# Amazon Connect Documentation

Giovanna Lorena Delgado Mendoza - A01656039

Karla Stefania Cruz Muñiz - A01661547

José Antonio Moreno Tahuilan - A01747922

Héctor González Sánchez - A01753863

Alfredo Azamar López - A01798100

Abner Maximiliano Lecona Nieves- A01753179

Benjamín Alejandro Cruz Cervantes - A01747811

Eric Manuel Navarro Martínez - A01746219

Joahan Javier García Fernández - A01748222

Bernardo Alejandro Limón Montes de Oca A01736575

**Desarrollo e implantación de sistemas de software (Gpo 501)**

**Profesores**

Alvaro Hernández Quijano

José Fernando Ignacio Tavera Parra

Alberto Michel Pérez Domínguez

Enrique González Núñez

Julio Guillermo Arriaga Blumenkron

Víctor Adrián Sosa Hernández

**Fecha de entrega**:

17 de junio del 2024

## Link to the main repository

https://github.com/Izzi-Connect-Tec

## Amazon Connect instance for Izzi Connect

### Description

The following Amazon Connect instance was created for the Izzi Connect application developed for the 2024 course of “Desarrollo e implantación de sistemas de software” in collaboration with Amazon US. It includes integrations with Lambda functions, Amazon Lex, and Contact Lens to achieve an outstanding product and guarantee excellence regarding customer service.

### Key information

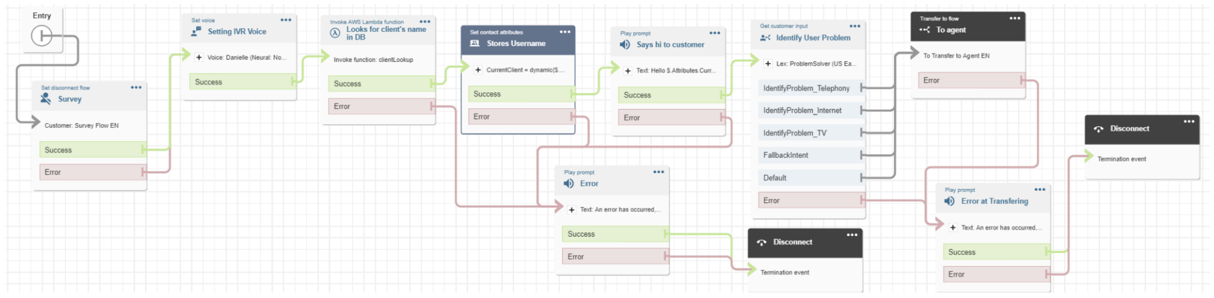
| | |
|---|---|
| Instance Alias | izzi-team |
| Access URL | https://izzi-team.my.connect.aws/ |
| Region | us-east-1 |
| Phone Number | +52 81 2682 1781 (DID) |
| S3 Bucket | Call recordings: amazon-connect-5bf6cf17bae2/connect/izzi-team/CallRecordings<br><br>Chat transcripts: amazon-connect-5bf6cf17bae2/connect/izzi-team/ChatTranscripts |
| Lambda Functions | UploadIssue: (arn:aws:lambda:us-east-1:905418447691:function:UploadIssue)<br><br>callDetails: (arn:aws:lambda:us-east-1:905418447691:function:callDetails)<br><br>clientLookup: (arn:aws:lambda:us-east-1:905418447691:function:clientLookup)<br><br>postSurvey: (arn:aws:lambda:us-east-1:905418447691:function:postSurvey) |

| | |
|---|---|
| Amazon Lex Bot | ProblemSolver : connectAlias |
| Approved origins | https://44.209.22.101:8443 |

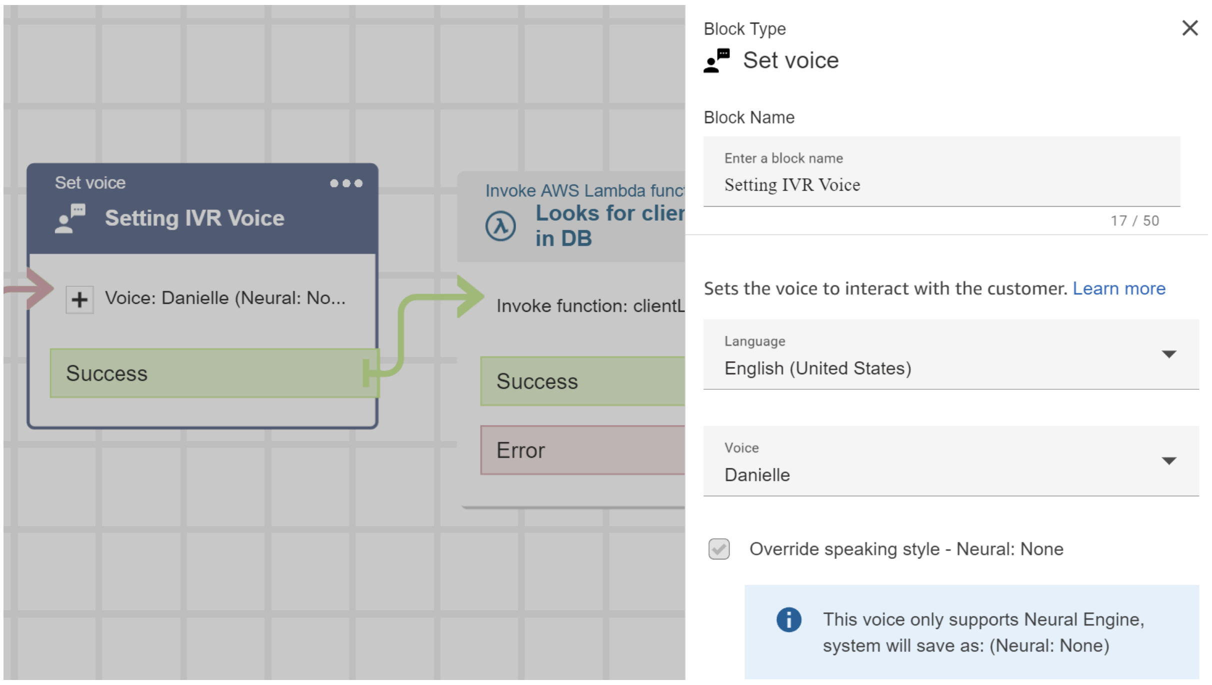
## Contact flows

Our main idea was to provide both Izzi an innovative solution that could help accelerate their standard process and to help them to keep their clients close by offering them an outstanding service. For that to be accomplished we decided to implement an Amazon Lex Bot alongside multiple Lambda Functions to create a smart IVR that could gather key information before every call and then transfer clients to a real agent.

### [1] First interaction contact flow



The purpose of this flow is to welcome the customer by name and obtain information about the issue the customer is experiencing through voice input. This information is then used to transfer the customer to an agent who will already have the necessary information to assist them. To make this possible, we start by configuring the voice that the IVR will use through the "Set voice" block, which in this case is in English.



