with config_lock:
                soba_ids = list(CONFIG.get('sobe', {}).keys())

            for soba_id in soba_ids:
```

```
                resolve_and_cache_ip(soba_id)
                time.sleep(1)

            logging.info("PROAKTIVNI RESOLVER: Provjera IP adresa završena. Spavam 5 minuta.")
            time.sleep(300)

        except Exception as e:
            logging.error(f"Greška u pozadinskom resolveru: {e}")
            time.sleep(60)

# ---------------------------------------------------------------------
#  POKRETANJE SERVERA
# ---------------------------------------------------------------------
if __name__ == '__main__':
    load_config()

    logging.info("Pokrećem inicijalno mapiranje IP adresa U POZADINI...")
    with config_lock:
        initial_soba_ids = list(CONFIG.get('sobe', {}).keys())

    def initial_map():
        for soba_id in initial_soba_ids:
            resolve_and_cache_ip(soba_id)
            time.sleep(0.5)
        logging.info("Inicijalno mapiranje završeno.")

    initial_map_thread = threading.Thread(target=initial_map, daemon=True)
    initial_map_thread.start()

    resolver_thread = threading.Thread(target=background_resolver_task, daemon=True)
    resolver_thread.start()

    print("--- Pokrećem Toplik Service (PRODUKCIJA Faza 7.0) ---")
    print("--- Server radi na [http://0.0.0.0:5000](http://0.0.0.0:5000) ---")
    serve(app, host='0.0.0.0', port=5000)
```

5. **Kreirajte Ostale Fajlove:**
    - nano login.html [cite: login.html] (zalijepite kod)
    - nano soba.html [cite: soba.html] (zalijepite kod)
    - nano admin_login.html [cite: admin_login.html] (zalijepite kod)
    - nano admin.html [cite: admin.html] (zalijepite kod)

- mkdir static i dodajte Vaše CSS/JS fajlove unutra.

Faza 3: Postavljanje ha-bridge (Alexa Emulator)

Ovaj softver će se pretvarati da je Philips Hue bridge, primati komande od Alexe i prosljeđivati ih Vašem server.py (koji smo podesili u Fazi 2).

1. **Kreirajte Direktorij:**
   mkdir /home/admin/habridge
   cd /home/admin/habridge

2. Preuzmite ha-bridge:
   (Koristimo tačnu verziju java11 koju ste originalno preuzeli)
   wget
   https://github.com/bwssytems/ha-bridge/releases/download/v5.4.1/ha-bridge-5.4.1-java11.jar

3. Kreirajte data Direktorij:
   ha-bridge zahtijeva ovaj folder da bi sačuvao svoju bazu uređaja (device.db) [cite: image_1c8cdc.png].
   mkdir data

Faza 4: Kreiranje Automatskih Servisa

Kreiramo "čuvarske" (systemd) servise koji će automatski pokrenuti oba servera pri paljenju i restartovati ih ako padnu.

1. **Kreirajte toplik.service (Vaš Python server):**
   sudo nano /etc/systemd/system/toplik.service

   Zalijepite ovaj sadržaj:
   [Unit]
   Description=Toplik Service Python Backend
   After=network.target

   [Service]
   User=admin
   Group=admin
   WorkingDirectory=/home/admin/ToplikService/
   ExecStart=/usr/bin/python3 -m waitress --port=5000 server:app
   Restart=always
   RestartSec=5

[Install]
WantedBy=multi-user.target

Sačuvajte (Ctrl+O, Enter, Ctrl+X).

2. **Kreirajte habridge.service (Alexa Emulator):**
   sudo nano /etc/systemd/system/habridge.service

   Zalijepite ovaj sadržaj. **Ovo je ključna ispravka** – dodajemo -Dserver.port=80 (da radi na standardnom portu) i -Ddisable.mDNS=true (da spriječimo "šum" na mreži koji smo vidjeli u logovima).
   [Unit]
   Description=HA Bridge (Alexa Emulator)
   After=network.target toplik.service
   BindsTo=toplik.service

   [Service]
   User=admin
   Group=admin
   WorkingDirectory=/home/admin/habridge/
   ExecStart=/usr/bin/java -Dha.config.dir=./data -Dserver.port=80 -Ddisable.mDNS=true -jar ha-bridge-5.4.1-java11.jar
   Restart=always
   RestartSec=5

   [Install]
   WantedBy=multi-user.target

   Sačuvajte (Ctrl+O, Enter, Ctrl+X).
3. **Aktivirajte Oba Servisa:**
   sudo systemctl daemon-reload
   sudo systemctl enable toplik.service
   sudo systemctl enable habridge.service

   # Pokreni ih (ili restartuj ako već rade)
   sudo systemctl restart toplik.service
   sudo systemctl restart habridge.service

   Provjerite status sa sudo systemctl status habridge.service. Trebali biste vidjeti active (running).

Faza 5: Konfiguracija ha-bridge Uređaja

Sada kada oba servera rade, moramo ha-bridge-u reći koje komande da šalje Vašem

server.py.

1. Otvorite ha-bridge sučelje u browseru: http://192.168.88.70 (radi na portu 80).
2. Idite na tab **"Add/Edit"**.

UREĐAJ 1: Chandelier 505 (Prekidač)

- **Name:** Chandelier 505 (*Ovo ime će Alexa prepoznati*)
- **Device Type:** Ostavite Switch
