

# System Design Documentation

Topics are covered:

1. Initialize Vonage Client
2. Outbound Call
3. Inbound call
4. Connect to WebSocket
5. Play audio into WebSocket
6. Speech to Text
7. Text to Speech
8. Flow Chart

## Initialize Vonage Client

```
client = vonage.Client(key='YOUR_API_KEY', secret='YOUR_API_SECRET')
voice = vonage.Voice(client)
```

Replace '**YOUR\_API\_KEY**' and '**YOUR\_API\_SECRET**' with your actual Vonage API key and secret.

## Inbound Call

Receive an inbound call.

```
router('/webhooks/answer', methods=['get', 'post'])
Func answer_url(request):
    call_from = request.args['from']
    isUser = check_user_exists(from_no)
    if isUser:
        return ncco = [
            {
                "action": "talk",
                "text": f"Hi John, we will be with you shortly."
            }
        ]
    else:
        return ncco = [
            {
                "action": "talk",
                "text": "Hello, sorry, we do not recognize your number."
            }
        ]
    ]
```

- 1) The **/webhooks/answer** endpoint is called by Vonage when a call is received.
- 2) It accepts both **GET** and **POST** requests, which is useful because Vonage might send the request using either method depending on how it is configured.
- 3) This function, **answer\_call**, is executed whenever a request is received at **/webhooks/answer**.
- 4) This line extracts the **from** parameter from the incoming request's query string.
- 5) **request.args** is used to get query parameters in Flask. **request.args['from']** retrieves the value of the from parameter, which is the caller's phone number.
- 6) It calls **check\_user\_exists(call\_from)** to determine if the caller is in the database.
- 7) **NCCO** stands for Nexmo Call Control Object, which is a list of actions that define the call flow.
- 8) In this example, the NCCO contains one action:
  - a) **'action': 'talk'** specifies that the action is to play a text-to-speech message.
  - b) **'text':** Message that will be spoken to the recipient.

```
clients = {}
```

The clients dictionary is used to store the unique client IDs (UUID).

Checks if the target client ID exists in the clients dictionary. This ensures that the target client is currently connected.

## Outbound Call

how to make an outbound call using the Vonage Voice

```
response = voice.create_call({
    'to': [{ 'type': 'phone', 'number': 'RECIPIENT_NUMBER' }],
    'from': { 'type': 'phone', 'number': 'YOUR_VONAGE_NUMBER' },
    'ncco': [
        {
            'action': 'talk',
            'text': 'This is a test call from Vonage API using Python. Have a
great day!'
        }
    ]
})
```

- 1) **voice.create\_call** is a method provided by the Vonage Python SDK to create an outbound call.
- 2) The **to** field specifies the recipient of the call. It is a list of dictionaries, where each dictionary represents a phone number to call.
  - a) **'type': 'phone'** indicates the type of endpoint is a phone.
  - b) **'number': 'RECIPIENT\_NUMBER'** is the phone number of the recipient. Replace **'RECIPIENT\_NUMBER'** with the actual phone number you want to call.
- 3) The **from** field specifies the caller ID, i.e., the number that will appear as the caller to the recipient.
  - a) **'type': 'phone'** indicates the type of endpoint is a phone.
  - b) **'number': 'YOUR\_VONAGE\_NUMBER'** is the Vonage number that will be shown as the caller ID. Replace **'YOUR\_VONAGE\_NUMBER'** with your actual Vonage virtual number.
- 4) **NCCO** stands for Nexmo Call Control Object, which is a list of actions that define the call flow.
- 5) In this example, the NCCO contains one action:
  - a) **'action': 'talk'** specifies that the action is to play a text-to-speech message.
  - b) **'text':** Message that will be spoken to the recipient.