Block Type

Set voice

Block Name

Enter a block name: Setting IVR Voice

17 / 50

Sets the voice to interact with the customer. Learn more

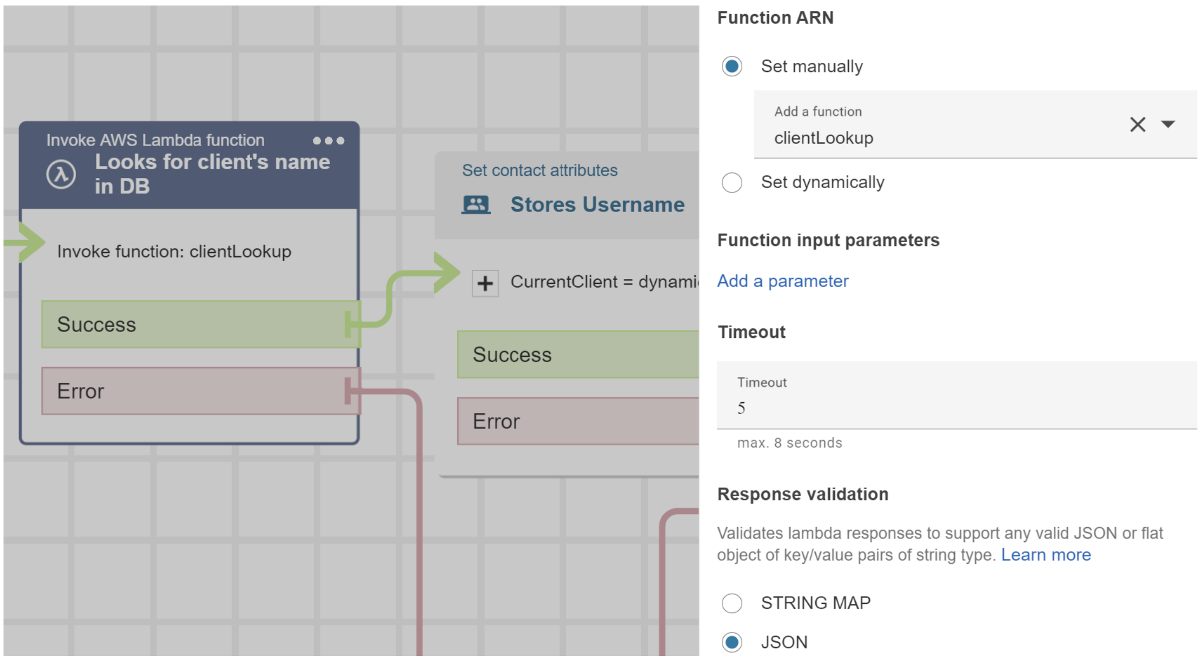
Language: English (United States)

Voice: Danielle

[✓] Override speaking style - Neural: None

This voice only supports Neural Engine, system will save as: (Neural: None)

Immediately afterwards, the "Invoke AWS Lambda function" block is used to call the "clientLookup" Lambda function.



Invoke AWS Lambda function
Looks for client's name in DB
Invoke function: clientLookup
Success
Error
Set contact attributes
Stores Username
CurrentClient = dynami
Success
Error

**Function ARN**

[✓ Set manually]

Add a function: clientLookup

Set dynamically

**Function input parameters**

Add a parameter

**Timeout**

Timeout: 5

max. 8 seconds

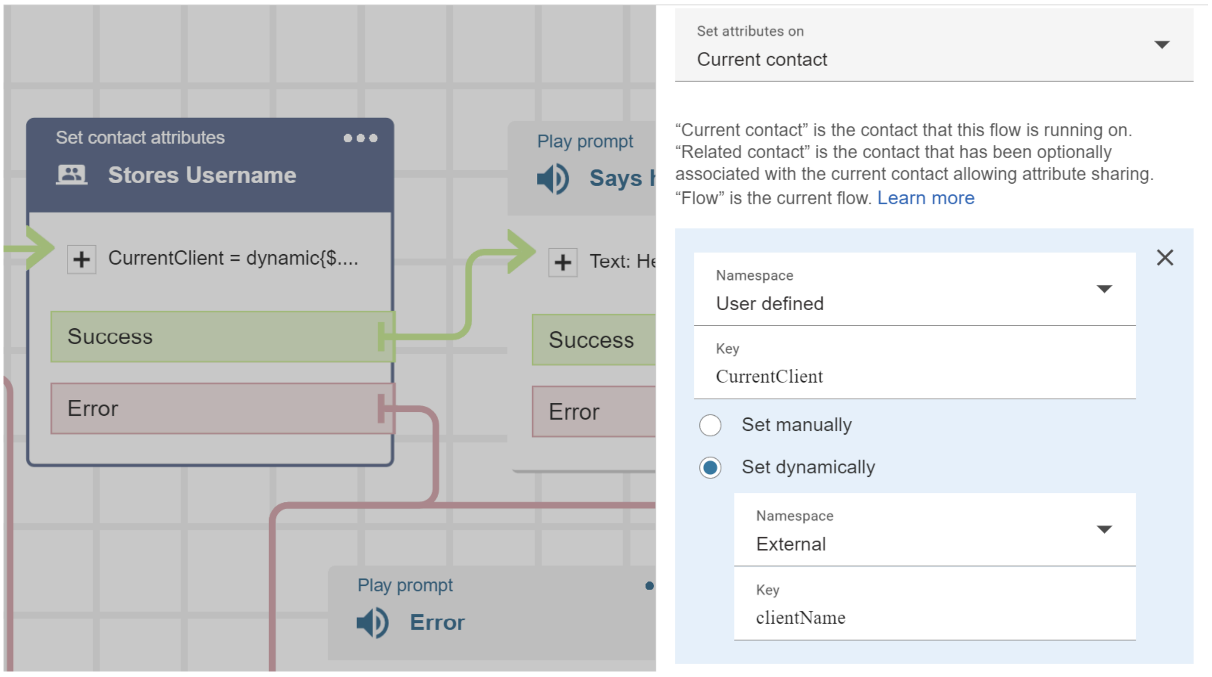
**Response validation**

Validates lambda responses to support any valid JSON or flat object of key/value pairs of string type. Learn more

STRING MAP [✓ JSON]

To see more details about the lambda function [click here.](#)

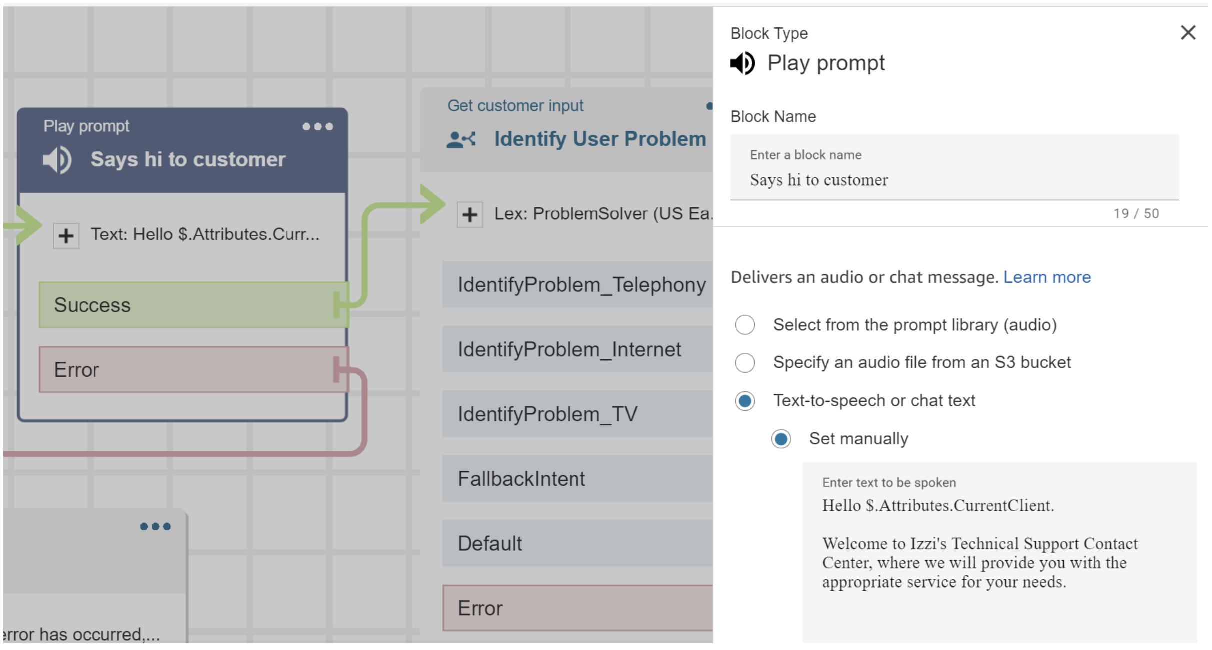
The retrieved customer name is stored as a contact attribute and is used in the next "Play prompt" block so that the customer hears a welcome message with their name and instructions on how to proceed.



