code code code

- vuejs
- button css
- hass
    - Nouvelle page
- R

vuejs

Models

```
<!-- BaseInput.vue component -->

<template>
    <div>
        <input type="text" :value="title" @input="updateValue" />
    </div>
</template>

<script>
export default {
    name: 'BaseInput',
    props: ['title'],
    model: {
        prop: 'title',
        event: 'changeTitle'
    },
    methods: {
        updateValue(event) {
            this.$emit('changeTitle', event.target.value)
        }
    },
}
</script>
```

button css

```
padding: 7px 10px;
border-radius: 3px;
font-size: 15px;
font-weight: 500;
border: var(--border-button-flat);
background-color: var(--background-secondary);
align-items: center;
background-image: linear-gradient(#FCFCFC, #F6F7F9);
border-radius: 10px;
```

hass

hass

Nouvelle page

```
"""Fallback Conversation Agent"""
from __future__ import annotations

import logging
import requests

from homeassistant.components import conversation
from homeassistant.config_entries import ConfigEntry
from homeassistant.core import HomeAssistant
from homeassistant.util import ulid
from home_assistant_intents import get_languages


from homeassistant.helpers import (
    config_validation as cv,
    intent,
)

from .const import (
    CONF_DEBUG_LEVEL,
    CONF_PRIMARY_AGENT,
    CONF_FALLBACK_AGENT,
    DEBUG_LEVEL_NO_DEBUG,
    DEBUG_LEVEL_LOW_DEBUG,
    DEBUG_LEVEL_VERBOSE_DEBUG,
    DOMAIN,
)


_LOGGER = logging.getLogger(__name__)

CONFIG_SCHEMA = cv.config_entry_only_config_schema(DOMAIN)
```

```
# hass.data key for agent.
DATA_AGENT = "agent"



async def async_setup_entry(hass: HomeAssistant, entry: ConfigEntry) -> bool:
    """Set up Fallback Conversation Agent from a config entry."""
    agent = FallbackConversationAgent(hass, entry)

    conversation.async_set_agent(hass, entry, agent)

    return True

def post_to_api(url, json):
        return requests.post(url, json=json)

class FallbackConversationAgent(conversation.AbstractConversationAgent):
    """Fallback Conversation Agent."""
    def __init__(self, hass: HomeAssistant, entry: ConfigEntry) -> None:
        """Initialize the agent."""
        self.hass = hass
        self.entry = entry

    @property
    def supported_languages(self) -> list[str]:
        """Return a list of supported languages."""
        return get_languages()

    async def async_process(
        self, user_input: conversation.ConversationInput
    ) -> conversation.ConversationResult:
        url = 'http://192.168.122.1:10500'
        data = {'text': user_input.text}

        response_from_api = await self.hass.async_add_executor_job(post_to_api, url, data)
#task.executor(requests.post, )

        response = intent.IntentResponse(language=user_input.language)
        response.async_set_speech(response_from_api.text)
        return conversation.ConversationResult(
            conversation_id=None,
```

```
            response=response
        )
```

```
"""Config flow for Fallback Conversation integration."""
from __future__ import annotations

import logging
# from types import MappingProxyType
from typing import Any

import voluptuous as vol

from homeassistant import config_entries
from homeassistant.const import CONF_NAME
from homeassistant.core import HomeAssistant, async_get_hass, callback
from homeassistant.data_entry_flow import FlowResult
from homeassistant.helpers.selector import (
    ConversationAgentSelector,
    ConversationAgentSelectorConfig,
    SelectSelector,
    SelectSelectorConfig,
    SelectOptionDict,
    SelectSelectorMode,
)

from .const import (
    CONF_DEBUG_LEVEL,
    CONF_PRIMARY_AGENT,
    CONF_FALLBACK_AGENT,
    DEBUG_LEVEL_NO_DEBUG,
    DEBUG_LEVEL_LOW_DEBUG,
    DEBUG_LEVEL_VERBOSE_DEBUG,
