به نام خدا

پروژه درس داده کاوی:

Smart Home Assistant

استاد:

دکتر کیانی

تهیه کنندگان:

فاطمه دلال

تارا روانلو

تابستان 1404

Agent.py:

ما در این فایل منطق اصلی "عامل هوشمند خانه‌ی هوشمند" را پیاده‌سازی کرده‌ایم. این عامل، ورودی کاربر (متنی) را دریافت کرده، آن را پردازش میکند، دستورات مربوطه را استخراج میکند، و نهایتاً آن‌ها را اجرا کرده و خروجی را تولید میکند. این فایل در واقع مغز سیستم ماست.

نقش فایل agent.py در پروژه:

فایل agent.py نقش یک عامل (Agent) رو بازی میکنه که:

1. فرمان کاربر (متنی) رو دریافت میکنه.
2. اون رو اصلاح املایی و دستوری میکنه.
3. کلمات کلیدی و مقادیر رو استخراج میکنه.
4. فرمان رو به دستورهای قابل اجرا روی وسایل خانه هوشمند ترجمه میکنه.
5. دستورها رو اجرا میکنه و پاسخ مناسب تولید میکنه.

ماژول‌ها و کتابخانه‌های استفاده‌شده:

در ابتدای فایل، کتابخانه‌های موردنیاز را ایمپورت کرده‌ایم:

- defaultdict:برای نگهداری کلیدواژه‌ها به‌صورت دسته‌بندی‌شده
- matplotlib.colors: برای تشخیص اینکه آیا یک کلمه رنگ معتبر است یا نه
- re: برای استفاده از regex در استخراج اعداد و واحدها
- json:برای بارگذاری فایل کلیدواژه‌ها
- os: برای ایجاد مسیر درست به فایل keywords.json

سپس فایل‌های خودمان را نیز ایمپورت کرده‌ایم که این فایل‌ها هرکدام عملکرد خاصی دارند:

- home.py: شامل توابع کنترل وسایل (روشن/خاموش کردن، تنظیم رنگ، دما و...)
- correct_spelling.py: برای تصحیح املای دستورات ورودی کاربر
- **ساختار اصلی عامل:**

1. map_keywords(text):

تابعی برای نگاشت کلمات موجود در متن کاربر به کلیدواژه‌هایی از پیش تعریف‌شده در فایل keywords.json.

```
def map_keywords(text):  1 usage
    mapped = defaultdict(list)
    for key, words in KEYWORDS.items():
        for w in words:
            if w in text:
                mapped[key].append(w)
    return mapped
```

2. is_color(word):

بررسی میکند که آیا یک کلمه یک رنگ معتبر (بر اساس CSS4) هست یا نه.

```
def is_color(word):  1 usage
    if word.lower() in m_colors.CSS4_COLORS:
        return True
    return False
```

3. extract_number_and_unit(text):

از طریق عبارات منظم (Regex) ، اعداد و واحدها (درصد، درجه و ...) را از متن کاربر استخراج میکند.

```python
def extract_number_and_unit(text):  1 usage
    patterns = [
        r'\bto\s+(\d+)\s*(percent|degrees|%)?',
        r'\bby\s+(\d+)\s*(percent|degrees|%)?',
        r'\b(\d+)\s*(percent|degrees|%)?',
    ]
    for pattern in patterns:
        match = re.search(pattern, text, re.IGNORECASE)
        if match:
            number = int(match.group(1))
            unit = match.group(2) if match.group(2) else None
            return number, unit
    return None, None
```

4. parse_command(correct_input):

این تابع، دستور کاربر را تجزیه میکند:

- ابتدا متن را به چند فرمان (با جداکننده "و" یا "and") تقسیم میکند.
- سپس با استفاده از کلیدواژه‌ها، نوع دستگاه، موقعیت مکانی، نوع عملیات، و مقدار مورد نظر را استخراج میکند.
- نتیجه را به صورت لیستی از دیکشنری‌ها برمیگرداند.

```python
if unit is None and "brightness" in command:
    unit = "%"
elif unit is None and "temperature" in command:
    unit = "degrees"

if "lights" in mapped:
    device = "lights"
elif "set_temperature" in mapped:
    device = "temperature"
elif "tv" in mapped or "set_channel" in mapped:
    device = "tv"
    if not location:
        location = "living_room"
elif "get_time" in mapped:
    action = "get_time"
elif "get_date" in mapped:
    action = "get_date"
elif "get_weather" in mapped:
    action = "get_weather"
elif "get_news" in mapped:
    action = "get_news"
```

5. execute_command(instructions):

این تابع بر اساس دستوراتی که parse_command تولید کرده، توابع متناظر را از home.py فراخوانی میکند و پاسخ‌ها را جمع‌آوری میکند.

```
for instruction in instructions:
    try:
        device = instruction["device"]
        location = instruction["location"]
        action = instruction["action"]
        value = instruction["value"]

        if action == "turn_on" and device is not None:
            responses.append(turn_on(device, location))

        elif action == "turn_off" and device is not None:
            responses.append(turn_off(device, location))

        elif action == "set_color_light" and device is not None and value:
            responses.append(str(turn_on(device, location) + " and " + set_color_light(device, location, value)))

        elif action == "set_brightness_light" and device is not None and value is not None:
            responses.append(str(turn_on(device, location) + " and " + set_brightness_light(device, location, value)))

        elif action == "set_temperature" and device == "temperature" and value is not None:
            responses.append(str(turn_on(device, location) + " and " + set_temperature(location, value)))
```

6. get_output(responses):

پاسخ‌ها را به صورت یک متن نهایی ترکیب میکند و خروجی آماده‌ی نمایش تولید میکند.

```
get_output(responses):  1 usage
return "\n".join(responses)
```

7. smart_home_agent(user_input):

تابع اصلی عامل ماست:

- اول متن ورودی کاربر را اصلاح املایی میکند.
- سپس آن را تجزیه کرده و به دستورات تبدیل میکند.
- دستورات را اجرا میکند.
- در نهایت خروجی را برمیگرداند.

```
smart_home_agent(user_input):  4 usages
correct_input = correct_spelling_and_grammar(user_input)
instructions = parse_command(correct_input)
responses = execute_command(instructions)
return get_output(responses)
```

نکات کلیدی در طراحی عامل:

- طراحی این عامل به صورت ماژولار انجام شده تا بخش‌های مربوط به تشخیص صدا، اصلاح املایی، کنترل وسایل و استخراج اطلاعات از هم جدا باشند.
- امکان گسترش سیستم به سادگی با اضافه کردن کلیدواژه‌های جدید یا توابع کنترلی جدید وجود دارد.
- طراحی طوری است که هم دستورات ترکیبی (مثلاً "تلویزیون رو روشن کن و چراغ‌ها رو کم کن") و هم دستورات تکی پشتیبانی می‌شوند.

