

SuperOffice Support Chatbot

How to integrate your chatbot into SuperOffice CRM (From R11).

Table of content:

1. [Introduction](#)
 - 1.1 [What is the goal?](#)
 - 1.2 [Customers point of view.](#)
 - 1.3 [What you won't find in this documentation.](#)
 - 1.4 [Recommended way of starting a project.](#)
2. [How does the integration work?](#)
 - 2.1 [Diagram](#)
 - 2.2 ["New chat session" trigger.](#)
 - 2.3 ["After saving chat message" trigger.](#)
 - 2.4 ["Chat session changed status" trigger.](#)
3. [Types of intents returned.](#)
 - 3.1 [AgentHandover](#)
 - 3.2 [DoneConversation](#)
 - 3.3 [Regular intents](#)
 - 3.4 [Fallback intents](#)
4. [How does the bot decide what to do?](#)
 - 4.1 [Diagram](#)
 - 4.2 [Code snippet](#)
5. [What needs to be done on Dialogflows side?](#)
 - 5.1 [Activating the API](#)
 - 5.2 [Getting the client ID and client secret.](#)
 - 5.3 [Getting the access and refresh tokens.](#)
6. [What needs to be done in SuperOffice?](#)
 - 6.1 [Configuring "After saving chat message".](#)
 - 6.2 [Configuring "Chat changed status"](#)
 - 6.3 [Configuring "New chat session"](#)
7. [Demos / Example scenarios](#)
 - 7.1 [Scenario #1](#)
 - 7.2 [Scenario #2](#)
 - 7.3 [Scenario #3](#)
8. [Content](#)
 - 8.1 [URL's](#)
 - 8.2 [Full API response](#)

Introduction

The purpose of this documentation is to give some insights on how you can integrate any external chatbot into SuperOffice.

The goal is to make a seamless integration to SuperOffice, that enables all the functionalities SuperOffice has to offer. But where the dialog itself is powered by a third party chatbot.

From the customers point of view, the chat session will be initialized from SuperOffice chat. The customer starts the chat session by clicking on the chat widget. They are then presented with a greeting message, and from here on they are chatting with a bot.

If the bot is unsuccessful with presenting the right answer, it will be possible to be “handed over” to a human agent instead. This will either be to a SuperOffice chat agent(*If present for the specific chat channel*), or a request will be created and registered on behalf of the customer in SuperOffice Service.

If the bot successfully manages to find the right solution and no agent interaction is required, a closed request will be created for keeping statistics. In the script, you have the possibility to set some rules for these requests. For example, you could choose to only create a request if the session had more than X messages. It could also be triggered only when the customer confirms that the chat dialog has helped them complete their task.

This documentation will cover how all of this comes together. It will also give insights on how to configure your chatbot in Dialogflow.

This documentation does not cover how a chatbot works. Reading this documentation will **not** answer any of the following questions:

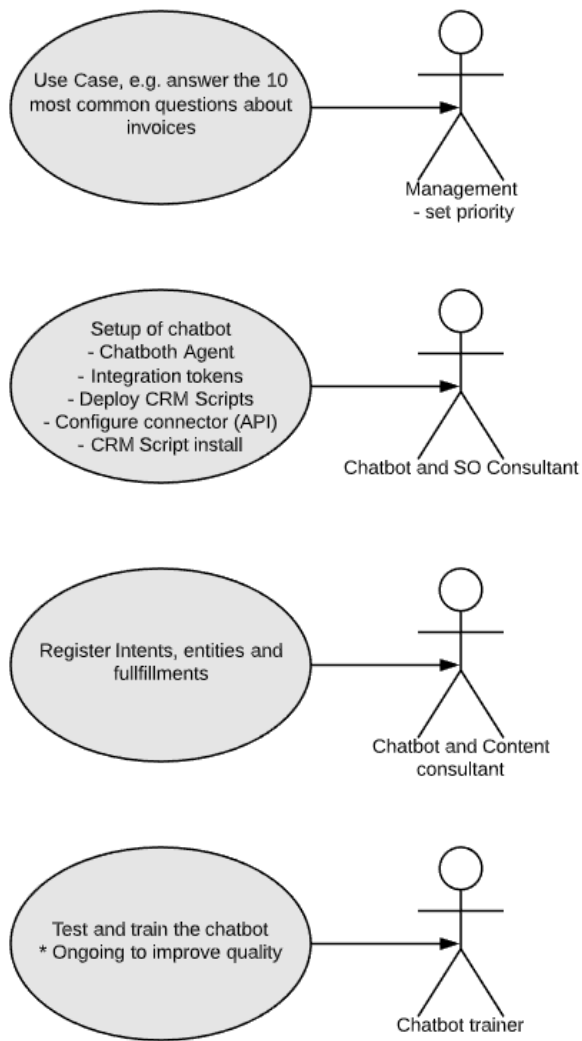
- What is a chatbot?
- How does it work?
- What is an intent?
- How do I configure training phrases?
- How do I configure a handover intent?
- What is context, and how can I save input values for later use?
- How do I measure the accuracy of my chatbot?

The topics above are advanced enough on its own, and are vital to have in place if you are planning to have this solution implemented successfully.

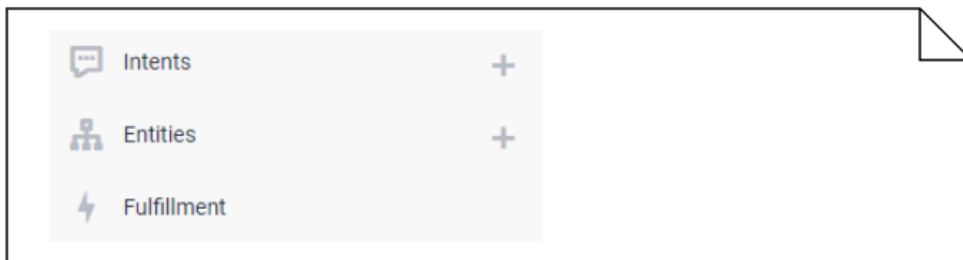
The complexity of the questions above are not to be taken lightly, and will be the greatest challenge in order to successfully implement a chatbot solution into your SuperOffice CRM.

Recommended way of starting a project:

If you are planning on integrating your SuperOffice CRM with an external chatbot provider, the following scheme shows the scope of such project. This diagram explains the recommended project participants, and the responsibilities for each member(s):



1. What's the problem and how will you solve it?
2. What will the bot do?
3. How will you build the bot?
4. How will you know if the bot is successful?
5. How will you launch the bot?



Following the hierarchy shown above will give you great conditions in bringing your chatbot into production with high efficiency.