Set attributes on
Current contact

"Current contact" is the contact that this flow is running on. "Related contact" is the contact that has been optionally associated with the current contact allowing attribute sharing. "Flow" is the current flow. Learn more

Namespace: User defined
Key: CurrentClient
Set manually
[✓ Set dynamically]
Namespace: External
Key: clientName



Block Type
Play prompt

Block Name
Enter a block name: Says hi to customer
19 / 50

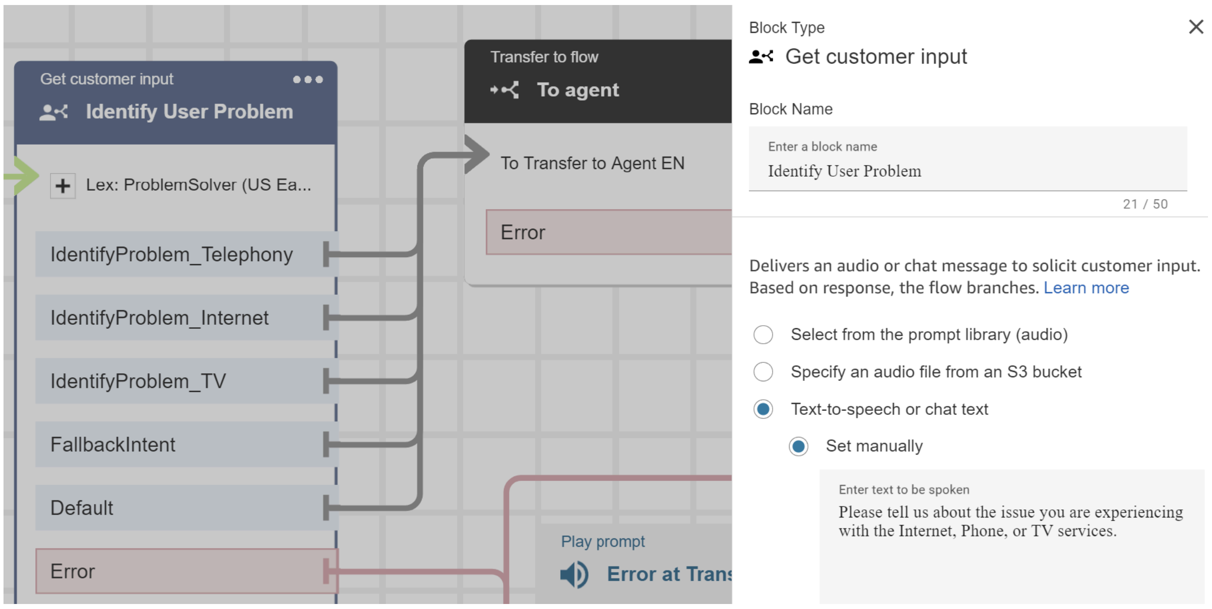
Delivers an audio or chat message. Learn more

Select from the prompt library (audio)
Specify an audio file from an S3 bucket
[✓ Text-to-speech or chat text]
[✓ Set manually]

Enter text to be spoken
Hello $.Attributes.CurrentClient.

Welcome to Izzi's Technical Support Contact Center, where we will provide you with the appropriate service for your needs.

Subsequently, we use a "Get customer input" block, which is directly integrated with an Amazon Lex bot capable of identifying the customer's issue. Depending on the problem, the customer is redirected to a specialized agent using the "Transfer to flow" block.



Block Type
Get customer input

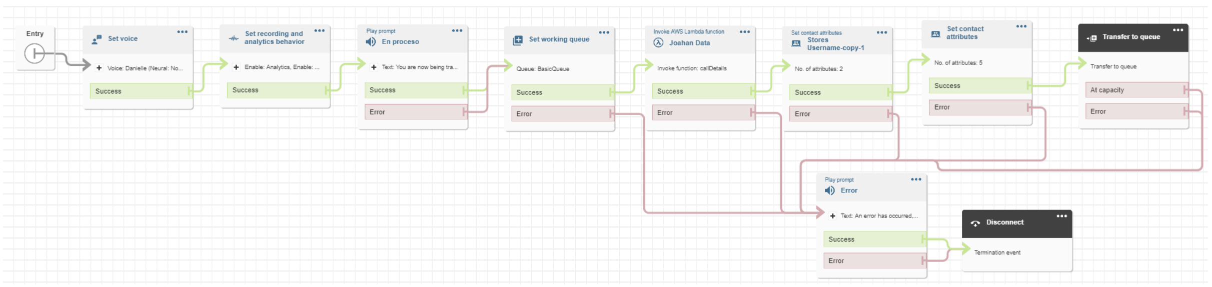
Block Name
Enter a block name: Identify User Problem
21 / 50

Delivers an audio or chat message to solicit customer input. Based on response, the flow branches. Learn more

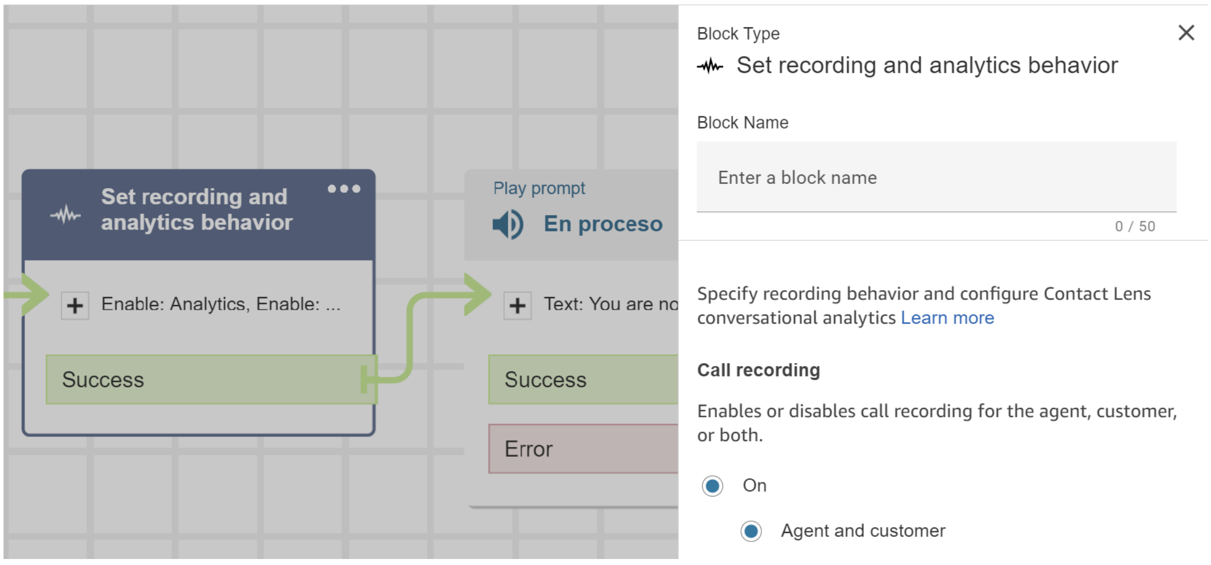
- ( ) Select from the prompt library (audio)
- ( ) Specify an audio file from an S3 bucket
- [✓ Text-to-speech or chat text]
  - [✓ Set manually]

Enter text to be spoken: Please tell us about the issue you are experiencing with the Internet, Phone, or TV services.

## [2] Transfer to agent contact flow



The first step in this flow is to enable call recording to initialize Contact Lens using the "Set recording and analytics behavior" block.