AGENT_TEST.py:

این فایل برای تست عملکرد عامل هوشمند ما (smart_home_agent) طراحی شده است. هدف از این فایل بررسی این است که آیا عامل میتواند انواع مختلفی از دستورات طبیعی را پردازش کرده، آن‌ها را تفسیر کرده و پاسخ مناسب را تولید کند یا نه.

ساختار فایل:

در ابتدا، تنها تابع مورد نیاز را از فایل agent.py ایمپورت میکنیم:

تابع smart_home_agent() همان عاملی است که ورودی کاربر را دریافت می‌کند، آن را تجزیه کرده، دستور مناسب را استخراج کرده، و در نهایت خروجی تولید می‌کند.

اجرای تست‌ها:

در بخش if __name__ == '__main__'، لیستی از دستورات نمونه تعریف شده است که ما آن‌ها را به عنوان تست به عامل میدهیم:

```python
if __name__ == '__main__':
    test_cases = [
        "turn on the lights in the kitchen",
        "turn off the lights in the bathroom",
        "set the color of the lights to blue",
        "set brightness to 60 in room1",
        "turn on heating in room2",
        "set the temperature to 25 degrees in the living room",
        "change the TV channel to 4",
        "get the weather",
        "what time is it",
        "get the news",
        "turn off lights and set brightness to 70 in kitchen"
    ]
```

این دستورات طیف متنوعی از امکانات سیستم را پوشش میدهند:

- کنترل وسایل (روشن/خاموش کردن چراغ‌ها و بخاری)
- تنظیم پارامترها (رنگ چراغ، دما، شدت نور)
- پرسش اطلاعاتی (وضعیت آب و هوا، ساعت، اخبار)
- دستورات ترکیبی (مثلاً خاموش کردن و تنظیم نور با هم)

سپس، هر تست به تابع عامل داده میشود و نتیجه چاپ میشود:

```python
for test in test_cases:
    print("Input:", test)
    response = smart_home_agent(test)
    print("Response:", response)
    print("-" * 60)
```

هدف فایل تست:

هدف این فایل موارد زیر است:

1. بررسی صحت عملکرد عامل هوشمند: آیا هر ورودی منجر به یک پاسخ درست و منطقی میشود؟
2. پوشش دادن سناریوهای متنوع: از دستورات ساده گرفته تا ترکیبی، با پارامتر یا بدون پارامتر.
3. آزمایش ماژولار بودن سیستم: اطمینان از این که ماژول‌های مختلف مثل کنترل نور، گرمایش، تلویزیون، آب و هوا و غیره، درست با هم کار میکنند.
4. شناسایی ایرادات احتمالی: اگر ورودی‌ای منجر به خطا یا پاسخ اشتباه شود، به راحتی قابل پیگیری است.

correct_spelling.py:

این فایل مسئول تصحیح خودکار املای کلمات و گرامر جملات انگلیسی است. ما از یک مدل پیش‌آموزش دیده‌ی زبان طبیعی استفاده کرده‌ایم که مبتنی بر معماری Seq2Seq (توالی به توالی) است و عملکرد دقیقی در تصحیح متون زبان انگلیسی دارد.

ایمپورت‌ها:

- Transformers: از کتابخانه معروف Hugging Face برای استفاده از مدل‌های از پیش آموزش‌دیده
- AutoTokenizer: توکنایزر هوشمند برای تبدیل متن به ورودی عددی قابل فهم برای مدل
- AutoModelForSeq2SeqLM: مدل زبانی مبتنی بر Seq2Seq برای وظایف مانند ترجمه، خلاصه سازی، و اصلاح گرامر و املا
- Torch: برای انجام inference بدون نیاز به یادگیری مجدد مدل

تابع اصلی correct_spelling_and_grammar(text) :

این تابع یک ورودی متنی به زبان انگلیسی دریافت میکند و نسخه اصلاح‌شده‌ی آن را بازمیگرداند.

```
def correct_spelling_and_grammar(text):
    if not text:
        return ""
    inputs = tokenizer(text, return_tensors="pt", max_length=128, truncation=True)
    with torch.no_grad():
        outputs = model.generate(**inputs, max_length=128, num_beams=5, early_stopping=True)
    corrected_text = tokenizer.decode(outputs[0], skip_special_tokens=True)
    return corrected_text
```

توضیح:

1. **توکنایز کردن متن ورودی**
   متن را به قالب مورد نیاز مدل (تنسور PyTorch) تبدیل میکند. همچنین برای جلوگیری از طولانی بودن بیش از حد، محدودیت max_length=128 و truncation=True لحاظ شده‌اند.
2. **غیرفعال کردن گرادیان‌ها**

   چون فقط قصد inference (استنتاج) داریم، نیاز به محاسبه گرادیان نیست.
3. **تولید خروجی اصلاح‌شده**

با استفاده از beam search (بهینه‌سازی خروجی)، مدل بهترین نسخه‌ی اصلاح شده از جمله را تولید میکند.

4. **دی‌کد کردن خروجی به متن قابل خواندن**

هدف استفاده:

ما از این ماژول برای اصلاح املای دستورات انگلیسی کاربران در سیستم خانه هوشمند استفاده کرده‌ایم. گاهی ممکن است کاربر دچار خطای تایپی یا گرامری شود (مثلاً "turn of the light" به جای "turn off the light") و این تابع قبل از پردازش نهایی توسط عامل، متن را اصلاح میکند تا عملکرد سیستم پایدارتر و هوشمندتر باشد.

speak_to_text.py:

این فایل وظیفه‌ی تبدیل صدای کاربر به متن انگلیسی را بر عهده دارد (Speech-to-Text). این ماژول یکی از اجزای کلیدی سیستم ماست، زیرا تعامل کاربران با سیستم خانه هوشمند به صورت صوتی انجام میگیرد. برای پیاده‌سازی این قابلیت از کتابخانه‌ی محبوب و سبک speech_recognition استفاده کرده‌ایم.

ایمپورت‌ها:

- speech_recognition:کتابخانه‌ای برای تشخیص گفتار که از چندین سرویس تشخیص صوت (مثل Google Speech API) پشتیبانی میکند.
- datetime:در این فایل استفاده نشده در فایل home.py استفاده شده است.