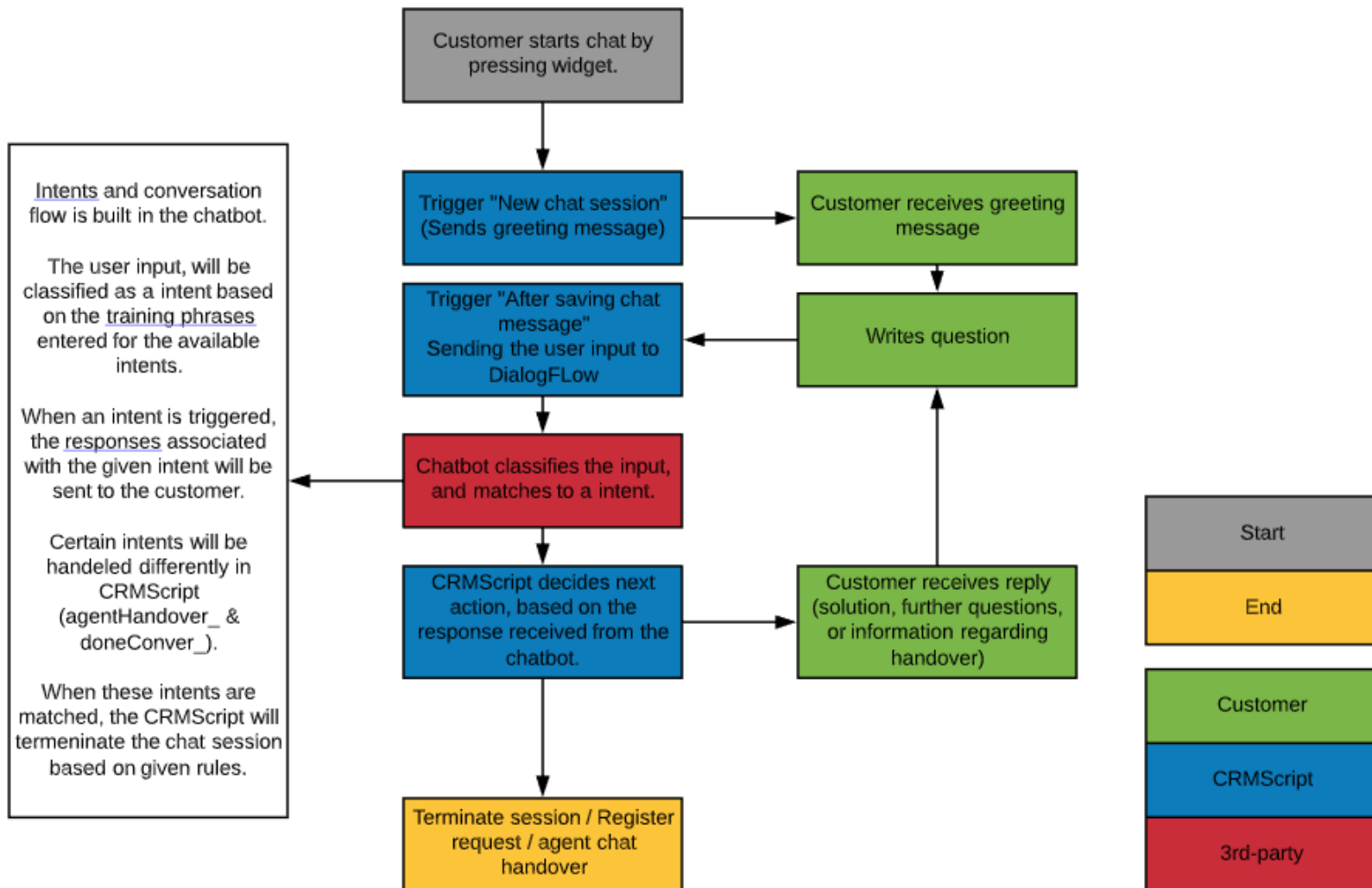
How does the integration work?

The integration is built in CRMScript. It requires 3 triggers to work:

- New chat session
- After saving chat message
- After changing status

“After saving chat message” communicates with the chosen providers REST API. Tokens for the providers API will be stored in the SuperOffice database.

A typical conversation can be described as following:



New chat session" trigger.

Each session is started by a person clicking the chat widget. This will execute the trigger “New chat session”.

```
#setLanguageLevel 3;

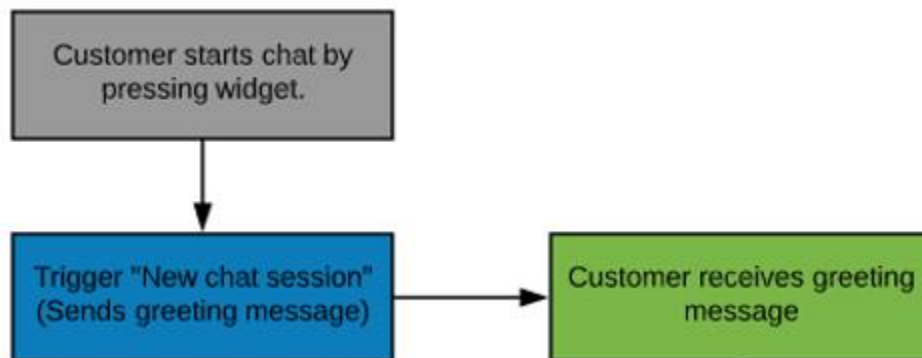
EventData ed = getEventData();
String chatSessionId = ed.getInputValue("chatSession.id");

SearchEngine getFirstName;
getFirstName.addField("chat_session.customer_id.firstname");
getFirstName.addCriteria("chat_session.id", "Equals", chatSessionId);
getFirstName.execute();
String firstName = getFirstName.getField("chat_session.customer_id.firstname");

if(firstName == ""){
    addChatMessage(chatSessionId.toInteger(),"Hello. \n \n I'm James - a digital assistant. \n How may I help you?",1,"James",0,"");
}else{
    addChatMessage(chatSessionId.toInteger(),"Hello "+firstName+". \n \n I'm James - a digital assistant. \n How may I help you?",1,"James",0,"");
}
```

If the person is logged in through Customer Portal, the persons first name will be included in the greeting message. If the conversation is initialized by an unknown customer, a generic greeting message will be sent instead.

This explains how the first part of the diagram is designed to work.



This trigger is required in order to avoid running into problems when using the prechat-form. The prechat form is used when identifying a customer using the [resetChat\(\)](#) function.

After the session has been initialized, the flow will follow as demonstrated by the diagram on the previous page.

After saving chat message trigger.

