System Design Documentation

Topics are covered:

1. Initialize Vonage Client
2. Outbound Call
3. Inbound call
4. Connect to WebSocket
5. WebSocket Connection
6. Play audio into WebSocket
7. WebSocket Disconnection
8. Speech to Text
9. Text to Speech
10. 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** sands 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.

WebSocket Connection

```
socketio.on('connect')
Func handle_connect():
    clients = request.args.get('uuid')
    if uuid:
        clients [uuid] = request.sid
        print(f'Connection established for call UUID: {uuid}')
```

1. ***@socketio.on('connect'):*** The connect event is triggered when a client successfully establishes a connection with the server.
2. ***def handle_connect():*** The handle_connect function that will be executed when a client connects.
3. ***uuid = request.args.get('uuid'):*** Retrieves the uuid from the query parameters of the connection request. request.args.get('uuid') extracts the uuid parameter from the URL of the WebSocket connection request.
4. ***if uuid:*** Checks if the uuid was provided in the connection request. This ensures that only clients with a valid uuid are processed.
5. ***clients[uuid] = request.sid:***
   a) If the **uuid** is present, it maps this **uuid** to the client's Socket.IO session ID **(request.sid).** request.sid is a unique identifier for the client's session, assigned by the Socket.IO server when the client connects.
   b) This line updates the clients dictionary, which keeps track of connected clients and their corresponding session IDs. This allows the server to later route messages to the correct client using their **uuid**.

Play audio into 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]
        audio_data = audio_file.read();
        emit('message', {'from': data.get('from'), audio_data, to=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
8. Audio data received from the WebSocket is saved to a file, and optionally, audio data can be sent back to the caller.
9. **File Handling:** Ensure that the audio_file is properly opened, read, and closed

WebSocket Disconnect

```
socketio.on('disconnect')
Func handle_disconnect():
    uuid = id
    if uuid:
        # 'Connection closed for call UUID'
        del clients
```

1. ***@socketio.on('disconnect'):*** The disconnect event is triggered when a client disconnects from the server.
2. ***def handle_disconnect():*** Defines the handle_disconnect function that will be executed when a client disconnects.
3. The clients dictionary (assumed to be declared globally or within an appropriate scope), which maps UUIDs to Socket.IO session IDs.
4. ***if uuid:*** Checks if the uuid list is not empty. If it contains at least one element, it means the session ID was found in the **clients** dictionary.
5. ***del clients[uuid[0]]:*** Removes the entry from the clients dictionary for the given **uuid**. This cleanup ensures that the dictionary only contains active connections.

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

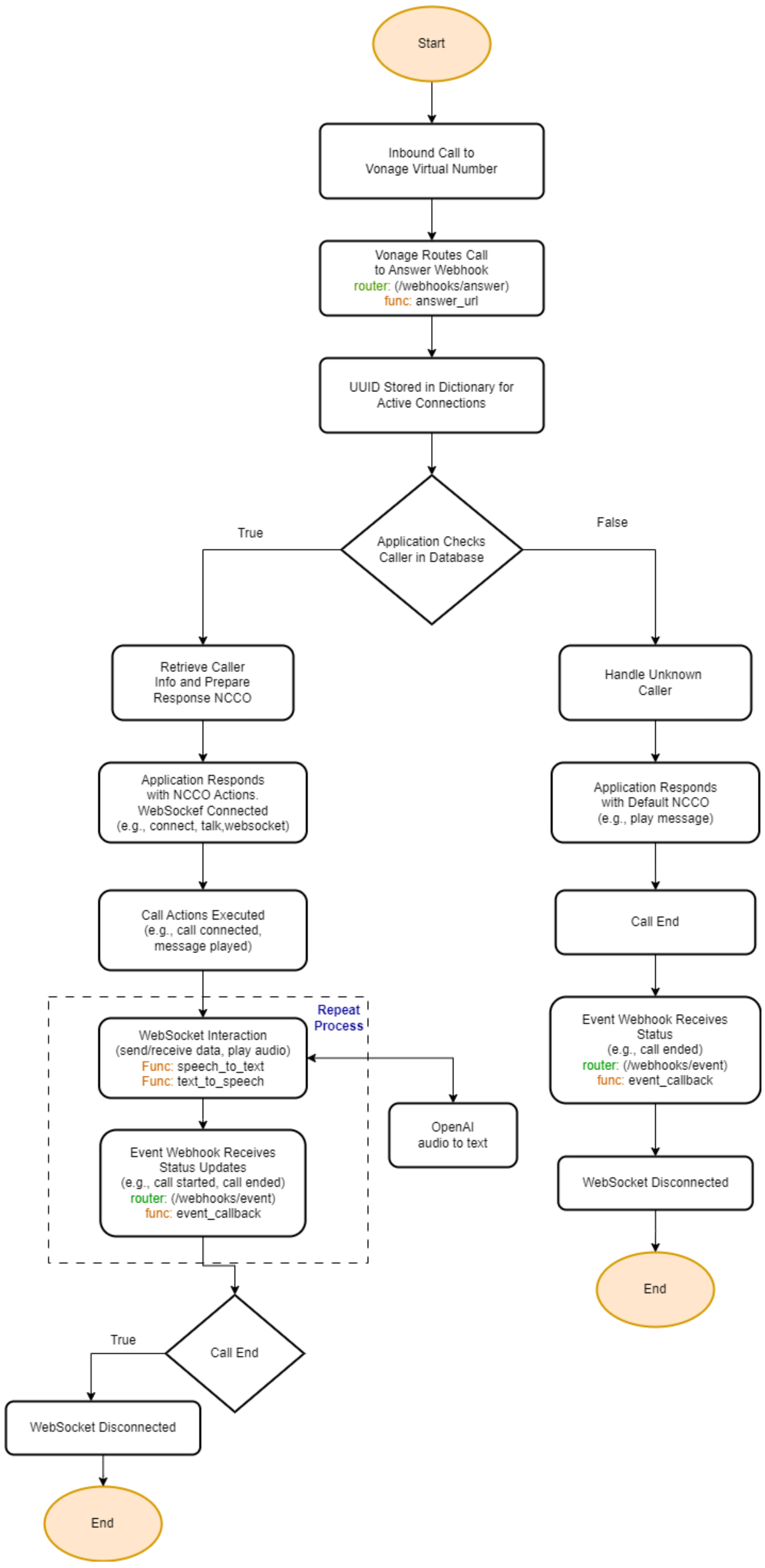
Handle incoming webhook events from Vonage (formerly Nexmo) related to call events

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


Start
Inbound Call to Vonage Virtual Number
Vonage Routes Call to Answer Webhook
router: (/webhooks/answer)
func: answer_url
UUID Stored in Dictionary for Active Connections
Application Checks Caller in Database
True
False
Retrieve Caller Info and Prepare Response NCCO
Application Responds with NCCO Actions. WebSockef Connected (e.g., connect, talk,websocket)
Call Actions Executed (e.g., call connected, message played)
Repeat Process
WebSocket Interaction (send/receive data, play audio)
Func: speech_to_text
Func: text_to_speech
OpenAI audio to text
Event Webhook Receives Status Updates (e.g., call started, call ended)
router: (/webhooks/event)
func: event_callback
Call End
True
WebSocket Disconnected
End
Handle Unknown Caller
Application Responds with Default NCCO (e.g., play message)
Call End
Event Webhook Receives Status (e.g., call ended)
router: (/webhooks/event)
func: event_callback
WebSocket Disconnected
End