تابع record_and_transcribe_long():

این تابع یک قطعه صوتی را از طریق میکروفون دریافت کرده و آن را به متن انگلیسی تبدیل میکند.

```python
def speak_to_text(audio_file):
    recognizer = sr.Recognizer()
    with sr.AudioFile(audio_file) as source:
        audio_data = recognizer.record(source)
    try:
        text = recognizer.recognize_google(audio_data, language="en-US")
        return text
    except sr.UnknownValueError:
        return None
    except sr.RequestError:
        return None
```

اجرای تابع:

1. **ایجاد یک شیء Recognizer**

این شیء وظیفه‌ی کنترل و تنظیم تشخیص صدا را برعهده دارد.

2. **فعال سازی میکروفون و ضبط صدا**
   - sr.audiofile () فایل مورد نظر را به عنوان منبع صدا در نظر میگیرد.
3. **تبدیل صدا به متن با استفاده ازGoogle API**
   - صدای ضبطشده به API رایگان Google Speech Recognition ارسال میشود و متن معادل آن استخراج میگردد.
   - زبان ورودی به صورت انگلیسی آمریکایی (en-US) تنظیم شده است.
4. **مدیریت خطاها**
   - اگر گفتار قابل تشخیص نباشد یا API در دسترس نباشد، None بازگردانده میشود.

کاربرد در پروژه:

در پروژه خانه هوشمند ما، این ماژول به عنوان ورودی صوتی اولیه استفاده می‌شود. کاربر با گفتن دستورات مانند "turn on the lights in the living room" با سیستم ارتباط برقرار میکند. این دستور توسط این ماژول به متن تبدیل شده و سپس به عامل هوشمند ما ارسال میشود.

text_to_speech.py:

این فایل یکی از اجزای کلیدی سیستم تعاملی ماست که وظیفه‌ی تبدیل خروجی متنی عامل هوشمند به صدای قابل شنیدن را برعهده دارد (Text-to-Speech). این ماژول به سیستم ما اجازه می‌دهد تا نه تنها بفهمد بلکه پاسخ هم بدهد آن هم به صورت صوتی.

برای پیاده‌سازی این قابلیت از کتابخانه‌ی سبک و آفلاین pyttsx3 استفاده کردیم تا بدون نیاز به اتصال اینترنت، خروجی صوتی تولید کنیم.

ایمپورت‌ها:

pyttsx3: یک ماژول Python برای تبدیل متن به گفتار که به صورت آفلاین عمل میکند.

تابع text_to_speech(text):

این تابع یک رشته متنی به عنوان ورودی دریافت کرده و آن را با صدای قابل شنیدن برای کاربر پخش میکند.

```
def text_to_speech(text):
    engine = pyttsx3.init()

    engine.setProperty('rate', 150)

    voices = engine.getProperty('voices')
    engine.setProperty('voice', voices[1].id)
    engine.save_to_file(text, r"static\audio\response.wav")

    engine.runAndWait()
```

عملکرد تابع:

1. **راه‌اندازی موتور گفتار**

یک شیء موتور صوتی ساخته می‌شود. بسته به سیستم عامل، موتور پیش‌فرض انتخاب میشود.

2. **تنظیم سرعت گفتار**

سرعت خواندن متن را روی ۱۵۰ کلمه در دقیقه تنظیم کردیم. این مقدار نه خیلی کند است و نه سریع؛ برای شنیدن واضح مناسب است.

3. **تنظیم صدای موتور**

لیست صداهای موجود در سیستم دریافت می‌شود. سپس صدای دوم (index 1) انتخاب می‌شود که معمولاً صدای زنانه است. در صورت نیاز، می‌توان به صدای مردانه (index 0) یا سایر صداهای موجود تغییر داد.

4. **خواندن متن ورودی**

متن ورودی به موتور داده می‌شود تا در فایل صوتی ذخیره شود. سپس با runAndWait() موتور شروع به پخش صوت میکند و تا پایان خواندن منتظر میماند.

کاربرد در پروژه:

در پروژه‌ی ما، بعد از اینکه کاربر یک فرمان صوتی میدهد و پاسخ مناسب از عامل هوشمند تولید مشود، این ماژول آن پاسخ متنی را به کاربر برمیگرداند اما این‌بار به صورت صوتی.

مزایای این ماژول:

- کاملاً آفلاین : بدون نیاز به اینترنت کار میکند.
- سریع و سبک : مناسب برای اجرای بلادرنگ روی سیستم‌های خانگی یا حتی رزبری‌پای.
- قابل تنظیم : میتوان نوع صدا، سرعت، و حتی لحن را تغییر داد.

home.py:

فایل home.py هسته منطقی (backend logic) سیستم خانه هوشمند ما را مدیریت می‌کند. این فایل وظیفه ذخیره‌سازی وضعیت وسایل مختلف خانه (مثل چراغ‌ها، سیستم گرمایشی و تلویزیون)، کنترل آن‌ها و فراهم کردن داده‌های مورد نیاز مثل زمان، تاریخ، وضعیت آب وهوا و اخبار را بر عهده دارد.

ساختار اصلی و عملکرد ماژول‌ها:

ایمپورت‌ها :

در ابتدای فایل، ما تعدادی از کتابخانه‌های استاندارد پایتون و دو API خارجی را وارد (import) کرده‌ایم تا بتوانیم عملکردهای اصلی سیستم خانه هوشمند را پیاده‌سازی کنیم.

- Json: در این پروژه برای ذخیره‌سازی وضعیت دستگاه‌ها (مثل چراغ‌ها، دما، تلویزیون) در فایل home_state.json استفاده شده است. تابع‌های json.load() و json.dump() از این کتابخانه برای بارگذاری و ذخیره فایل JSON استفاده می‌شوند.
- Os: در این پروژه برای اطمینان از اینکه فایل home_state.json وجود دارد یا نه استفاده شده. اگر وجود نداشته باشد، فایل با مقدار پیش‌فرض ساخته می‌شود.

- requests : در این پروژه از این کتابخانه برای دریافت اطلاعات آب‌وهوا (از OpenWeatherMap) و دریافت اخبار روز (ازNewsAPI) استفاده می‌شود.
- Datetime: برای دریافت تاریخ و زمان فعلی سیستم.

1. ذخیره و بازیابی وضعیت وسایل خانه

در ابتدا، برای اینکه اطلاعات روشن یا خاموش بودن وسایل، رنگ چراغ‌ها، دمای اتاق‌ها و کانال تلویزیون پس از هر بار اجرا باقی بماند، از یک فایل JSON به نام home_state.json استفاده میکنیم.