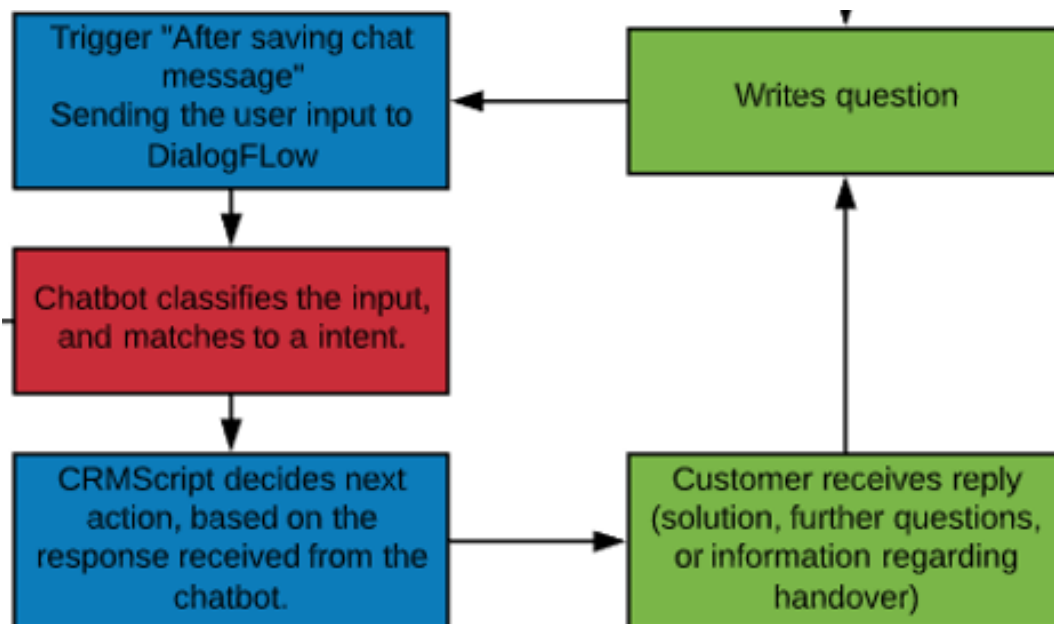
Whenever a person replies to the greeting message, it will execute the next trigger “After saving chat message”. This trigger takes the users input and sends it to the chatbot, returns a response and thereafter takes an action based on it.

We only want this trigger to execute during certain conditions. For example, if a session has been handed over to an agent, the trigger should not execute. It should also only trigger when the user sent the message.

These conditions are met with the following logics:

```
//Saving input data from the trigger.  
String chatSessionId = ed.getInputValue("chatMessage.sessionId"); //The ID for the specific chat session.  
String text_item      = ed.getInputValue("chatMessage.message"); //The message sent by the user.  
String specialType    = ed.getInputValue("chatMessage.specialType"); //Only for regular text messages.  
String userId         = ed.getInputValue("chatSession.userId"); //Will not be -1 if the request has been handed over.  
String chatType       = ed.getInputValue("chatMessage.type"); // Only for incoming messages.  
  
//Only trigger if certain conditions are met.  
if(userId == "-1" && specialType != "10" && specialType != "11" && chatType == "2"){
```

The flow of the dialog from this point on, will be chosen depending on the returned values from the Dialogflow API.



Chat session changed status trigger

When the entire conversation is between bot <-> agent, we won't see the conversation history easily from GUI. The purpose of this trigger is to keep better track of those conversations.

After a conversation without any agent interaction has been terminated, this trigger will save the transcript as a closed request in SuperOffice Service. This trigger only executes when the chat is terminated by the customer. It will only save the request when the conversation had more than X messages. You can set this limit yourself, and it is meant to help with preventing us to register "spammy" requests.

This trigger is optional for the integration.

Types of intents returned

There are 2 types of intents that are different from the rest. Those are the *AgentHandover* and *DoneConversation*. If any of these 2 intents are triggered, special actions will be taken.

AgentHandover

If an intent with a name that starts with AgentHandover is triggered, the session will be handed over to an agent. This typically happens when a chatbot failed to help the customer further.

If there **is an agent present** in the chat, the session will be transferred to the chat queue. When the agent accepts the chat, the history of the conversation will be visible for the agent to quickly get up on speed.

If **no agent is present** in the chat, a request will be registered in SuperOffice Customer Service for the customer. This is done using the [resetChat\(\)](#) function. The customer will be notified that a request has been created, and presented with the ID of the request.

DoneConversation

If you hit an intent called DoneConversation, the session will be identified as a completed conversation. The session will then be terminated by the script, and a thank you / good bye message will be sent to the customer. A closed request with the transcript will be saved in Customer Service for statistics.

This typically happens when the customer answers “yes” when the bot asks “Did this solve your question?”

Regular intents

If none of those are returned, it will send the response connected to the given intent.

Some intents can have multiple text responses associated to them. The idea behind this, is that a second message sometimes should be sent as a confirmation message. Below is an idea on how this could look like:

-Here is an answer!

Few seconds later

-Was this what you were looking for?

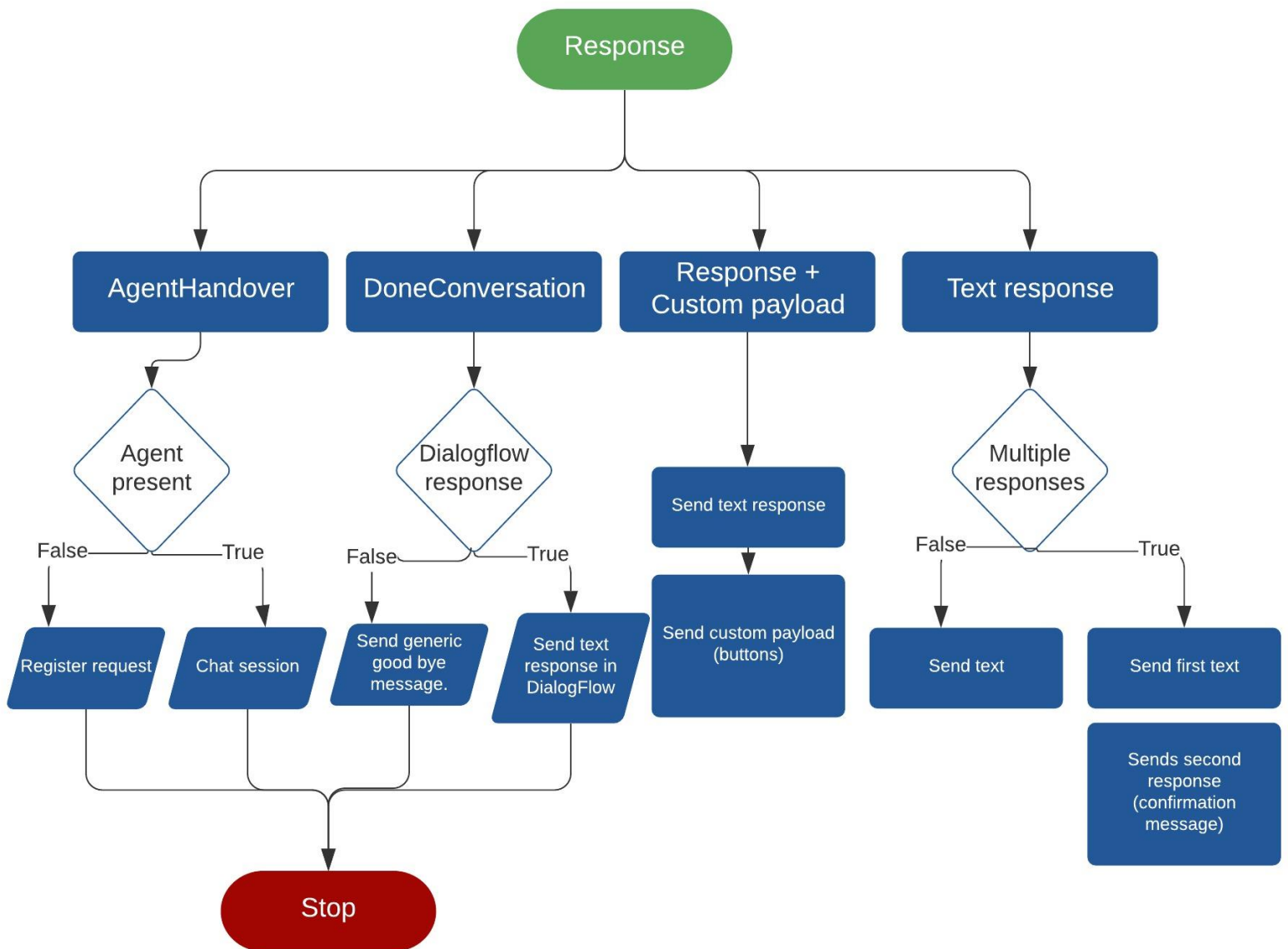
Some intents also has buttons connected to them. These buttons are configured as a custom payload in Dialogflow. This will be used to generate buttons in the SuperOffice Chat. Buttons is a great way of controlling the direction of the conversation, as well as presenting options to the user. The buttons are added using the [addChatMessage\(\)](#) function.

