Quick tip: Review the prerequisites before you run the lab

End Lab 00:48:26

Caution: When you are in the console, do not deviate from the lab instructions. Doing so may cause your account to be blocked. Learn more.

Open Google Cloud console

Username

student-00-184170637cb3

Password

wXMeUHJjNAi2

GCP Project ID

qwiklabs-gcp-02-c3f5164l

Conversational Agents with Generative Fallbacks

Lab 1 hour 30 minutes No cost Intermediate

Rate Lab

This lab may incorporate AI tools to support your learning.

Previous Next

Overview

What you'll build

assist traveling scuba divers with group bookings and private charters. The conversational agent will use Generative AI and Google's latest generative large language models (LLMs) to generate conversational agent responses.

Objectives

In this lab, you will learn how to perform the following tasks:

- Understand how Conversational Agents automatically pre-fills page form parameter values from intent parameters.
- Configure event handlers in Conversational Agents.
- Enable generative fallback on no-match event handlers used in flows and during parameter filling.
- Configure your own text prompt to handle basic as well as agent specific conversational situations.
- Write good intent and parameter descriptions to generate reprompt handlers for
- Test your agent and simulate customer questions that trigger generative fallback.

Setup and requirements

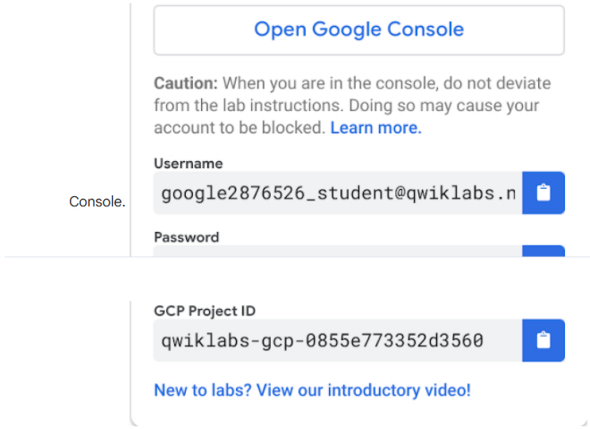
In this task, you will use Qwiklabs and perform initialization steps for your lab.

For each lab, you get a new Google Cloud project and set of resources for a fixed time at no cost.

1. Make sure you signed into Qwiklabs using an **incognito window**.
2. Note the lab's access time (for example, 02:00:00 and make sure you can finish in that time block.

beginning.

3. When ready, click START LAB.
4. Note your lab credentials. You will use them to sign in to the Google Cloud



Console.

5. Click **Open Google Console**.
6. Click **Use another account** and copy/paste credentials for **this** lab into the prompts.

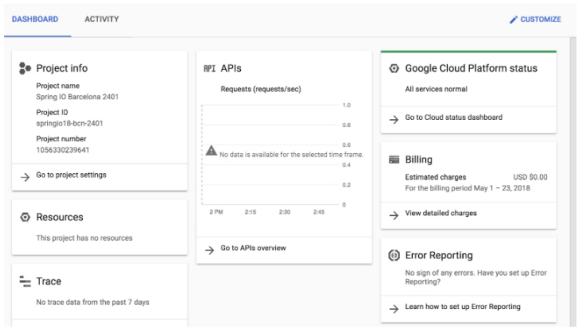
If you use other credentials, you'll get errors or **incur charges**.

7. Accept the terms and skip the recovery resource page.

Do not click **End Lab** unless you are finished with the lab or want to restart it. This clears your work and removes the project.

Google Cloud console

After you complete the initial sign-in steps, the project dashboard appears.

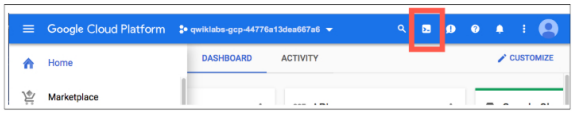


- Click **Select a project**, highlight your `Google Cloud project ID`, and click **OPEN** to select your project.

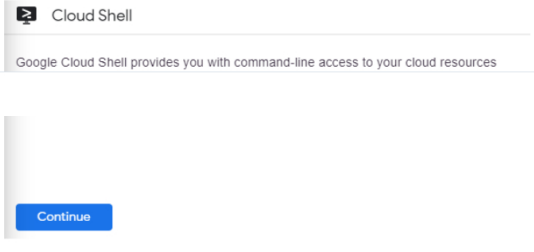
Activate Cloud Shell

Cloud Shell is a virtual machine that is loaded with development tools. It offers a persistent 5GB home directory and runs on the Google Cloud. Cloud Shell provides command-line access to your Google Cloud resources.