Set recording and analytics behavior
Enable: Analytics, Enable: ...
Success
Play prompt
En proceso
Text: You are no
Success
Error

Block Type

Set recording and analytics behavior

Block Name

Enter a block name

0 / 50

Specify recording behavior and configure Contact Lens conversational analytics Learn more

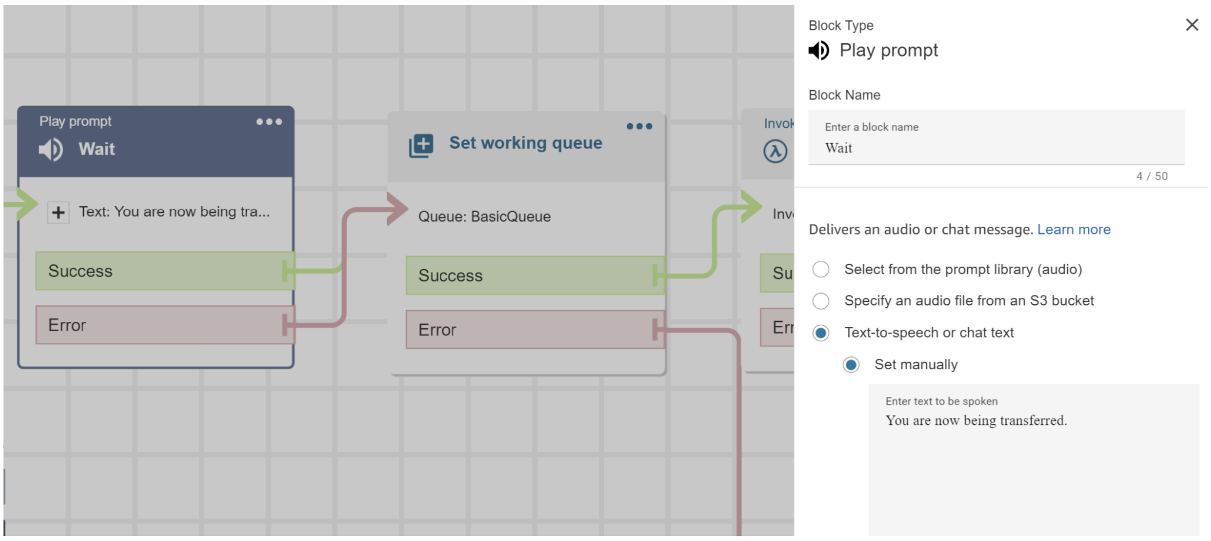
**Call recording**

Enables or disables call recording for the agent, customer, or both.

[✓ On]

[✓ Agent and customer]

Once Contact Lens is initialized, we use a "Play prompt" block to inform the user that they are on hold while being transferred to an agent. A queue is set up to send the customer there.



Block Type

Play prompt

Block Name

Enter a block name

Wait

4 / 50

Delivers an audio or chat message. Learn more

Select from the prompt library (audio)

Specify an audio file from an S3 bucket

[✓ Text-to-speech or chat text]

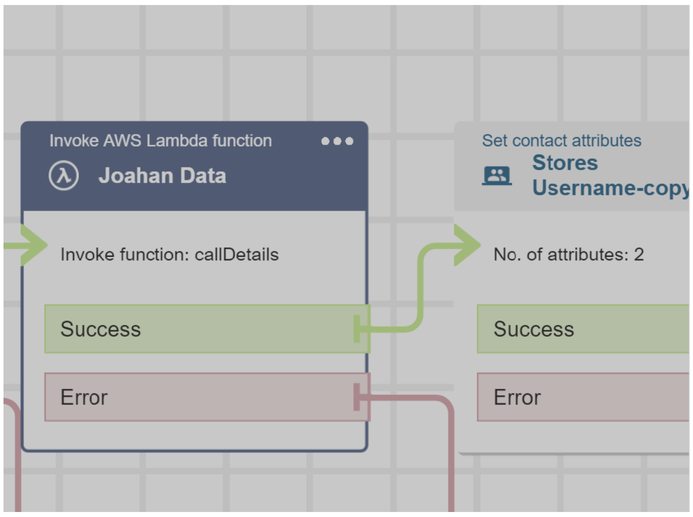
[✓ Set manually]

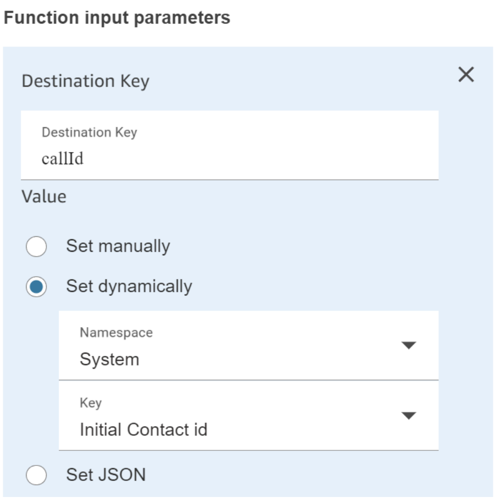
Enter text to be spoken

You are now being transferred.

Again, a Lambda function is used through the "Invoke AWS Lambda Function" block to extract data from our database hosted on Amazon RDS and save it as contact attributes to be displayed on our frontend later.

To see more details about the lambda function, click here.




**Function input parameters**

Destination Key

Destination Key: callId

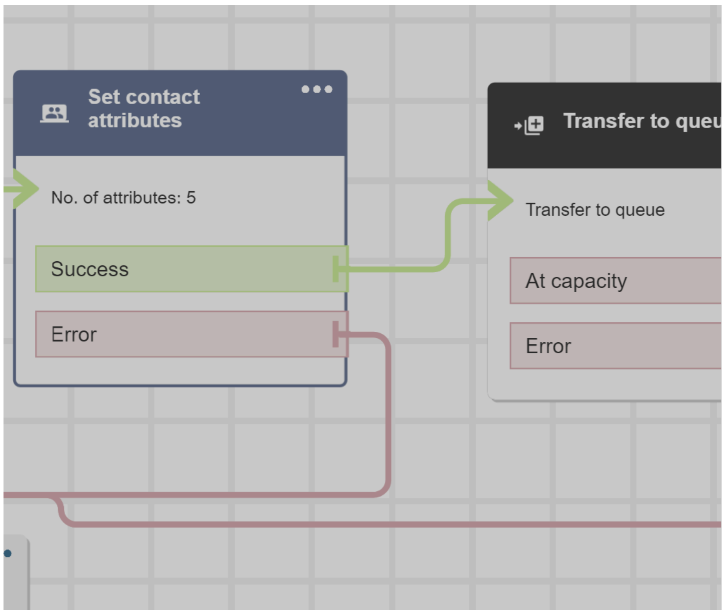
Value

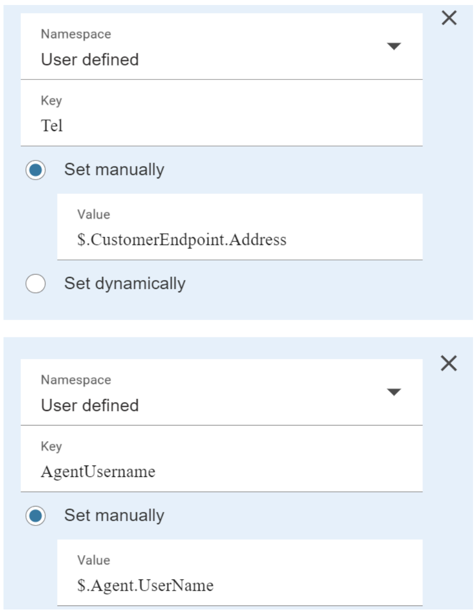
- Set manually
- [✓ Set dynamically]

Namespace: System

Key: Initial Contact id

- Set JSON




Namespace: User defined

Key: Tel

- [✓ Set manually]

Value: $.CustomerEndpoint.Address

- Set dynamically

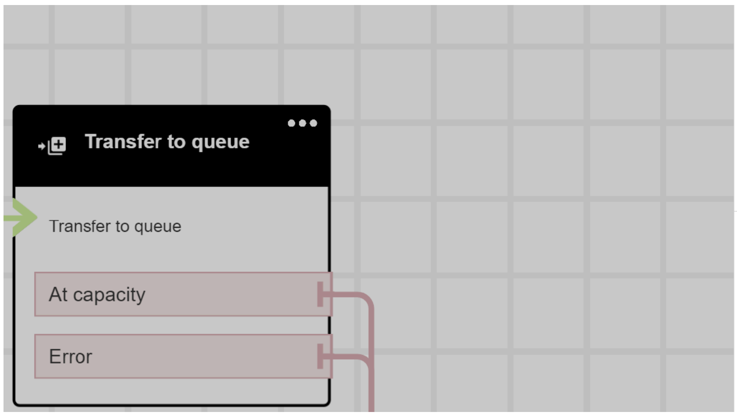
Namespace: User defined

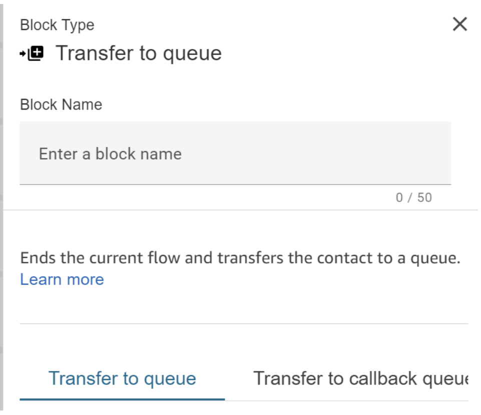
Key: AgentUsername

- [✓ Set manually]

Value: $.Agent.UserName

Finally, the customer is transferred to an agent using the "Transfer to queue" block.