Fallback intents

Define the threshold value for the confidence score. If the returned value is less than the threshold value, then a fallback intent will be triggered, or if there is no fallback intents defined, no intent will be triggered.

How does the script decide what to do?

Depending on the response returned by calling the API, the script can take any of the 4 following paths.



Code snippet

Here is how that diagram translates into the CRMScript used in this integration.

```
//Saving the returned values needed.
XMLNode x2      = parseJSON(getResponse());
String response  = x2.getValueFromPath("1.fulfillmentMessages.0.text.text.0"); //Returned the actual response configured in DialogFlow.
String confirmMsg = x2.getValueFromPath("1.fulfillmentMessages.1.text.text.0"); //Second payload text response, Will be sent as second response.
String intentName = x2.getValueFromPath("1.intent.displayName"); //Returned intent name found in DialogFlow. Will be used to decide action based on intent.
String optionButtons = x2.getValueFromPath("1.fulfillmentMessages.1.payload.options"); //Returned the actual response configured in DialogFlow.
String confirmButtons = x2.getValueFromPath("1.fulfillmentMessages.2.payload.options"); //Returned the actual response configured in DialogFlow.

//DateTimes controlling when to post a message.
DateTime msgDelay;

//Determines next action, based on returned values from DialogFlow.
if(intentName.toUpperCase().startsWith("AGENTHANDOVER_")){//If the returned intent name is a handover intent - handover to agent(chat/ticket).
    addChatMessage(chatSessionId.toInteger(),"I will pass this over to a colleague of mine. Please hang on while I look if someone is available right away.",1,"Hugo",0,"");
    resetChat(chatSessionId.toInteger());
}
else if(intentName.toUpperCase().startsWith("DONECONVER_")){//If a doneconver_ intent is recieved, close the session.
    if(response != ""){ //If the response is set in DialogFlow - send that one.
        addChatMessage(chatSessionId.toInteger(),response.utf8Decode(),1,"James",0,"");
        setChatStatus(chatSessionId.toInteger(), 7);
    }
    else{//If no text response is set in DialogFlow, send the pre-defined one below.
        addChatMessage(chatSessionId.toInteger(),"Great to hear! \ If you have any other questions, please do not hesitate to contact me once again.",1,"James",0,"");
        setChatStatus(chatSessionId.toInteger(), 7);
    }
}
else if(optionButtons.getLength() > 0){//If the response has a custom payload, populate the text response with buttons.
    addChatMessage(chatSessionId.toInteger(), response.utf8Decode(), 1, "James", 17, optionButtons, msgDelay.addSec(2));
}
else if(confirmMsg != ""){ // If it has a second text payload, send the first one, and then the second one.
    addChatMessage(chatSessionId.toInteger(),response.utf8Decode(),1,"James",16,"showAt="+msgDelay.addSec(2).toString());
    addChatMessage(chatSessionId.toInteger(),confirmMsg.utf8Decode(),1,"James",17,confirmButtons,msgDelay.addSec(4));
}
else{
    addChatMessage(chatSessionId.toInteger(),response.utf8Decode(),1,"James",16,"showAt="+ msgDelay.addSec(2).toString());
}
}
```

The Boolean statements shown on the diagram are evaluated using the function [resetChat\(\)](#). All conversations will sooner or later end up with a AgentHandover or DoneConversation intent, and terminate the session.

What needs to be done on Dialogflows side?

In order to use the integration, you will have to activate the DialogFlow API. The API uses OAuth2 authentication. You have to consider 4 distinct roles within this process:

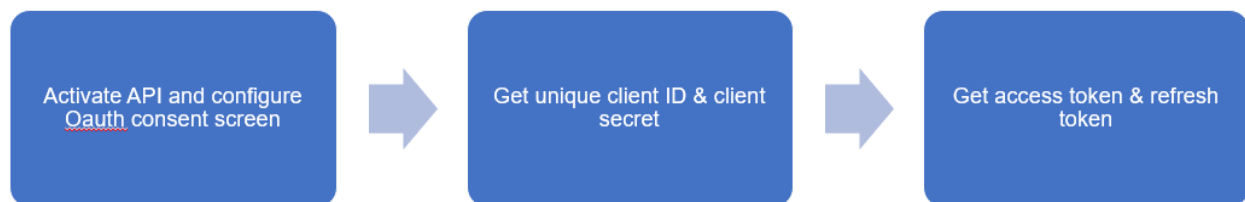
- The resource owner(*The google account that created the agent in DialogFlow*).
- The resource server where the owner stores his data (*DialogFlow Console*).
- The client application that will need access to the resource of the owner (*the integration*).
- The authorization server that grants access to the application. (OAuth Playground).

Whenever the client application (*the integration*) needs to access a resource, it must provide a token. This token is used to access the resource located on the resource server.