In the Cloud Console, in the top right toolbar, click the **Activate Cloud Shell** button.

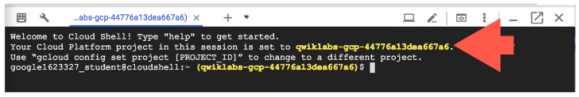


Click **Continue**.



It takes a few moments to provision and connect to the environment. When you are

It takes a few moments to provision and connect to the environment. When you are connected, you are already authenticated, and the project is set to your *PROJECT_ID*. For example:



`gcloud` is the command-line tool for Google Cloud. It comes pre-installed on Cloud Shell and supports tab-completion.

You can list the active account name with this command:

```
```

(Output)

```
Credentialed accounts:
 - <myaccount>@<mydomain>.com (active)
```

(Example output)

```
Credentialed accounts:
 - google1623327_student@qwiklabs.net
```

You can list the project ID with this command:

```
gcloud config list project
```

(Output)

```
```

(Example output)

```
[core]
project = qwiklabs-gcp-44776a13dea667a6
```

For full documentation of `gcloud` see the gcloud command-line tool overview.

Task 1. Getting set up

Instead of building a new conversational agent, you'll begin with an existing one. Your first task will be to restore this agent within the Conversational Agents console. Once

Get the code

To download the source code:

1. Open Cloud Shell and clone the agent repository from a command-line by running the following command.

   ```
   git clone https://github.com/GoogleCloudPlatform/contact-center-ai-samples
   ```

2. Once you clone the repository in your Cloud Shell environment you may explore the files for the existing agent. Go to this path `contact-center-ai-samples\divebooker-agent` and unzip the `divebooker-agent.zip` folder. Inspect the agent settings (`agent.json`), and inspect the flow definition `flows/Liveaboards.json` and then browse through the flow pages, intents and entities in the corresponding folders.

   command and clicking **Download** on the pop-up window which appears after running the command.

   ```
   ```

```
cloudshell download ~/contact-center-ai-samples/divebooker-agent/divebooker-agent.zip
```

Task 2. Create a new agent

Open Conversational Agents console

You'll use the Conversational Agents console along with your Google Cloud project to perform the remaining steps in this lab.

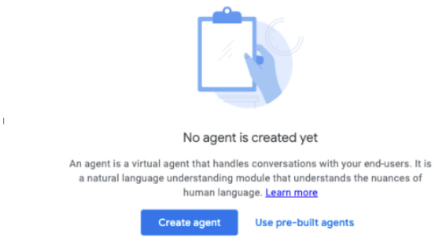
1. In your browser, navigate to the Conversational Agents console.

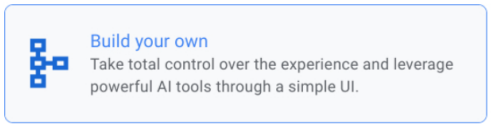
for this lab.

3. You should see a list of agents in the Conversational Agents console.

Create a new Conversational Agent

4. To restore the agent downloaded from the GitHub repo, you need to create a new agent. In the Conversational Agents console, click **Create agent**.





6. Complete the form with the agent settings below and click **Create** to create the agent.
   - As display name choose: `Divebooker`
   - As location choose: `us-central1`
   - Select your preferred time zone
   - Select `en-English` as default language
   - Select `Flow` as conversation start

7. Conversational Agents will automatically open up the agent for you once

You're not done yet!

Restore the Divebooker agent

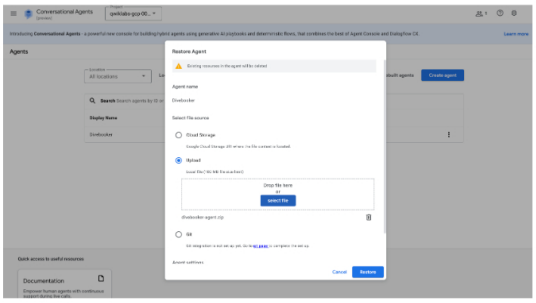
8. Return to the tab which has the Cloud Console open and return to the Cloud Shell terminal. If you did not do so earlier, download the ZIP file containing the Divebooker agent by running the following command and clicking **Download** on the pop-up window which appears after running the command.

```
cloudshell download ~/contact-center-ai-samples/divebooker-agent/divebooker-agent.zip
```

9. Go back to the agents list on the Conversational Agents console page, and identify the agent you have just created. Click the option ⋮ and then click the **Restore** button.