- **On Items:**
  - **Type:** httpDevice
  - **Http Verb:** POST
  - **Http Content Type:** application/json
  - **Target Item:** http://127.0.0.1:5000/api/ifttt/control
  - **Http Body:**
    {"api_key": "KLJUC-ZA-HOTELSKI-SOFTVER-98765", "room_id": "505", "uredjaj": "light_luster", "vrijednost": "on"}

- **Off Items:**
  - **Type:** httpDevice
  - **Http Verb:** POST
  - **Http Content Type:** application/json
  - **Target Item:** http://127.0.0.1:5000/api/ifttt/control
  - **Http Body:**
    {"api_key": "KLJUC-ZA-HOTELSKI-SOFTVER-98765", "room_id": "505", "uredjaj": "light_luster", "vrijednost": "off"}

- **Map Type (KLJUČNO):**
  - Označite (čekirajte) **hueDevice**.
- Kliknite **"Add Bridge Device"**.

UREĐAJ 2: Thermostat 505 (Termostat)

Ovo je ključna ispravka za rješavanje problema "percent".

- **Name:** Thermostat 505
- **Device Type (KLJUČNO):** Postavite na **Thermostat**
- **On Items / Off Items:** Ostavite prazno.
- **Dim Items (Ovo je za temperaturu):**
  - **Type:** httpDevice
  - **Http Verb:** POST
  - **Http Content Type:** application/json
  - **Target Item:** http://127.0.0.1:5000/api/ifttt/control
  - **Http Body:**
    {"api_key": "KLJUC-ZA-HOTELSKI-SOFTVER-98765", "room_id": "505", "uredjaj": "termostat_set", "vrijednost": "${intensity.percent}"}

- **Map Type (KLJUČNO):**
  - Označite (čekirajte) **hueDevice**.
- Kliknite **"Add Bridge Device"**.

(Ponavljate Fazu 5 za sve ostale prekidače (Device Type: Switch) i termostate (Device Type: Thermostat) koje želite dodati).

Faza 6: Povezivanje sa Alexom (Finalni Proces)

Ovo je najosjetljiviji dio, jer Alexa "pamti" stare greške. Moramo je natjerati na "čisti" start.

1. **Obrišite Stare Uređaje (u Alexa Aplikaciji):**
   - Otvorite Vašu **Alexa Aplikaciju**.
   - Idite na **"Devices"** (Uređaji).
   - Pronađite **SVE** stare uređaje koje ste pokušavali dodati ("Chandelier 505", "Thermostat 505", itd.) i **obrišite ih** (Settings -> Delete/Forget).
2. **Restartujte Echo Dot (Ključni Korak za Brisanje Keša):**
   - **Isključite Vaš Echo Dot (192.168.88.81) iz struje.**
   - Sačekajte 30 sekundi.
   - Uključite ga ponovo u struju i sačekajte da se sistem digne.
3. **Pokrenite Discovery:**
   - Idite na ha-bridge sučelje (http://192.168.88.70) -> tab **"Bridge Control"**.
   - **Važno:** Ako vidite listu korisnika (User list), obrišite sve stare korisnike pritiskom na (X). Ako ne vidite, ignorišite ovaj korak.
   - Kliknite **"Press Link Button"**.
   - **ODMAH** recite Alexi: **"Alexa, discover devices"**.

Ovaj put, Alexa će pronaći sve uređaje koje ste unijeli u Fazi 5, i **ispravno će prepoznati termostat kao termostat.**

Faza 7: Podešavanje Alexa Grupa (Kontekstualne Komande)

Ovo rješava Vaš zahtjev da **ne morate govoriti broj sobe**.

1. Otvorite Vašu **Alexa Aplikaciju**.
2. Idite na tab **"Devices"** (Uređaji) i kliknite na **"+"** (plus) u gornjem desnom uglu.
3. Odaberite **"Add Group"**.
4. Dajte grupi ime, npr. **"Soba 505"**.
5. U sekciji **"Alexa Devices"**, odaberite Echo Dot koji se **fizički nalazi** u Sobi 505.
6. U sekciji **"Devices"**, odaberite uređaje koji pripadaju toj sobi (npr. "Chandelier 505" i "Thermostat 505").
7. Kliknite **"Save"**.

Faza 8: Finalne Glasovne Komande (Korištenje)

Sada, kada stojite u Sobi 505 (blizu Echo Dot-a koji je u toj grupi):

- **Za Prekidače (bez broja sobe):**
  - "Alexa, turn on Chandelier."
  - "Alexa, turn on the light."
- **Za Termostat (Bez "Percent"):**
  - **"Alexa, set thermostat to 22."**
  - **"Alexa, set thermostat to 30."**
  - **"Alexa, set thermostat to 100."** (Skripta iz Faze 2 će ovo automatski ograničiti na 30°C)
  - **"Alexa, set thermostat to 10."** (Skripta će ovo ograničiti na 18°C)

Dodatak A: Istorija Debugovanja (Samo za evidenciju)

Tokom instalacije, isproban je diyHue servis, ali je izazvao konflikte. Ovo su koraci koji su poduzeti da se on ukloni.

- **Faza A: Instalacija diyHue (Neuspješno)**
  - diyHue je instaliran pomoću curl -fsSL https://get.d.iyhue.org -o install.sh i sudo ./install.sh.
  - Servis je radio, ali je ha-bridge odabran kao jednostavnije rješenje.
- **Faza B: Uklanjanje diyHue Servisa (Ispravan Način)**
  - diyHue je kreirao servis hue-emulator.service.
  - Servis je zaustavljen: sudo systemctl stop hue-emulator.service
  - Servis je onemogućen: sudo systemctl disable hue-emulator.service
  - Fajlovi su obrisani: sudo rm -rf /opt/hue-emulator i sudo rm /etc/systemd/system/hue-emulator.service
  - Ovo je oslobodilo port 80 za ha-bridge.