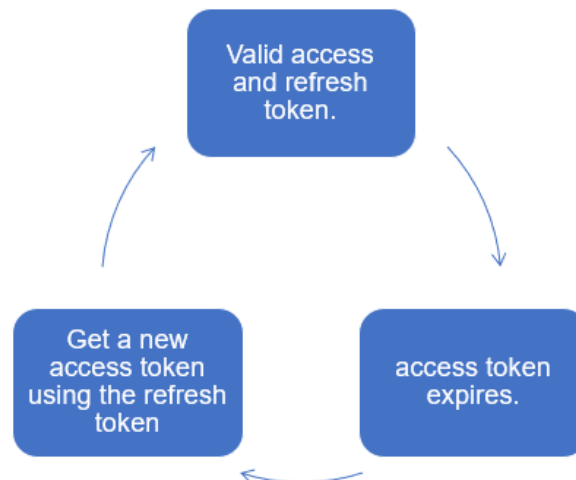
For the integration to work, we will therefor need to fill the following information in the script:

- Client ID
- Client secret
- Access token
- Refresh token

The access and refresh tokens are obtained from the authorization server that grants access to the application. For this we will need a client ID and client secret. This can be visualized in 3 steps:



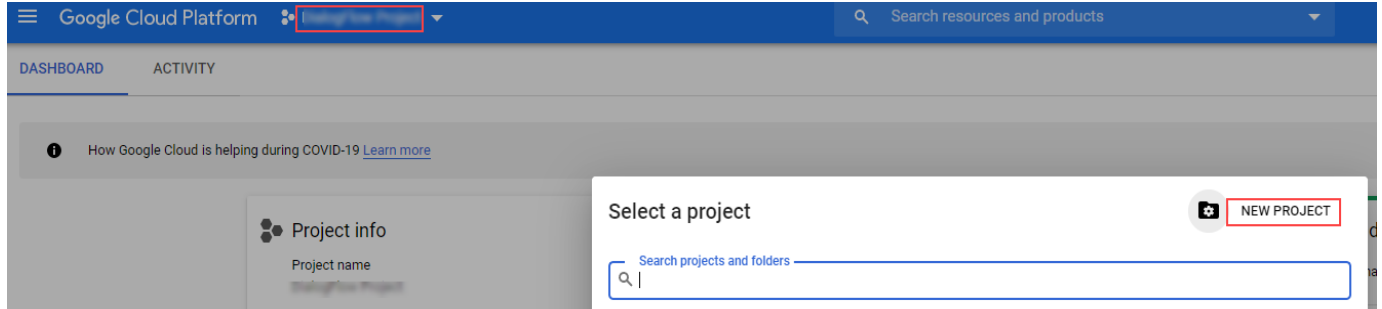
Each access token is valid for 60 minutes. Once it becomes invalid, you can obtain a new access token by using the refresh token. The refresh token will stay the same until revoked. This logic is being taken care of by the script. The following graph visualizes how this looks like in the script:



Activating the API

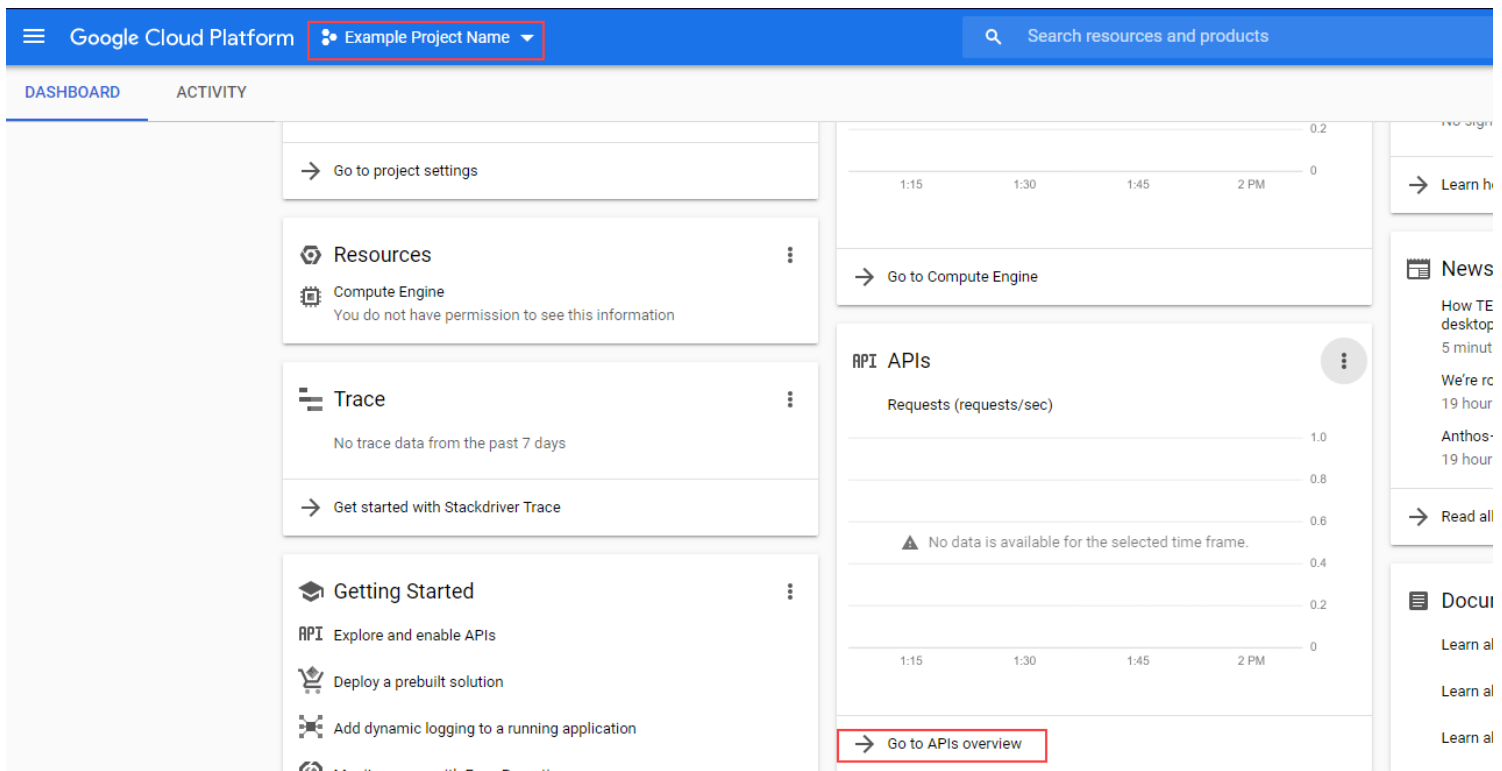
Start by logging into the [Google Cloud Platform](#). Login using the same Google account as the one you created your agents with in DialogFlow.