Block Type

Transfer to queue

Block Name

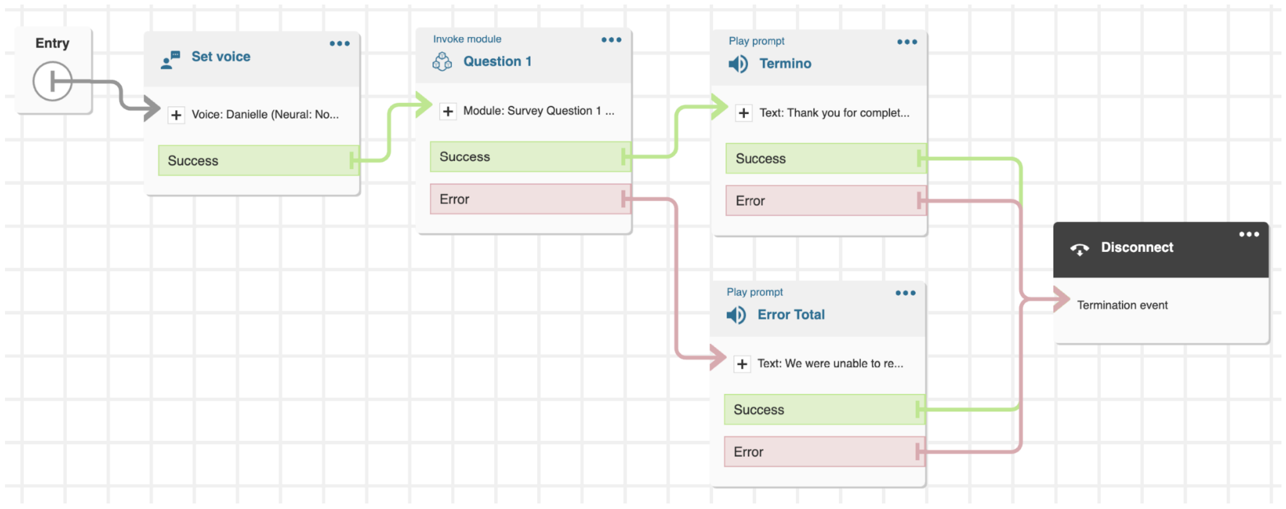
Enter a block name

0 / 50

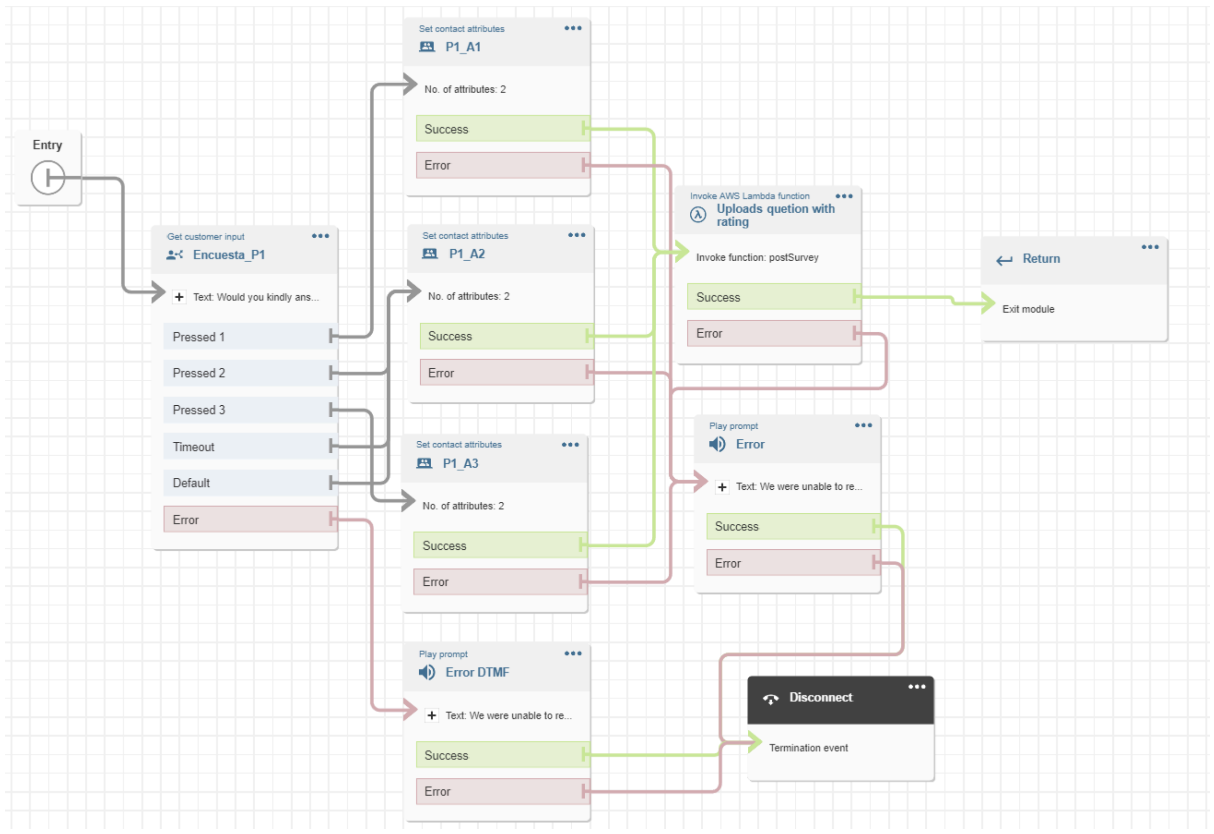
Ends the current flow and transfers the contact to a queue. Learn more