تابع load_state():

اگر فایل home_state.json وجود نداشته باشد، برای اولین بار مقدار پیش فرض را با تابع default_state() تولید و ذخیره میکند، سپس محتوای آن را بارگذاری میکند.

```
def load_state():  7 usages
    if not os.path.exists(STATE_FILE):
        with open(STATE_FILE, "w") as f:
            json.dump(default_state(), f, indent=4)
    with open(STATE_FILE, "r") as f:
        return json.load(f)
```

تابع save_state(state):

وضعیت جدیدی که بعد از هر تغییر ایجاد میشود را در همان فایل ذخیره میکند.

```
def save_state(state):  6 usages
    with open(STATE_FILE, "w") as f:
        json.dump(state, f, indent=4)
```

تابع default_state():

ساختار اولیه وضعیت خانه را تولید میکند که شامل:

- چراغ‌ها در هر اتاق (روشن/خاموش، رنگ، شدت نور)
- سیستم گرمایشی (روشن/خاموش، دما)
- تلویزیون (روشن/خاموش، کانال)

```python
def default_state():  1 usage
    return {
        "lights": {
            "kitchen": {"on": False, "color": "white", "brightness": 100},
            "living_room": {"on": False, "color": "white", "brightness": 100},
            "bathroom": {"on": False, "color": "white", "brightness": 100},
            "WC": {"on": False, "color": "white", "brightness": 100},
            "room1": {"on": False, "color": "white", "brightness": 100},
            "room2": {"on": False, "color": "white", "brightness": 100},
        },
        "temperature": {
            "kitchen": {"on": False, "value": 24},
            "living_room": {"on": False, "value": 24},
            "room1": {"on": False, "value": 24},
            "room2": {"on": False, "value": 24},
        },
        "tv": {
            "living_room": {"on": False, "channel": 1},
        }
    }
```

2. کنترل وسایل خانه

این قسمت شامل توابعی است که از طریق آن‌ها میتوان وسایل مختلف خانه را روشن، خاموش یا پیکربندی کرد:

:turn_on(device, location) / turn_off(device, location)

برای روشن یا خاموش کردن وسایل استفاده میشود.

```python
def turn_on(device, location):  5 usages
    state = load_state()
    if device in state and location in state[device]:
        state[device][location]["on"] = True
        save_state(state)
        return f"{device} in {location} turned ON."
    return "Invalid device or location."


def turn_off(device, location):  1 usage
    state = load_state()
    if device in state and location in state[device]:
        state[device][location]["on"] = False
        save_state(state)
        return f"{device} in {location} turned OFF."
    return "Invalid device or location."
```

:set_color_light(device, location, color)

برای تغییر رنگ چراغ در یک مکان خاص، فقط در صورتی که چراغ روشن باشد استفاده میشود.

```python
def set_color_light(device, location, color):  1 usage
    state = load_state()
    if device in state and location in state[device]:
        if state[device][location]["on"]:
            state[device][location]["color"] = color
            save_state(state)
            return f"{device} in {location} set to color {color}"
        return f"{device} in {location} turned OFF."
    return "Invalid device or location."
```

:set_temperature(location, temperature)

برای تنظیم دمای محیط در اتاق‌های دارای سیستم گرمایشی استفاده میشود. فقط دماهای بین 0 تا 100 درجه سانتی‌گراد مجاز هستند.

```python
def set_temperature(location, temperature):  1 usage
    if 0 <= temperature <= 100:
        state = load_state()
        if location in state["temperature"]:
            state["temperature"][location]["on"] = True
            state["temperature"][location]["value"] = temperature
            save_state(state)
            return f"Temperature in {location} set to {temperature}°C and system turned ON."
        return "Invalid location for temperature control."
    return "Invalid temperature value."
```

:set_brightness_light(device, location, brightness)

برای تنظیم شدت نور چراغ و فعال کردن آن به طور همزمان استفاده میشود.

```python
def set_brightness_light(device, location, brightness):  1 usage
    state = load_state()
    if device in state and location in state[device]:
        state[device][location]["on"] = True
        state[device][location]["brightness"] = brightness
        save_state(state)
        return f"{device} in {location} set to {brightness}% brightness and turned ON."
    return "Invalid device or location."
```

:set_channel_tv(device, location, channel)

برای تغییر کانال تلویزیون به عددی بین ۱ تا ۹ و روشن کردن آن استفاده میشود.

```python
def set_channel_tv(device, location, channel):  1 usage
    if 1 <= channel <= 9:
        state = load_state()
        if device in state and location in state[device]:
            state[device][location]["on"] = True
            state[device][location]["channel"] = channel
            save_state(state)
            return f"{device} in {location} changed to channel {channel} and turned ON."
        return "Invalid device or location."
    return "Invalid channel number."
```

3. دریافت وضعیت کلی خانه

get_status():

این تابع یک خلاصه کامل از وضعیت فعلی خانه را به صورت متن تولید میکند. شامل:

- وضعیت چراغ‌ها: روشن/خاموش، رنگ، شدت نور
- سیستم گرمایشی: روشن/خاموش، دما
- تلویزیون‌ها: روشن/خاموش، کانال

```python
def get_status():  1 usage
    state = load_state()
    status_lines = []

    status_lines.append("Lights:")
    for room, s in state["lights"].items():
        line = f"  - {room.capitalize()}: {'ON' if s['on'] else 'OFF'}, Color: {s['color']}, Brightness: {s['brightness']}%"
        status_lines.append(line)

    status_lines.append("\nTemperature Systems:")
    for room, s in state["temperature"].items():
        line = f"  - {room.replace('_', ' ').capitalize()}: {'ON' if s['on'] else 'OFF'}, Temperature: {s['value']}°C"
        status_lines.append(line)

    status_lines.append("\nTVs:")
    for room, s in state["tv"].items():
        line = f"  - {room.replace('_', ' ').capitalize()}: {'ON' if s['on'] else 'OFF'}, Channel: {s['channel']}"
        status_lines.append(line)

    return "\n".join(status_lines)
```

4. اطلاعات عمومی خانه

get_time():

بازگرداندن ساعت فعلی سیستم (به فرمت 24 ساعته)

```
def get_time():  1 usage
    return datetime.now().strftime("%H:%M")
```

:get_date()

بازگرداندن تاریخ فعلی سیستم(به فرمت YYYY-MM-DD)

```
def get_date():  1 usage
    return datetime.now().strftime("%Y-%m-%d")
```

5. دریافت اطلاعات زنده

:get_weather()

از طریق API سایت OpenWeatherMap ، وضعیت آب و هوای شهر تهران را دریافت و به صورت متنی برمیگرداند.