Once logged in, go to the top left bar on the site, to create a new Project:

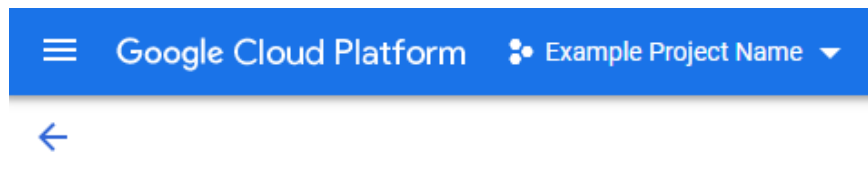
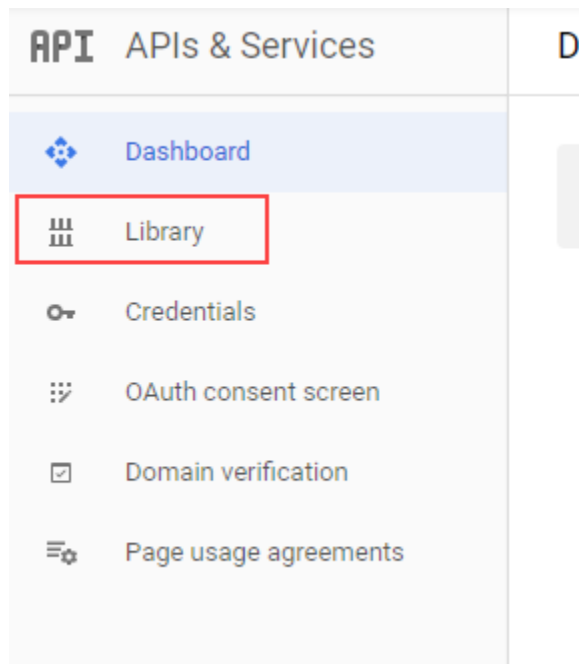


Here you will have to give your project a name. Soon as you are done, press **“Create”**.

Once the project has been created, make sure that you have selected the right project in the top left corner. Thereafter click on **“Go to APIs overview”**.



When you're in the API overview, navigate to "**Library**" in the left hand side menu. In the search bar, search for "Dialogflow". Here you have the possibility to enable to API.



Dialogflow API

Google

Builds conversational interfaces

ENABLE

TRY THIS API [↗](#)

Getting the client ID and Client secret.

Once the API has been enabled for the project, you will have to get the client ID and client secret.

This is used to get the access and refresh tokens. After enabling the API, go to “**Credentials**” on the left side of the screen. Then click “**Configure consent screen**”.

The screenshot shows the Google Cloud Platform interface for the 'Dialogflow API'. On the left, a sidebar contains navigation links: Overview, Metrics, Quotas, and Credentials (which is highlighted with a red box). The main content area is titled 'Credentials' and includes a '+ CREATE CREDENTIALS' button and a 'DELETE' button. Below this, a section titled 'Credentials compatible with this API' provides a link to 'Credentials in APIs & Services'. A warning message states: 'Remember to configure the OAuth consent screen with information about your application.' To the right of this message is a button labeled 'CONFIGURE CONSENT SCREEN' (highlighted with a red box). Below the warning, there is a section for 'OAuth 2.0 Client IDs' with a table that has columns for Name, Creation date, Type, and Client ID. The table currently shows 'No OAuth clients to display'.

During the configuration, the only thing we need to enter is the Application name. Set this to something you find suitable. Thereafter scroll down and click “**Save**”.

Once you are finished with configuring your OAuth consent screen, go back to “**Credentials**”.

Now click “**+ Create Credentials**” -> “**OAuth client ID**”:

This screenshot shows the 'Create Credentials' dropdown menu on the Google Cloud Platform 'Credentials' page. The dropdown is open, showing several options: 'API key' (described as 'Identifies your project using a simple API key to check quota and access'), 'OAuth client ID' (described as 'Requests user consent so your app can access the user's data'), 'Service account' (described as 'Enables server-to-server, app-level authentication using robot accounts'), and 'Help me choose' (described as 'Asks a few questions to help you decide which type of credential to use'). The 'OAuth client ID' option is highlighted with a red box. In the background, the 'Credentials' page is visible, with the '+ CREATE CREDENTIALS' button also highlighted with a red box.

Application Type = Web application.

Name = Give this credentials a name you find suitable.

Authorized redirect URIs = This one is **important**. Enter:

`https://developers.google.com/oauthplayground`

Once you are done with this, click **“Create”**:

For applications that use the OAuth 2.0 protocol to call Google APIs, you can use an OAuth 2.0 client ID to generate an access token. The token contains a unique identifier. See [Setting up OAuth 2.0](#) for more information.

Application type

- ☒ Web application
- ☐ Android [Learn more](#)
- ☐ Chrome App [Learn more](#)
- ☐ iOS [Learn more](#)
- ☐ Other

Name ?

Bot integration

Restrictions

Enter JavaScript origins, redirect URIs, or both [Learn More](#)

Origins and redirect domains must be added to the list of Authorized Domains in the [OAuth consent settings](#).

Authorized JavaScript origins

For use with requests from a browser. This is the origin URI of the client application. It can't contain a wildcard (`https://*.example.com`) or a path (`https://example.com/subdir`). If you're using a nonstandard port, you must include it in the origin URI.

`https://www.example.com`

Type in the domain and press Enter to add it

Authorized redirect URIs

For use with requests from a web server. This is the path in your application that users are redirected to after they have authenticated with Google. The path will be appended with the authorization code for access. Must have a protocol. Cannot contain URL fragments or relative paths. Cannot be a public IP address.

`https://developers.google.com/oauthplayground`

`https://www.example.com`

Type in the domain and press Enter to add it



Create

Cancel

You will now get your client ID and secret presented on the screen. We will use this to get our access and refresh tokens.

OAuth client created

The client ID and secret can always be accessed from Credentials in APIs & Services

OAuth is limited to 100 [sensitive scope logins](#) until the [OAuth consent screen](#) is published. This may require a verification process that can take several days.

Your Client ID

[illegible]

Your Client Secret



OK

Getting the access and refresh tokens.

Now when we have the client ID and client secret, we can get our access and refresh tokens.

Browse to <https://developers.google.com/oauthplayground/> and check “Use your own OAuth credentials” in the configuration. Soon as this is done, we need to authorize the API. The scope we need to select is Dialogflow API v2 -> <https://www.googleapis.com/auth/dialogflow>.

Mark this one and press “Authorize API’s”.

Proceed and authorize your app.



This app hasn't been verified by Google yet. Only proceed if you know and trust the developer.

Advanced

You will now get your refresh token and access token. We need to enter the refresh token into the script.