[✓ Transfer to queue] Transfer to callback queue

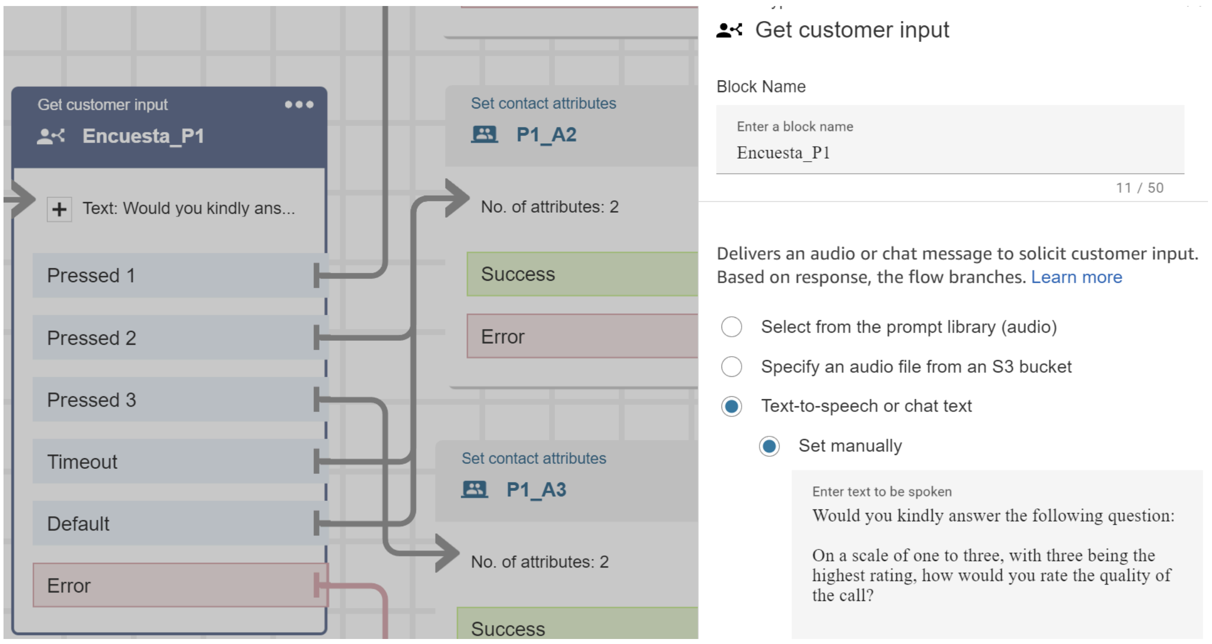
## [3] Survey contact flow



This flow is set to work at the end of the call when the agent has finished it. The customer is asked to answer a satisfactory survey related to the service they've encountered. Rating the clients satisfaction from 1 to 3 based on the experience during the call. The flow then redirects the client to the module of the first question. After answering it, finish with a goodbye and end the call.



This module includes a satisfaction survey conducted at the end of the call. Using a "Get customer input" block configured to work with DTMF, the user can enter a number from 1 to 3 to rate the call quality.



Get customer input

Block Name

Enter a block name
Encuesta_P1

11 / 50

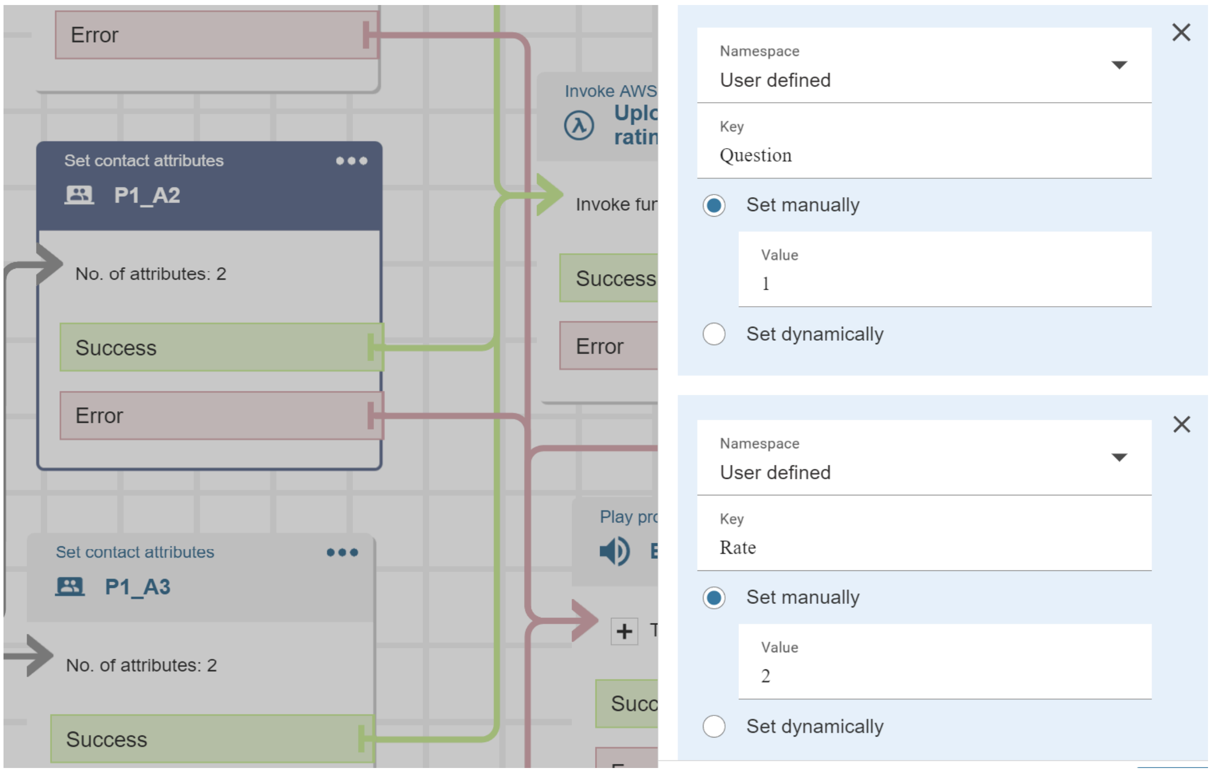
Delivers an audio or chat message to solicit customer input. Based on response, the flow branches. Learn more

- Select from the prompt library (audio)
- Specify an audio file from an S3 bucket
- [✓ Text-to-speech or chat text]
  - [✓ Set manually]

Enter text to be spoken
Would you kindly answer the following question:

On a scale of one to three, with three being the highest rating, how would you rate the quality of the call?

After detecting the number entered by the customer, the user's response is saved as a call attribute using the "Set contact attributes" block.



Namespace
User defined

Key
Question

- [✓ Set manually]

Value
1

- Set dynamically

Namespace
User defined

Key
Rate

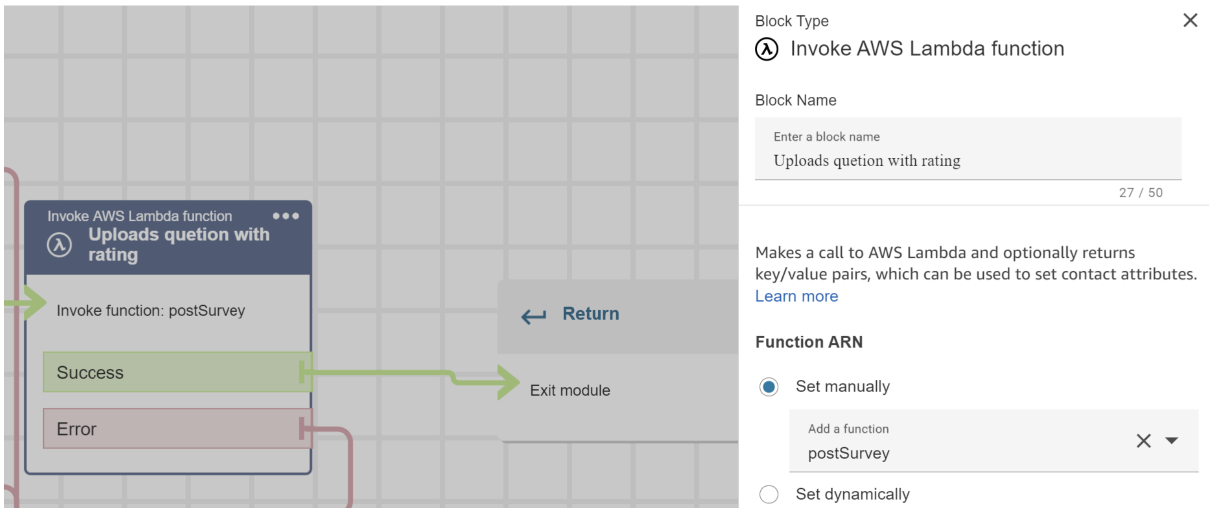
- [✓ Set manually]

Value
2

- Set dynamically

Finally, a last Lambda function called "postSurvey" is invoked with the "Invoke AWS Lambda Function" block.

To see more details about the lambda function, [click here.]