## Connect to WebSocket

You can use the Vonage Voice API to connect a call to a WebSocket, giving you a two-way stream of the call audio delivered over the WebSocket protocol in real-time

```
route("/webhooks/answer")
Func answer_call():
    ncco = [
        {
            "action": "talk",
            "text": "We will now connect you to the echo server, wait a moment then start speaking.",
        },
        {
            "action": "connect",
            "from": "Vonage",
            "endpoint": [
                {
                    "type": "websocket",
                    "uri": f"wss://{request.host}/socket",
                    "content-type": "audio/l16;rate=16000",
                }
            ],
        },
    ],
]

return jsonify(ncco)
```

1. This line defines a Flask route that listens for incoming HTTP **GET requests** at the URL **/webhooks/answer**.
2. This function, **answer\_call**, is executed whenever a GET request is received at /webhooks/answer.
3. This NCCO is a list of actions that define the call flow.
4. The first action is a **"talk"** action, which instructs the system to play a text-to-speech message to the caller, informing them that they will be connected to the echo server.
5. The second action is a **"connect"** action, which connects the call to an external WebSocket endpoint.
  - a) **"type": "websocket"** specifies that the endpoint type is WebSocket.
  - b) **"uri"** is the URI of the WebSocket server. It's set to the current host (**request.host**) with /socket appended to it. This assumes that the WebSocket server is running on the same host as the Flask application and listening on the /socket path.
  - c) **"content-type"** specifies the content type of the audio data expected by the WebSocket server. In this case, it's set to **audio/l16;rate=16000**, which indicates linear PCM audio data with a sample rate of 16 kHz.
6. This line converts the NCCO list to a JSON response using Flask's jsonify function and returns it.

## Play audio into a WebSocket

```
socketio.on('message')
Func handle_message(data):
    target_client_id = data.get('targetClientId')
    message_content = data.get('message')
    if target_client_id in clients:
        target_sid = clients[target_client_id]
        emit('message', {'from': data.get('from'), 'content': message_content},
            room=target_sid)
```

1. **@socketio.on('message')**: This decorator registers the function as an event handler for the message event. This event is triggered when a client sends a message.
2. **def handle\_message(data)::** Defines the function to handle the message event. The data parameter contains the message data sent by the client.
3. **target\_client\_id = data.get('targetClientId')**: Extracts the target client ID from the message data. This ID specifies the intended recipient of the message.
4. **message\_content = data.get('message')**: Extracts the actual message content from the message data.
5. **if target\_client\_id in clients::** Checks if the target client ID exists in the clients dictionary. This ensures that the target client is currently connected.
6. **target\_sid = clients[target\_client\_id]**: Retrieves the UUID of the target client from the clients dictionary.
7. Sends the message to the target client. The emit function sends the message with the event name 'message' and includes the sender's ID (**data.get('from')**) and the message content (**message\_content**). The **room=target\_id** parameter ensures that the message is sent to the specific UUID of the target client

## Speech to Text

```
Func speech_to_text(mp3: file):
    // some thing happening
    return text
```

1. **mp3**: This parameter represents the audio file (in MP3 format) that contains the speech data to be converted to text.
2. The function returns the transcribed text extracted from the audio input. This text represents the spoken words captured in the audio file.

## Text to Speech

```
Func text_to_speech(text):  
    // some thing happening  
    return audio
```

1. **text:** This parameter represents the text input that needs to be converted into speech.
2. The function returns the generated audio data in the form of an MP3 file. This audio file contains the speech corresponding to the input text.

## Event

Handle incoming webhook events from Vonage (formerly Nexmo) related to call events

Call status updates or other events relevant to the call lifecycle

```
route("/webhooks/event", methods=['POST'])  
Func event_callback():  
    data = request.get_json()  
    return ('', 204)
```

1. listens for incoming HTTP **POST** requests at the URL **/webhooks/event**.
2. This function, **event\_callback**, is executed whenever a POST request is received at /webhooks/event.
3. **request.get\_json()** is a Flask method to parse JSON data from the request body.
4. This line returns an empty response with HTTP status code **204 (No Content)**.
5. Vonage expects a 204 response to indicate that the webhook event was successfully **received and processed**.

## Flowchart