- ▶ Step 1 Select & authorize APIs

- ▼ Step 2 Exchange authorization code for tokens

Once you got the Authorization Code from Step 1 click the **Exchange authorization code for tokens** button, you will get a refresh and an access token which is required to access OAuth protected resources.

Authorization code: 4-yarH5L3grV9NwK044AC0WEZemMFCx8LgY8s_15lhgh

Exchange authorization code for tokens

Refresh token: 1:040d334e9-NtUACgYAAACAC20RwF-L-Sigal-SmDakVY

Access token: Refresh access token

☐ Auto-refresh the token before it expires.

The access token will expire in **3592** seconds.

- ▶ Step 3 Configure request to API

Request / Response

```
POST /token HTTP/1.1
Host: oauth2.googleapis.com
Content-length: 322
content-type: application/x-www-form-urlencoded
user-agent: google-oauth-playground
```

```
code=&
&redirect_uri=https%3A%2F%2Fdevelopers
.google.com%2Foauthplayground&client_id=9
&client_
secret=
&scope=&grant_type=authorization_code
```

```
HTTP/1.1 200 OK
Content-length: 425
X-xss-protection: 0
X-content-type-options: nosniff
Transfer-encoding: chunked
Vary: Origin, X-Origin, Referer
Server: scaffolding on HTTPServer2
-content-encoding: gzip
Cache-control: private
Date: Thu, 23 Apr 2020 16:46:51 GMT
```

☒ Wrap Lines

What needs to be done in SuperOffice?

Configuring the integration.

The CRMScripts will need some adjustments before they are ready to go.

For the trigger “After saving chat message”, there are some variables which needs to be filled out. The first part will be for the configuration of the bot. This will control how the bot behaves.

```
3 //Bot configuration
4
5 //What is the name of your bot?
6 String botName;
7
8 //What is the name of your handover intent?
9 String handOverIntent;
10
11 //What should your bot say before handing over the session to an agent?
12 String handOverMsg;
13
14 //What is the name of your done conversation intent?
15 String doneConversationIntent;
16
17 //What should your bots good-bye message be?
18 String goodBye;
19
20 //How much delay should it be before the message is posted(In seconds)?
21 Integer messageDelay;
22
23 //How much delay should it be before the second text response is sent(In seconds)?
24 Integer secondMsgDelay;
25
```

If you don't know what these intents are for, read more here: [3. Types of intents returned](#).

The second part is for the API. Here you will need to enter your Dialogflow details. If you don't know where to find this information, see 4. [What needs to be done on Dialogflows side?](#)

```

////////////////////////
//API configuration //
////////////////////////

//What is your client ID?
String clientId;


//What is your client secret?
String clientSecret;

//What is your refresh token?
String refreshToken;

//What is your projectID?
Stringt projectId;

```

projectl_id is a unique ID for agents in Dialogflow. This decides which agent in Dialogflow you will use as the chatbot. You will find this under the settings for your agent in the Dialogflow console:


[UPDATE AVATAR](#)

DESCRIPTION

Describe your agent

DEFAULT TIME ZONE

(GMT+2:00) Europe/Kaliningrad

Date and time requests are resolved using this timezone.

GOOGLE PROJECT

Project ID	project-id-magfly
Service Account	default@project-id-magfly.iam.gcpusercontent.com

Configuring “Chat changed status” trigger

Next you will need to configure your “Chat changed status” trigger.

Here you will need to enter the following:

```
//In which category(ID) do you want to save the request?  
String categoryId;  
  
//What should the minimum amount of messages be before a request is saved?  
Integer numOfMessages;
```

Finally, “New chat session” trigger.

Here you need to enter the name of your chatbot

```
//What is the name of your bot?  
String botName;
```

Once you are done with these configurations, you are ready to go. The chatbot is now activated.

Example scenarios:

Below are three imaginary scenarios that are good to use for demo purposes.

Scenario #1

Objective:

- Show a scenario where the bot successfully manages to find the solution by itself.
- Show that the conversation is only between bot and customer, no agent interaction. The conversation is logged as a ticket for statistics.
- Also show that the bot can handle spelling mistakes.

Customer:

1. navigates to a customer center page, and clicks the chat widget.
2. Receives a greeting message.
3. Customer types a question:
4. ""I need to add a *nwe category*, could you please help me with that?"".

Bot:

Bot replies to get more information, "What kind of category do you need? Is it for requests in Service, or for person/companies in S&M?"

Customer:

Customer will be able to answer using the 2 buttons populated by the bot. Customer clicks the button for "S&M".

Bot:

Bot replies back with a message, containing a FAQ that explains how to achieve this.

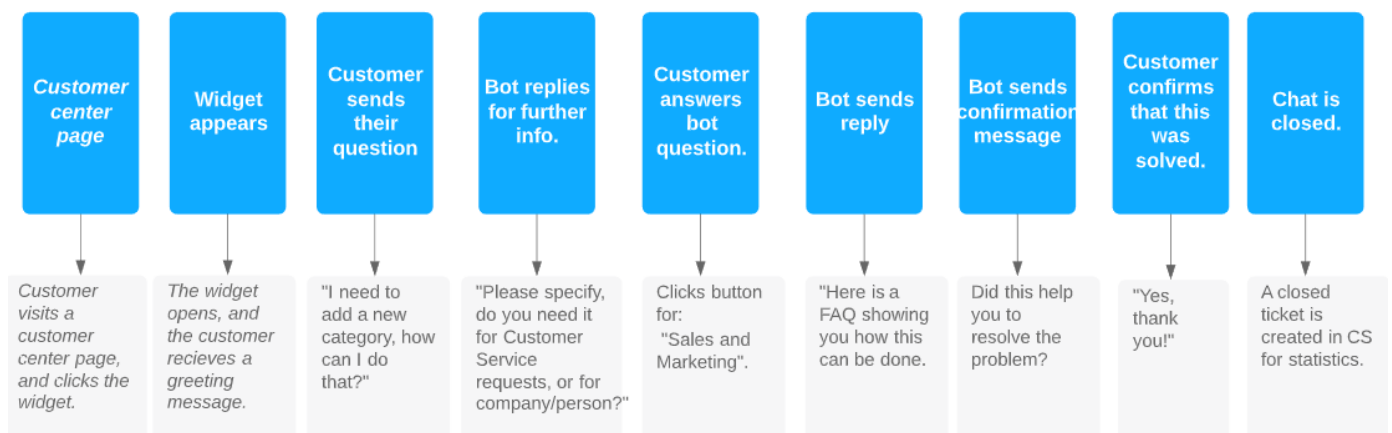
A few seconds later, the bot sends a confirmation message to ask if this was a correct answer.

Customer:

Replies – "Yes, thank you!".

Bot:

The session is closed, and a closed ticket is created in CS for keeping history and statistics.