Invoke AWS Lambda function
Uploads question with rating
Invoke function: postSurvey
Success
Error
Return
Exit module

Block Type

Invoke AWS Lambda function

Block Name

Enter a block name

Uploads quetion with rating

27 / 50

Makes a call to AWS Lambda and optionally returns key/value pairs, which can be used to set contact attributes. Learn more

Function ARN

[✓ Set manually]

Add a function

postSurvey

Set dynamically

## Amazon Lex

Amazon Lex is implemented to gather client information regarding the issue they might be facing. This bot consists of 7 different intents, each of them aligned with the most common issues Izzi Telecom has to deal with everyday.

| | |
|---|---|
| IdentifyProblem_TV | Bot asks customers to talk about the problem they are currently facing and sends the transcription to a DB. Focusing on T.V. problems. |
| IdentifyProblem_Internet | Bot asks customers to talk about the problem they are currently facing and sends the transcription to a DB. Focusing on Internet problems. |
| IdentifyProblem_Telephony | Bot asks customers to talk about the problem they are currently facing and sends the transcription to a DB. Focusing on Telephony problems. |
| IdentifyProblem_Incidents | Bot asks customers to talk about the problem they are currently facing and sends the transcription to a DB. Focusing on receiving major incidents. |
| IdentifyProblem_Customer | Bot asks customers to talk about the problem they are currently facing and sends the transcription to a DB. Focusing on customer data problems. |
| Sales | Bot listens to customers talk about wanting to hire new services. |
| Doubts_Customer | Bot asks customers to talk about their concerns about the service they have hired. |
| FallbackIntent | Default intent when no other intent matches |

Each intent has an integrated lambda function that uploads the user voice input to the database. More details about this function are given in the next section.

** To integrate Amazon Lex into an Amazon Connect flow, a series of steps must be followed. To see these steps, click here.

## Lambda functions

A total of four Lambda functions are used in our established flows. As previously mentioned, the primary function of each of these is to connect to a relational database hosted on Amazon RDS to query and upload data.

Three of these functions were already mentioned in the previous section. However, there is a fourth Lambda function directly integrated within the bot created in Amazon Lex. This Lambda function is solely responsible for uploading data to the database.

** It is important to note that to integrate any Lambda function with Amazon Connect, a series of steps must be followed. To see these steps, click here.

** Note that all lambda functions require the following environment variables to be set:

DB_HOST: The hostname of the MySQL database.
DB_PORT: The port number of the MySQL database (default is 3306).
DB_USER: The username for the MySQL database.
DB_PASSWORD: The password for the MySQL database.
DB_NAME: The name of the MySQL database.

### [1] clientLookup

JavaScript
```javascript
const mysql = require('mysql2/promise');

const handler = async (event) => {

    // Environment variables
      const dbHost = process.env.DB_HOST;
      const dbPort = process.env.DB_PORT || "3306";
      const dbUser = process.env.DB_USER;
      const dbPassword = process.env.DB_PASSWORD;
      const dbName = process.env.DB_NAME;

      // Create a connection to the database
      const connection = await mysql.createConnection({
            host: dbHost,
            port: dbPort,
            user: dbUser,
            password: dbPassword,
            database: dbName
      });

      const                          phoneNumber                         =
event.Details.ContactData.CustomerEndpoint.Address;
```

```
        try {
                // Retrieves info from the customer from the database
                const [rows] = await connection.execute(
                "SELECT Nombre FROM Cliente WHERE Celular = ?",
                [phoneNumber]
                );

                let nombreCliente = "";

                if (rows.length > 0) {
                        nombreCliente = rows[0].Nombre;
                }

                const response = {
                        "clientName": nombreCliente
                };

                return response;

        } catch (error) {
                console.error("Error while connecting to DB:", error);
                throw error;
        } finally {
                await connection.end();
        }
};
module.exports = { handler };
```

This Lambda function receives a default JSON parameter that Connect automatically sends when the "Invoke AWS Lambda Function" block is used. Thanks to this JSON, we can obtain the phone number from which the customer is calling. We then use this number to search for the record in our relational database hosted on RDS and return the name of the customer to whom the phone number belongs.

To see more information about handling this JSON, click here.

## [2] Upload Issue

```javascript
const mysql = require('mysql2/promise');

const handler = async (event) => {
    // Environment variables
    const dbHost = process.env.DB_HOST;
    const dbPort = process.env.DB_PORT || "3306";
    const dbUser = process.env.DB_USER;
    const dbPassword = process.env.DB_PASSWORD;
    const dbName = process.env.DB_NAME;

    // Create a connection to the database
    const connection = await mysql.createConnection({
        host: dbHost,
        port: dbPort,
        user: dbUser,
        password: dbPassword,
        database: dbName
    });

    // Function that calculates and returns the current date and time
    // as a string, adjusted for a specific timezone offset
    const getDateTime = () => {
        const date = new Date();
        const utcDate = Date.UTC(
            date.getUTCFullYear(),
            date.getUTCMonth(),
            date.getUTCDate(),
            date.getUTCHours(),
            date.getUTCMinutes(),
            date.getUTCSeconds()
        );
        const offset = -6;
        const localDate = new Date(utcDate + offset * 3600 * 1000);
        return localDate.toISOString().slice(0, 19).replace('T', ' ');
    };

    // Map possible issues
    const possibleIssues = {
        "IdentifyProblem_Telephony": "telefonia",
        "IdentifyProblem_Incidents": "soporte",
        "ItendifyProblem_Customer": "cliente",
        "IdentifyProblem_Internet": "internet",
        "IdentifyProblem_TV": "television",
        "Doubts_Customer": "dudas",
        "Sales": "ventas",
        "FallbackIntent": "ERROR"
    };

     const callId = event.sessionState.sessionAttributes.callId; // Primary
Key
    const inputText = event.inputTranscript;
    const issue = event.interpretations[0].intent.name;
    const status = true;
                                    const         sentimentPreview         =
event.interpretations[0].sentimentResponse.sentiment;
        const  clientId  =  event.sessionState.sessionAttributes.clientId;  //
Foreign Key

    const response = {
        "sessionState": {
            "dialogAction": {
                "type": "Close"
            },
            "intent": {
                "name": issue,
                "state": "Fulfilled"
            }
        }
    };

    try {
        // Insert issue in the database
        const [rows] = await connection.execute(
                 "INSERT  INTO  Llamada  (IdLlamada,  FechaHora,  Notas,  Estado,
Sentiment, Asunto, Celular) VALUES (?, ?, ?, ?, ?, ?, ?)",
                              [callId,   getDateTime(),   inputText,   status,
sentimentPreview.toLowerCase(), possibleIssues[issue], clientId]
        );

        return response;

    } catch (error) {
        console.error("Error while connecting to DB:", error);
    } finally {
        await connection.end();
    }
};
module.exports = { handler };
```

This function takes as a parameter the name of the Amazon Lex bot current active intent and the customer voice input transcript to upload them to the database, such as the current call sentiment and the initial start time of the call.

### [3] callDetails

JavaScript
```javascript
const mysql = require('mysql2/promise');

const handler = async (event) => {

    // Environment variables
       const dbHost = process.env.DB_HOST;
       const dbPort = process.env.DB_PORT || "3306";
       const dbUser = process.env.DB_USER;
       const dbPassword = process.env.DB_PASSWORD;
       const dbName = process.env.DB_NAME;

       // Create a connection to the database
       const connection = await mysql.createConnection({
              host: dbHost,
              port: dbPort,
              user: dbUser,
              password: dbPassword,
              database: dbName
       });

       const callId = event.Details.Parameters.callId;
       console.log("Call Id:", callId);


       try {
              // Retrieves info from the current call from the database
              const [rows] = await connection.execute(
              "SELECT Notas, Asunto FROM Llamada WHERE IdLlamada = ?",
              [callId]
              );


              let notasLlamada = "InputCliente";
              let asuntoLlamada = "Problema";

              if (rows.length > 0) {
                     notasLlamada = rows[0].Notas;
                     asuntoLlamada = rows[0].Asunto;
              }

              const response = {
                     "callNotes": notasLlamada,
                     "callIssue": asuntoLlamada
              };

              console.log(response);

              return response;
```

```
      } catch (error) {
            console.error("Error while connecting to DB:", error);
            throw error;
      } finally {
            await connection.end();
      }
};
module.exports = { handler };
```

This function takes the Initial Contact Id as a parameter to make the request and fetch the information regarded by the IVR such as the main issue and the description provided by the client.