- API Key استفاده‌شده: 0cd1d47284dc6ddbe045dde4908bae28
- اطلاعات برگشتی شامل شرح آب و هوا و دمای فعلی است.

```
def get_weather():  1 usage
    city_name = "Tehran"
    api_key = "0cd1d47284dc6ddbe045dde4908bae28"
    url = f"https://api.openweathermap.org/data/2.5/weather?q={city_name}&appid={api_key}&units=metric&lang=en"
    response = requests.get(url)
    if response.status_code == 200:
        data = response.json()
        return f"Weather in {city_name}: {data['weather'][0]['description']}, Temperature: {data['main']['temp']}°C"
    return "Error retrieving weather data"
```

:get_news()

از طریق API سایت NewsAPI ، ۵ خبر اول روز از کشور آمریکا را دریافت میکند.

- API Key استفاده‌شده: eef5ed61874640ff8caa9a0bb4adf9eb
- خروجی شامل تیترهای ۵ خبر مهم روز به صورت لیست شماره‌دار است.

```python
def get_news():  1 usage
    api_key = "eef5ed61874640ff8caa9a0bb4adf9eb"
    url = f"https://newsapi.org/v2/top-headlines?country=us&apiKey={api_key}"
    try:
        response = requests.get(url)
        data = response.json()
        if data["status"] == "ok":
            headlines = [article["title"] for article in data["articles"][:5]]
            return "\n".join([f"{i + 1}. {headline}" for i, headline in enumerate(headlines)])
        return "Error: Failed to fetch news."
    except Exception as e:
        return f"An error occurred: {e}"
```

جمع‌بندی:

ماژول home.py به عنوان مغز اصلی سیستم کنترل خانه هوشمند عمل می‌کند. تمام اطلاعات وضعیت به صورت فایل JSON ذخیره شده و همه توابع کنترل و گزارش‌گیری با آن فایل تعامل دارند. همچنین با اتصال به APIهای خارجی مثل آب و هوا و اخبار، تجربه‌ای پویا و واقعی از یک خانه هوشمند فراهم میشود.

Home_state.json:

این فایل JSON ساختار وضعیت لحظه‌ای دستگاه‌های مختلف خانه هوشمند را نمایش میدهد. داده‌ها به صورت ساختاریافته و طبقه‌بندی‌شده ذخیره شده‌اند تا سیستم بتواند به راحتی وضعیت فعلی هر بخش از خانه را دریافت، پردازش و کنترل کند. ساختار این فایل به سه بخش اصلی تقسیم می‌شود: چراغ ها و دما و تی‌وی.

چراغ‌ها(Lights) :

در این بخش، برای هر فضا از خانه، مشخص شده که آیا چراغ روشن است یا خاموش، چه رنگی دارد و میزان روشنایی‌اش چقدر است. به طور مثال:

```json
"lights": {
    "kitchen": {
        "on": true,
        "color": "white",
        "brightness": 70
    },
    "living_room": {
        "on": false,
        "color": "blue",
        "brightness": 100
    },
```

- on: مقدار true یعنی چراغ روشن است، و false یعنی خاموش است.
- Color: رنگ فعلی نور چراغ (مثلاً "white").
- Brightness: شدت روشنایی به درصد (عددی بین 0 تا 100).

سیستم تهویه یا دما(Temperature) :

این بخش وضعیت روشن/خاموش بودن سیستم کنترل دما (مانند کولر یا بخاری) و مقدار دمای تنظیم شده را برای هر اتاق نمایش میدهد. به طور مثال:

- On: اگر true باشد، یعنی سیستم دمایی فعال است.
- Value: مقدار دمای تنظیم شده برای آن اتاق، بر حسب درجه سلسیوس.

تلویزیون(TV) :

در این بخش، وضعیت روشن یا خاموش بودن تلویزیون و کانال فعلی آن در اتاق نشیمن (living_room) داریم . به طور مثال:

- On: اگر true باشد یعنی تلویزیون روشن است.
- Channel: عدد کانالی که تلویزیون روی آن تنظیم شده.

Keywords.json:

این فایل JSON شامل دسته‌بندی‌های مختلفی از عبارات و کلمات کلیدی است که کاربر ممکن است هنگام تعامل با دستیار خانه هوشمند استفاده کند. هدف از این ساختار، درک دقیق تر دستورات طبیعی کاربر و تطبیق آن‌ها با عملکردهای قابل اجراست.

دستورهای کنترلی(Actions) :

این بخش‌ها شامل عباراتی هستند که برای تشخیص نوع دستور (مثل روشن کردن، تنظیم دما، تغییر کانال و...) به کار می‌روند:

:turn_on

عباراتی که به سیستم میفهمانند کاربر می‌خواهد دستگاهی را روشن کند:

```
"turn_on": [
  "turn on",
  "switch on",
  "power on",
  "activate",
  "start",
  "enable",
  "turn the .* on",
  "switch the .* on",
  "power the .* on",
  "enable the .*",
  "turning on",
  "switching on"
],
```

:turn_off

عباراتی برای خاموش کردن دستگاه‌ها:

```
"turn_off": [
  "turn off",
  "switch off",
  "off",
  "deactivate",
  "stop",
  "disable",
  "turn the .* off",
  "switch the .* off",
  "power the .* off",
  "disable the .*",
  "turning off",
  "switching off"
],
```

:set_color

برای تغییر رنگ چراغ‌ها:

```
"set_color": [
  "set color",
  "change color",
  "color to",
  "light color",
  "make it",
  "turn .* to .* color",
  "adjust color",
  "set the color of",
  "change the color of"
],
```

:set_brightness_light

برای تنظیم میزان روشنایی چراغ‌ها:

```
"set_brightness_light": [
  "set brightness",
  "adjust brightness",
  "brightness to",
  "make it brighter",
  "make it dimmer",
  "increase brightness",
  "decrease brightness",
  "lower brightness",
  "raise brightness",
  "brighten",
  "dim",
  "set the brightness",
  "change brightness"
],
```

: set_temperature

برای تغییر دمای اتاق یا کنترل سیستم تهویه:

```
"set_temperature": [
  "set temperature",
  "adjust temperature",
  "temperature to",
  "cool to",
  "heat to",
  "change temperature",
  "increase temperature",
  "decrease temperature",
  "cooling",
  "ac",
  "air conditioner",
  "air conditioning",
  "heater",
  "heating",
  "radiator",
  "boiler",
  "set thermostat",
  "adjust thermostat",
  "temperature setting"
],
```

:set_channel

برای تغییر کانال تلویزیون:

```
"set_channel": [
  "set channel",
  "change channel",
  "channel to",
  "switch to channel",
  "go to channel",
  "tune to channel",
  "turn to channel",
  "select channel",
  "switch channel",
  "change the channel"
],
```

دستگاه‌ها و مکان‌ها(Devices and Locations) :

در این بخش، سیستم از این کلمات استفاده می‌کند تا تشخیص دهد کدام دستگاه یا مکان هدف دستور است.