    DOMAIN,
    DEFAULT_NAME,
    DEFAULT_DEBUG_LEVEL,
)

_LOGGER = logging.getLogger(__name__)
```

```
STEP_USER_DATA_SCHEMA = vol.Schema(
    {
        vol.Optional(CONF_NAME, default=DEFAULT_NAME): str,
        vol.Optional(CONF_DEBUG_LEVEL, default=DEFAULT_DEBUG_LEVEL): SelectSelector(
            SelectSelectorConfig(
                options=[
                    SelectOptionDict(value=DEBUG_LEVEL_NO_DEBUG, label="No Debug"),
                    SelectOptionDict(value=DEBUG_LEVEL_LOW_DEBUG, label="Some Debug"),
                    SelectOptionDict(value=DEBUG_LEVEL_VERBOSE_DEBUG, label="Verbose Debug"),
                ],
                mode=SelectSelectorMode.DROPDOWN
            ),
        ),
    }
)

class ConfigFlow(config_entries.ConfigFlow, domain=DOMAIN):
    """Fallback Agent config flow."""

    VERSION = 1

    async def async_step_user(self, user_input: dict[str, Any] | None = None) -> FlowResult:
        """Handle the initial step."""
        _LOGGER.debug("ConfigFlow::user_input %s", user_input)
        if user_input is None:
            return self.async_show_form(
                step_id="user",
                data_schema=STEP_USER_DATA_SCHEMA,
            )

        return self.async_create_entry(
            title=user_input.get(CONF_NAME, DEFAULT_NAME),
            data=user_input,
        )

    @staticmethod
    @callback
    def async_get_options_flow(config_entry: config_entries.ConfigEntry) ->
config_entries.OptionsFlow:
        """Create the options flow."""
```

```
        return OptionsFlow(config_entry)

class OptionsFlow(config_entries.OptionsFlow):
    """Fallback config flow options handler."""

    def __init__(self, config_entry: config_entries.ConfigEntry) -> None:
        """Initialize options flow."""
        self.config_entry = config_entry
        self._options = dict(config_entry.data)
        self._options.update(dict(config_entry.options))

    async def async_step_init(
        self, user_input: dict[str, Any] | None = None
    ) -> FlowResult:
        """Manage the options."""
        if user_input is not None:
            self._options.update(user_input)
            return self.async_create_entry(
                title=user_input.get(CONF_NAME, DEFAULT_NAME),
                data=self._options,
            )

        schema = await self.fallback_config_option_schema(self._options)

        return self.async_show_form(
            step_id="init",
            data_schema=vol.Schema(schema),
        )

    async def fallback_config_option_schema(self, options: dict) -> dict:
        """Return a schema for Fallback options."""
        return {
            vol.Required(
                CONF_DEBUG_LEVEL,
                description={"suggested_value": options.get(CONF_DEBUG_LEVEL,
DEFAULT_DEBUG_LEVEL)},
                default=DEFAULT_DEBUG_LEVEL,
            ): SelectSelector(
                SelectSelectorConfig(
                    options=[
```

```
                        SelectOptionDict(value=DEBUG_LEVEL_NO_DEBUG, label="No Debug"),
                        SelectOptionDict(value=DEBUG_LEVEL_LOW_DEBUG, label="Some Debug"),
                        SelectOptionDict(value=DEBUG_LEVEL_VERBOSE_DEBUG, label="Verbose
Debug"),
                    ],
                    mode=SelectSelectorMode.DROPDOWN
                ),
            ),
        };
```

R

```
data1 <- read.csv("1950.csv", header=TRUE, sep=";")
head(data1,3)
blagnac <- data1[c(2, 6, 17)]  |> subset(NOM_USUEL == "TOULOUSE-BLAGNAC")
noel <- blagnac[grep("1225", blagnac$AAAAMMJJ), ]
mean(noel$TM)
median(noel$TM)
noelaubalcon <- noel[noel[, "TM"]>7,]
paque <- blagnac[substr(blagnac$AAAAMMJJ,5,9) == "0410", ]
```

Moyenne Noel: 5.976712 Mediane: 6.8

mediane paque: 11.7