## [4] postSurvey

JavaScript
```javascript
const mysql = require('mysql2/promise');

const handler = async (event) => {

    // Environment variables
    const dbHost = process.env.DB_HOST;
    const dbPort = process.env.DB_PORT || "3306";
    const dbUser = process.env.DB_USER;
    const dbPassword = process.env.DB_PASSWORD;
    const dbName = process.env.DB_NAME;

    // Create a connection to the database
    const connection = await mysql.createConnection({
        host: dbHost,
        port: dbPort,
        user: dbUser,
        password: dbPassword,
        database: dbName
    });

    // Data to be inserted
    const callId = event.Details.ContactData.InitialContactId;
    const question = event.Details.ContactData.Attributes.Question;
      const  rate  =  event.Details.ContactData.Attributes.Rate;  //  Customer's
input

    try {
        // Insert survey into the database
        const [rows] = await connection.execute(
             "INSERT INTO Encuesta (Pregunta, Calificacion, IdLlamada) VALUES
(?, ?, ?)",
            [question, rate, callId]
        );

        const response = {
            statusCode: 200,
            body: JSON.stringify('Successful'),
        };

        return response;

    } catch (error) {
        console.error("Error while connecting to DB:", error);
    } finally {
        await connection.end();
    }
};
module.exports = { handler };
```

This function takes the value entered by the customer to rate the survey and uploads it to our database on Amazon RDS.

## SNS

Our approach when designing the app was to provide a differentiated service to our Izzi customers. Our system aims to anticipate customer calls by sending notifications to clients in a specific area.

### General Information

| | |
|---|---|
| Topic | Incidents |
| ARN | arn:aws:sns:us-east-1:905418447691 |
| Type | Standard |
| Owner’s topic | 905418447691 |

### Acces policy

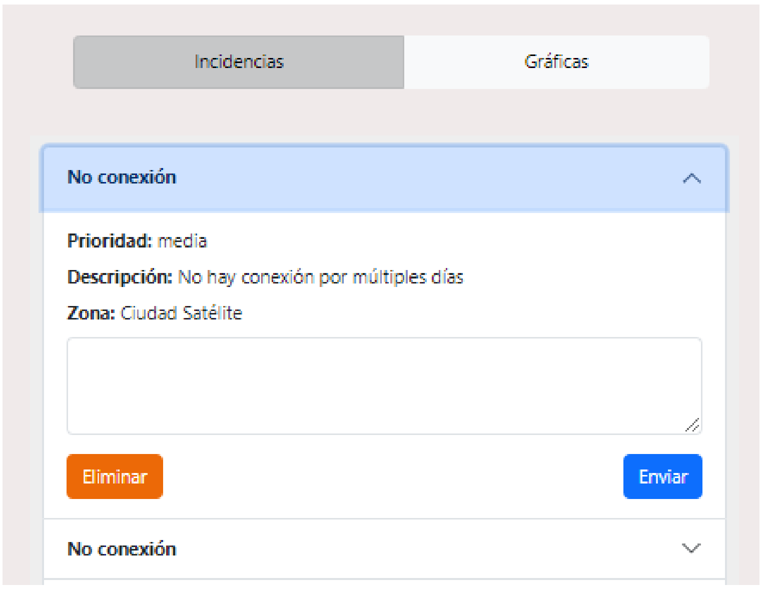
The access policy defines who can publish or subscribe to this topic and what actions they can perform. Here, the detailed access policy should be included, typically specified in JSON format.

```javascript
{
  "Version": "2008-10-17",
  "Id": "__default_policy_ID",
  "Statement": [
    {
      "Sid": "__default_statement_ID",
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": [
        "SNS:Publish",
        "SNS:RemovePermission",
        "SNS:SetTopicAttributes",
        "SNS:DeleteTopic",
        "SNS:ListSubscriptionsByTopic",
        "SNS:GetTopicAttributes",
        "SNS:AddPermission",
        "SNS:Subscribe"
      ],
      "Resource": "arn:aws:sns:us-east-1:905418447691:Incidencias",
      "Condition": {
        "StringEquals": {
          "AWS:SourceOwner": "905418447691"
        }
      }
    }
  ]
}
```

- Access key ID : AKIA5FTZFDNFSXNNRAMU
- Secret access key :  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

## Usage

When an incident is reported, it appears on the supervisor's screen, where a message can be written to everyone in a specific area.



**Prioridad:** media

**Descripción:** No hay conexión por múltiples días

**Zona:** Ciudad Satélite

Whatever is written in this text will pass to the following code, where it will retrieve all the numbers from this area, and subsequently send the message to the obtained numbers.

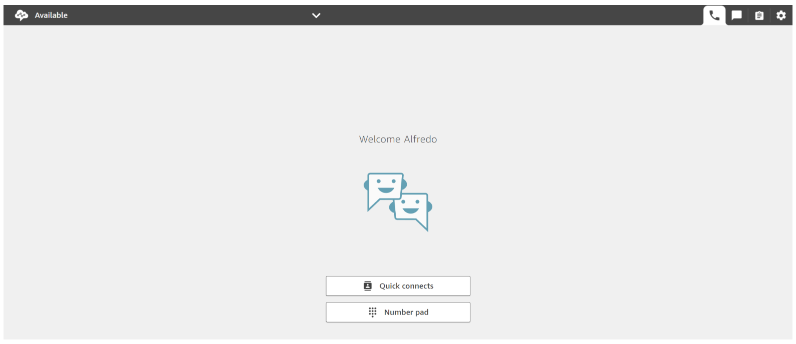
Unset
```
// Function that take the text of the message and send the message to the
users in the zone of the incident
  const handleSend = async () => {
    setButtonText('Enviando...');

    try {

      // Fetch the phone numbers of the users in the zone of the incident
                                                const      response      =      await
fetch(`http://${url}/cliente/telefonoPorZona/${zone}`, {
        headers: { Authorization: `Bearer ${token}` }
      });
      if (!response.ok) {
                      throw   new   Error(`Error   fetching   phone   numbers:
${response.statusText}`);
      }
      const data = await response.json();
      console.log(data);

      // Send the message to the users in the zone of the incident
      await Promise.all(data.map(async (element) => {
        const sendMessageResponse = await fetch(
          `http://${url}/sns/send-message`,
          {
            method: 'POST',
            headers: {
              'Content-Type': 'application/json',
              Authorization: `Bearer ${token}`
            },
            body: JSON.stringify({
              phoneNumber: element.Celular,
              message
            })
          }
        );
        if (!sendMessageResponse.ok) {
                                throw    new    Error(`Error    sending    message:
${sendMessageResponse.statusText}`);
        }
      }));
```

This request reaches the backend, which will call the aws-sdk services to allow sending a message to each one.

```
Unset
private async sendMessage(req: Request, res: Response) {
        try {
            const { phoneNumber, message } = req.body;

            // Configurar AWS con tus credenciales
            AWS.config.update({
                accessKeyId: AWS_ACCESS_KEY_ID_C,
                secretAccessKey: AWS_SECRET_ACCESS_KEY_C,
                region: AWS_REGION // Por ejemplo, 'us-east-1'
            });

            // Crear un nuevo objeto SNS
            const sns = new AWS.SNS();

            console.log("phoneNumber", phoneNumber);
            console.log("message", message);

            // Definir el mensaje que deseas enviar
            const params = {
                Message: message,
                PhoneNumber: phoneNumber
            };

            // Enviar el mensaje
            const data = await sns.publish(params).promise();
            console.log("Mensaje enviado con éxito:", data);
            res.status(200).json({ message: "Mensaje enviado con éxito" });
        } catch (error) {
            console.error('Error al enviar el mensaje:', error);
            res.status(500).json({ error: 'Error al enviar el mensaje' });
        }
    }
```

## Initialize CCP

La librería Amazon Connect Streams provee una manera de embeber el CCP directamente en nuestra aplicación web. Para ello sugerimos consultar el siguiente enlace.


Available
Welcome Alfredo
Quick connects
Number pad