:Lights

کلمات مترادف با چراغ‌ها:

```
"lights": [
  "light",
  "lights",
  "lamp",
  "bulb",
  "ceiling light",
  "led",
  "lighting",
  "chandelier",
  "desk lamp",
  "floor lamp",
  "spotlight"
],
```

:Tv

کلمات مرتبط با تلویزیون:

```
"tv": [
  "tv",
  "television",
  "smart tv",
  "screen",
  "the tv",
  "flat screen",
  "tele",
  "led tv",
  "oled tv"
],
```

:Kitchen

کلمات مختلف برای آشپزخانه:

```
"kitchen": [
  "kitchen",
  "cooking area",
  "kitchenette",
  "cuisine"
],
```

:room1 و room2

نام‌های مختلف برای اتاق اول و دوم:

```
"room1": [
  "room1",
  "room 1",
  "first room",
  "bedroom 1",
  "main bedroom",
  "master bedroom",
  "primary bedroom",
  "room one"
],
"room2": [
  "room2",
  "room 2",
  "second room",
  "bedroom 2",
  "guest room",
  "second bedroom",
  "room two"
],
```

bathroom و WC:

کلمات برای حمام و سرویس بهداشتی:

```
"WC": [
  "wc",
  "water closet",
  "toilet",
  "bathroom",
  "restroom"
],
```

```
"bathroom": [
  "bathroom",
  "restroom",
  "washroom",
  "toilet",
  "wc",
  "bath",
  "lavatory"
],
```

living_room:

مترادف‌های مختلف برای اتاق نشیمن:

```
"living_room": [
  "living room",
  "hall",
  "main room",
  "lounge",
  "salon",
  "family room",
  "sitting room"
],
```

سوالات اطلاعاتی(Informational Intents) :

این بخش مخصوص سوال‌هایی است که کاربر برای دریافت اطلاعات میپرسد، نه کنترل مستقیم دستگاه‌ها:

:get_time

برای پرسیدن ساعت فعلی:

```
"get_time": [
  "what time",
  "current time",
  "tell me the time",
  "time is it",
  "what is the time",
  "do you know the time",
  "give me the time",
  "show me the time"
],
```

:get_date

برای سوال درباره تاریخ:

```
"get_date": [
  "what date",
  "today's date",
  "current date",
  "tell me the date",
  "date is it",
  "which day is it",
  "give me the date",
  "show me the date"
],
```

:get_weather

برای پرسش در مورد آب و هوا:

```
"get_weather": [
  "weather",
  "what's the weather",
  "how is the weather",
  "weather like",
  "forecast",
  "temperature outside",
  "weather report",
  "tell me the weather",
  "current weather"
],
```

:get_news

برای دریافت اخبار روز:

```
"get_news": [
  "news",
  "latest news",
  "what's in the news",
  "tell me the news",
  "headlines",
  "update me",
  "news update",
  "news headlines",
  "breaking news"
],
```

:get_status

برای اطلاع از وضعیت فعلی دستگاه‌ها:

```
"get_status": [
  "device status",
  "what is running",
  "what is on",
  "status of devices",
  "check devices",
  "check status",
  "is the .* on",
  "are the .* on",
  "show device status",
  "show status",
  "what's currently on"
]
```

config.json

این فایل تنظیمات اصلی و کلی مدل زبانی ما (که بر پایه‌ی مدل BART هست) رو مشخص میکنه. یعنی مشخص میکنه مدل دقیقاً چجوری کار کنه، ساختارش چیه و چه رفتارهایی داشته باشه.

مواردی که این فایل تعیین میکنه:

- **نوع مدل :** این مدل از نوع encoder-decoder هست (مناسب برای تسک‌هایی مثل تصحیح املایی یا خلاصه‌سازی).
- **معماری :** از مدل BartForConditionalGeneration استفاده میشه، یعنی برای تولید متن کاربرد داره.
- **تعداد لایه‌ها :** مدل ما 6 لایه در encoder و 6 لایه در decoder داره (برای فهم و تولید زبان).
- **تعداد هدهای توجه(Attention heads) :** در هر لایه 12 هد هست که به مدل کمک میکنه همزمان به چند قسمت مختلف جمله توجه کنه.
- **اندازه بردارها(d_model) :** هر کلمه به یک بردار 768 بعدی تبدیل میشه.
- **توکن‌ها :** شماره توکن‌هایی مثل bos (شروع جمله)، eos (پایان جمله)، pad(پرکننده) مشخص شدن.
- **درصد dropout :** تنظیم شده برای جلوگیری از overfitting (به مدل کمک میکنه بهتر یاد بگیره و حفظ نکنه).
- **beam search:** برای تولید خروجی بهتر، از beam search با ۴ مسیر استفاده میشه.
- **تنظیمات مربوط به تسک‌ها :** داخل task_specific_params پارامترهای خاصی برای تسک‌هایی مثل خلاصه‌سازی متن هم نوشته.

نتیجه گیری:

در کل، این فایل مشخص می‌کنه مدل چه شکلیه، چطور آموزش دیده و چطور باید خروجی تولید کنه. بدون این فایل، مدل نمیتونه بدرستی بارگذاری و استفاده بشه.

tokenizer.json:

این فایل مربوط به توکنایزر مدل هست و مشخص میکنه که متن ورودی چجوری به توکن (کلمات کوچک شده یا تکه‌های معنی‌دار) تبدیل بشه تا مدل بتونه باهاش کار کنه.

1. **برش یا کوتاه‌سازی(Truncation) :**

- متن‌هایی که خیلی طولانی باشن، از سمت راست بریده میشن (مثل جمله‌های خیلی بلند).
- حداکثر طول مجاز برای ورودی 1024 توکنه.
- از استراتژی LongestFirst استفاده میکنه، یعنی از آخرین بخش‌ها شروع به حذف میکنه.

2. **توکن‌های خاص(Special Tokens) :**

- این بخش نشون میده که چه توکن‌هایی برای شروع(<s>) ، پایان(</s>) ، پر کردن(<pad>) ، ناشناخته (<unk>)و ماسک‌شده (<mask>) استفاده میشن.
- این توکن‌ها برای هدایت درست مدل توی درک جمله‌ها و تولید خروجی خیلی مهمن.