Scenario #2

Objective:

- Show a scenario where the bot cannot help the customer further. It will then handover the chat conversation to an agent. It will be handed over as a chat session.
- Show how it looks like from SuperOffice side(no chat visible before handover).
- Show how all history of the conversation will be kept for the agent when taking over the chat.

Customer:

1. navigates to a customer center page, and clicks the chat widget.
2. Receives a greeting message.
3. Customer types a question:
4. "I cannot login to SuperOffice".

Bot:

"Is it only you who cannot login, or is it everyone?"

Gives 2 buttons as options – Everyone / Only me.

Customer:

Customer clicks the button "Everyone".

Bot:

Bot informs that this will be transferred to a agent instead for further help.

Presents the prechat-form to identify customer.

Sends the chat session to In-queue.

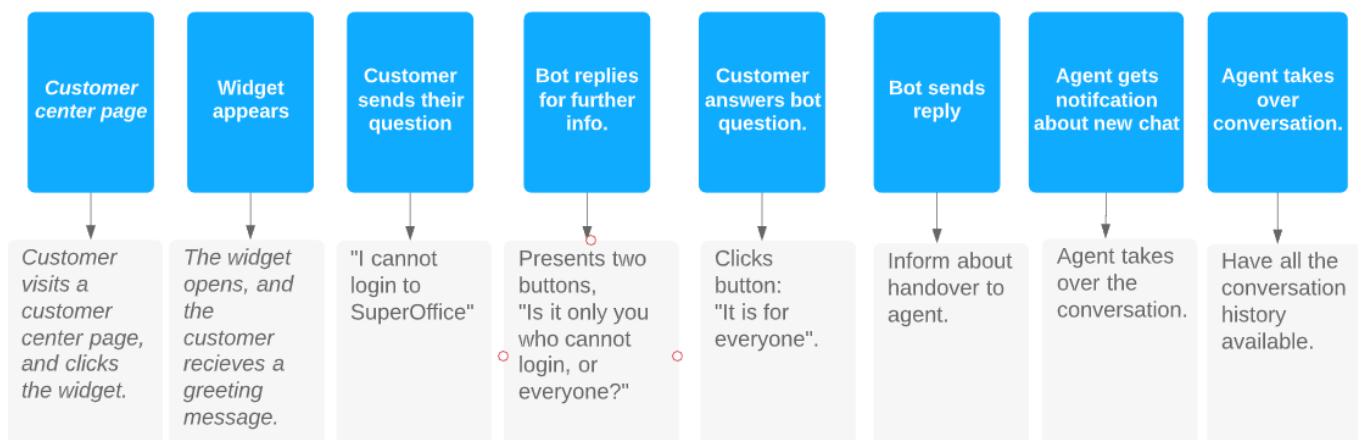
SuperOffice agent:

Gets a notification that a new chat is available.

Clicks "Get next in queue".

Agent gets chat with all of the history,

Bot interaction is now disabled.



Scenario #3

Objective:

Show a scenario where the bot is unable to assist further, and there are no available agents in the chat. An open ticket will then be created, and the request ID will be presented to the customer.

Customer:

1. navigates to a customer center page, and clicks the chat widget.
2. Receives a greeting message.
3. Customer types a question:
4. "How can I get a backup of our database?"

Bot:

Bot replies to get more information, "Are you using SuperOffice Online, or Onsite?"

Presents two buttons for customer to choose from.

Customer:

Customer clicks the button for "Online".

Bot:

Sends FAQ for how to get the online backup. Since the FAQ mentions that they need to contact support, it asks "Do you want to talk with a colleague of mine to get a backup right away?"

Customer:

Answers "Yes please".

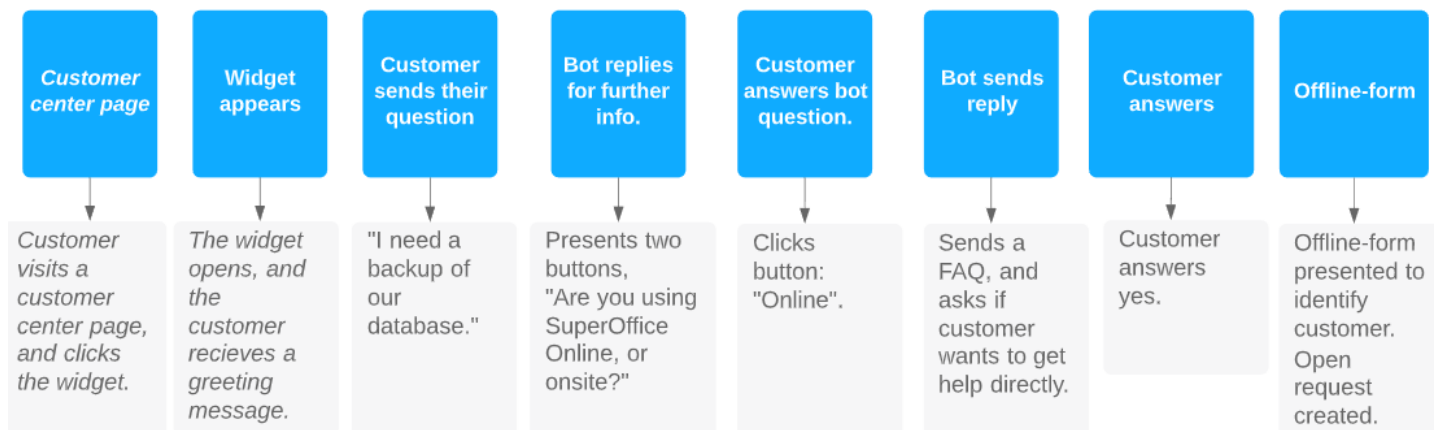
Bot:

-There are no agents available, so bot presents the offline-form to register a request.

Customer:

Fills in the offline-form, a request is created.

Gets the request ID from chat.



Content

URLs

- [SuperOffice Customer Service SDK](#)
- [SuperOffice Community](#)
- [SuperOffice chat features](#)
- [Dialogflow REST API documentation](#)
- [Dialogflow console](#)
- [Dialogflow pricing](#)
- [Google Developers OAuthplayground](#)
- [Google Cloud Platform](#)
- [Google Unverified apps](#)

Full API response

```
{
  "responseId": "6a256a05-3826-4532-8175-11939842e7f0-eec93b43",
  "queryResult": {
    "queryText": "test",
    "parameters": {},
    "allRequiredParamsPresent": true,
    "fulfillmentText": "This works. Is coming form new dialogflow agent.",
    "fulfillmentMessages": [{
      "text": {
        "text": [
          "This works. Is coming form new dialogflow agent."
        ]
      }
    }],
    "intent": {
      "name": "projects/t[REDACTED]/agent/intents/95f7883e-d321-4733-95aa-b78373e817ee",
      "displayName": "Test_Intent"
    },
    "intentDetectionConfidence": 1,
    "languageCode": "en"
  }
}
```