3. پیش‌پردازش متن(Pre-tokenizer) :

- این قسمت تعیین میکنه متن قبل از تبدیل به توکن چجوری تکه‌تکه بشه.
- از نوع ByteLevel استفاده میشه، که حتی با کاراکترهای خاص یا زبان‌های مختلف خوب کار میکنه.

4. پس‌پردازش(Post-processor) :

- بعد از اینکه متن به توکن تبدیل شد، یه سری توکن خاص مثل <s> و </s> به ابتدا و انتهای جمله اضافه میشن تا مدل بهتر متوجه ساختار جمله بشه.
- این کار با روش RobertaProcessing انجام میشه، چون BART شبیه Roberta از این ساختار استفاده میکنه.

5. Decoder:

- مشخص میکنه که بعد از خروجی مدل، چطور باید توکن‌ها دوباره به متن قابل خوندن تبدیل بشن.
- باز هم از نوع ByteLevel هست.

نتیجه گیری:
این فایل تعیین میکنه که متن ورودی چطور به شکل عددی و قابل فهم برای مدل تبدیل بشه، و چطور خروجی عددی مدل دوباره به متن تبدیل شه. وجود این فایل برای استفاده‌ی درست از مدل زبانی ضروریه.

Tokenizer_config.json:

این فایل شامل تنظیمات کلیدی توکنایزر مدل BART هست. وظیفه‌اش اینه که رفتار توکنایزر رو در تبدیل متن به توکن (و برعکس) مشخص کنه. به زبون ساده، این فایل به مدل میگه از چه توکن‌هایی استفاده کن؟ چطور متن رو آماده کن؟ و چقدر طول متن مجازه؟

vocab.json:

این فایل فهرست کامل توکن‌ها و شماره‌ی اختصاصی هرکدومه. در این فایل هر کلمه یا قطعه‌کلمه (subword) که مدل بلد هست، با یک عدد مشخص شده. مثلاً "<s>" توکن شروع جمله‌ست و عدد 0 داره .توکن‌هایی مثل "Ġ"نشان‌دهنده‌ی فاصله هستند (برای تشخیص اول کلمات)، و توکن‌هایی مثل "âĢ" یا "Ļ" هم معمولاً به

دلیل encoding خاص داده‌ها توی pretraining به وجود میان. این فهرست درواقع دیکشنری مدل زبانی هستش و برای تبدیل متن به توکن عددی (و برعکس) استفاده میشه.

:ui.html

این فایل شامل محتویات ui میباشد که شامل 3 بخش اصلی است:

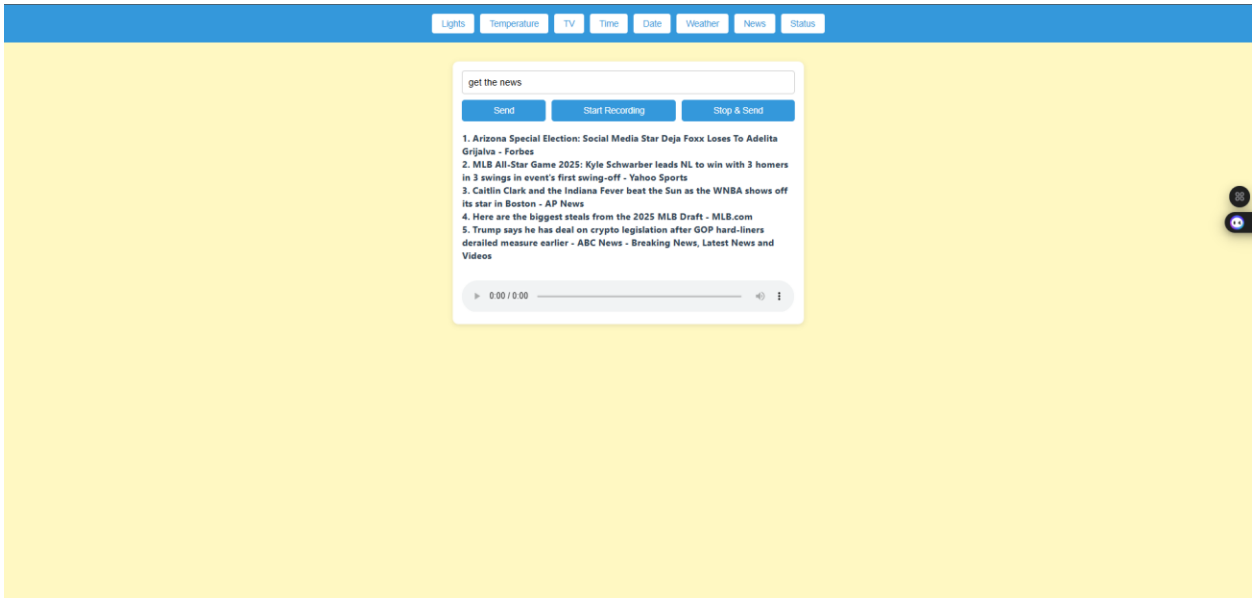
- بخش commad-line: مستقیم command میتوان نوشت.
- بخش user-friendly: شامل چند دکمه برای کنترل چند چیز متفاوت است. هر دکمه modal مربوط به آن بخش را باز میکند.
- بخش صدا: شامل دو دکمه است که یکی از آن ها ضبط صدا را آغاز میکند و دیگری ضبط صدا را متوقف میکند و فایل ساخته شده توسط کتابخانه recorder.js را به فایل ui_connection ارسال میکند.

:ui_connection

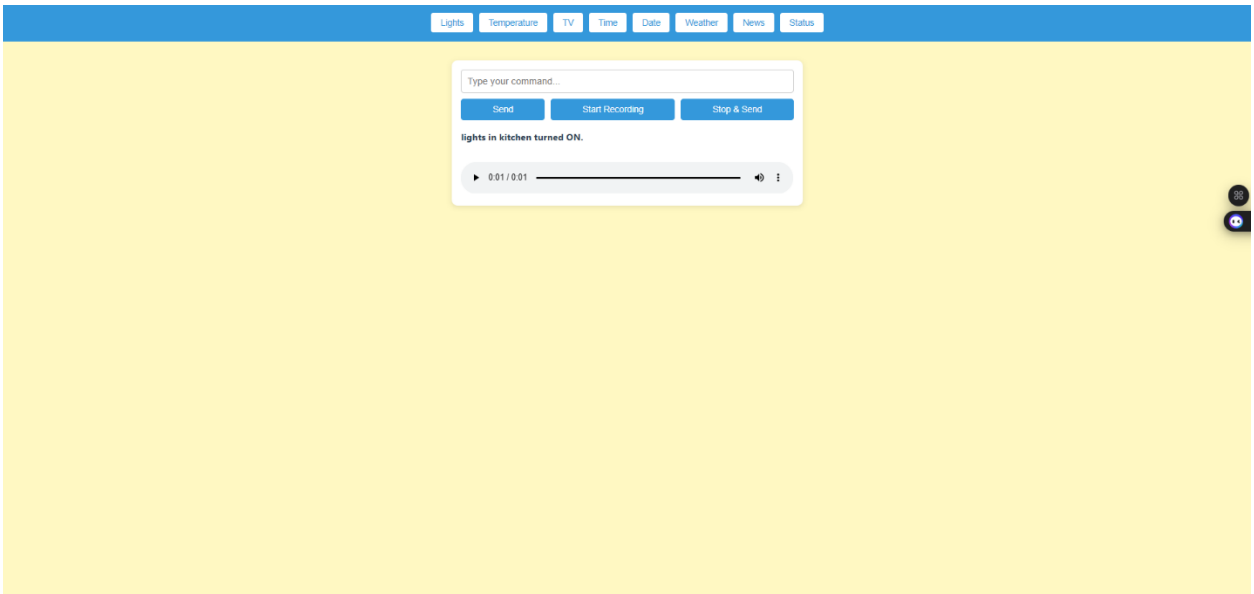
این فایل یک سرور با استفاده از کتابخانه flask میسازد که بین backend برنامه و فایل html که مربوط به ui هست ارتباط برقرار میکند. شامل 3 بخش اصلی است:

- فانکشن process: این فانکشن دستورات مستقیم را پردازش میکند و به فانکشن smart_home_agent در فایل agent.py میفرستد و خروجی را از آن میگیرد و به کد html، POST میکند.
- فانکشن process_audio: کد جاوااسکریپت موجود در فایل ui.html صدای ضبط شده را به این فانکشن POST میکند و این فانکشن فایل دریافت شده را به تابع speak_to_text در فایل speech_to_text.py میدهد و در ازای آن متن استخراج شده از فایل صوتی را میگیرد و مانند فانکشن process آن را پردازش میکند و به تابع smart_home_agent در فایل agent.py میفرستد و پس از به پایان رسیدن عملیات پاسخ دریافت شده را با استفاده از تابع text_to_speech در فایل text_to_speak.py آن متن را تبدیل به صدا میکند و در فایل static/audio/response.wav ذخیره میکند.
- فانکشن get_audio: فایل صوتی ساخته شده را به ui میفرستد.

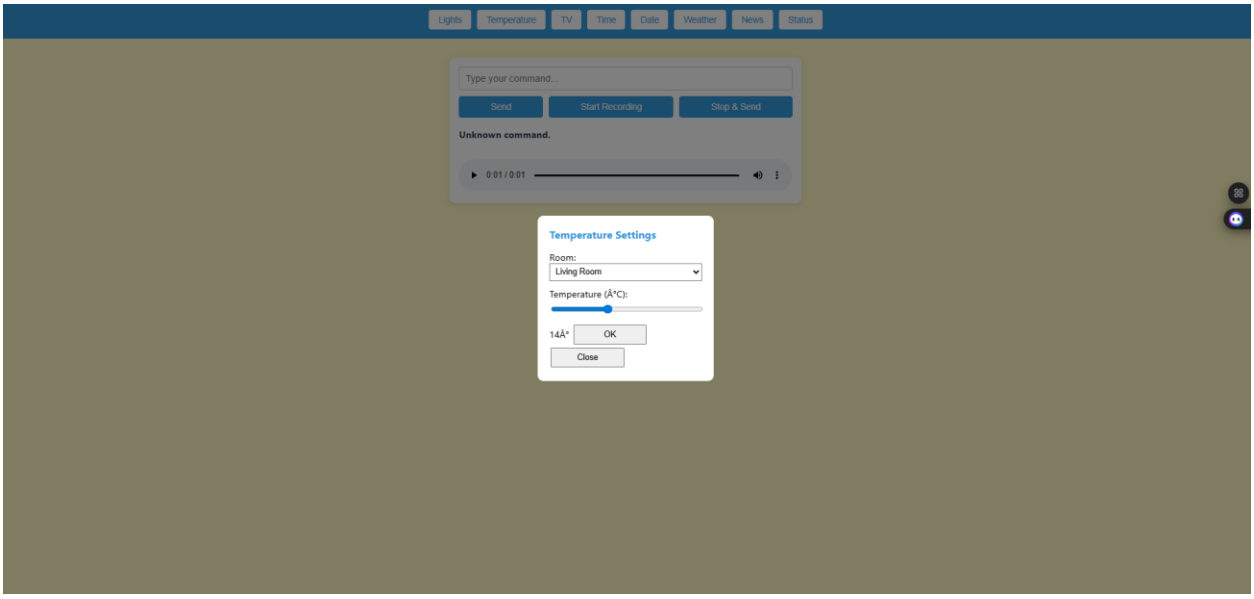
مثال اول:



مثال دوم:



مثال سوم:



نمودار درختی فایل ها

Templates /

فایل مربوط به ui :Ui.html

Static/

Js/

فایل مربوط به صدا :Recorder.js

Audio/

فایلی که هر دفعه که متن تبدیل به صدا میشود ، بروزرسانی میشود :Response.wav

Models/

Spelling_correction_english_base/

تمام فایل های این فولدر مربوط به مدل از پیش آموزش دیده است :*

Ui_connection.py: فایل مربوط به وصل کردن ui به backend

فایل تست کردن بک اند برنامه یا همان تست فایل AGENT_TEST.py: agent.py

Agent.py:همان فایل مغز پروژه ما است که ورودی های کامندی را تجزیه و پردازش میکند و تغییرات را در فایل home_state.json با استفاده از تابع های موجود در home.py اعمال میکند

از مدل از پیش آموزش دیده برای اصلاح گرامری ورودی استفاده میکند :correct_spelling.py

تابع های مورد نیاز برای انجام دادن تغییرات در فایل home_state.json را درون خود دارد :home.py

وضعیت خانه درون این فایل ذخیره شده است :home_state.json

تمام کلماتی را که کاربر برای هر کاری ممکن است بگوید درون این فایل ذیره شده اند :keywords.json

صدا را به متن و نوشته تبدیل میکند :speak_to_text.py

متن را به صدا تبدیل میکند و در فایل response.wav ذخیره میکند :text_to_speak