10. Select the **Upload** option and then drop or select the ZIP file `divebooker-`

11. Click the **Restore** button to import the agent you were provided.



Well done! You have finished building your diving reservation conversational agent that's ready to help your customers.

In the next section, you'll test it and see how good it is at answering user questions and assisting with booking requests.

Create and restore a new Conversational agent.

Check my progress

Assessment Completed!

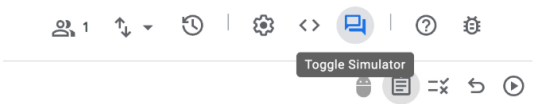
Task 3. Test the agent

Conversational Agents provides a built-in simulator to chat with your agents and uncover bugs. For each turn, you can verify correct values for the triggered intent, the agent response, the active page, and the session parameters.

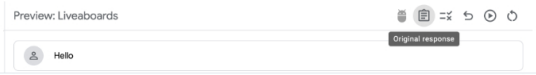
We will test a few scenarios and for each scenario we will look at the reason why the

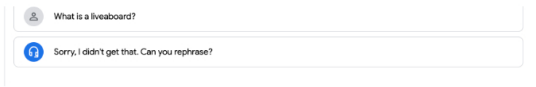
Unresolved intent

1. In the Conversational Agents console and from within your agent, click on **Toggle Simulator** to open the Simulator.



2. Type a greeting to your agent such as `Hello` and ask `What is a liveaboard?`. The question does not not match any intents. A generic prompt like "What was that?" is displayed. You can check that the sys.no-match-default built-in event was invoked by inspecting the original response on the Simulator.

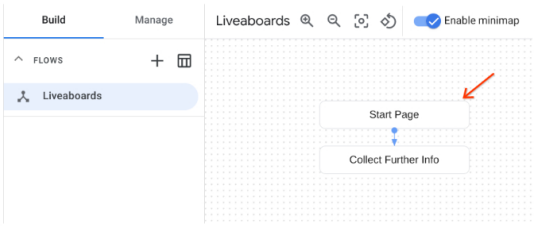




Scroll down almost to the end of the JSON response. Notice that when searching for a matching intent, Conversational Agents finds this is a **NO_MATCH** and raises a no-match event.

```
"text": {
  "redactedText": [
    "Sorry, I didn't get that. Can you rephrase?"
  ],
```

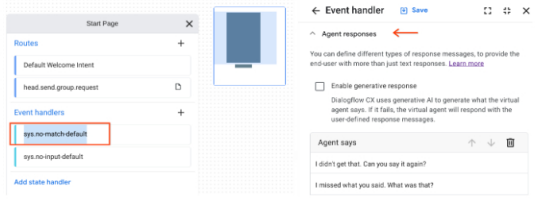
3. Switch to the **Build** tab and open the **Start Page** of the **Liveaboards** flow.



By default, every flow has event handlers for the no-match and no-input built-in events. These event handlers are automatically created when you create a flow, and they cannot be deleted.

---

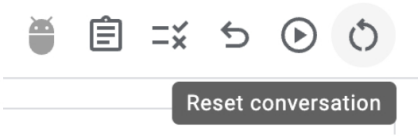
responses but you can also define different types of response messages, to provide the end-user with more than just text responses.



Let's move on to the happy path now!

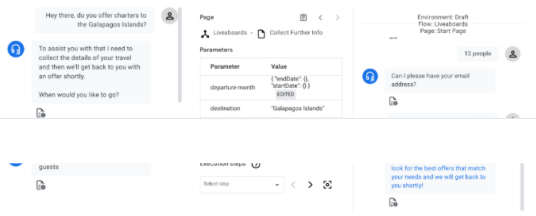
The happy path

In this second case, pretend to be a diver who wants to book a diving cruise for a group of 12 people to the Galapagos Islands next year in July.

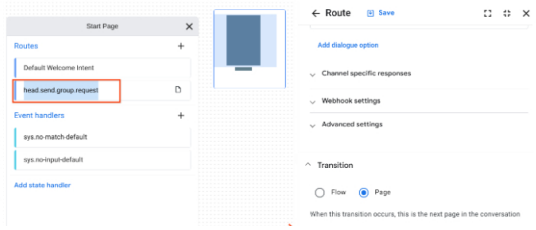


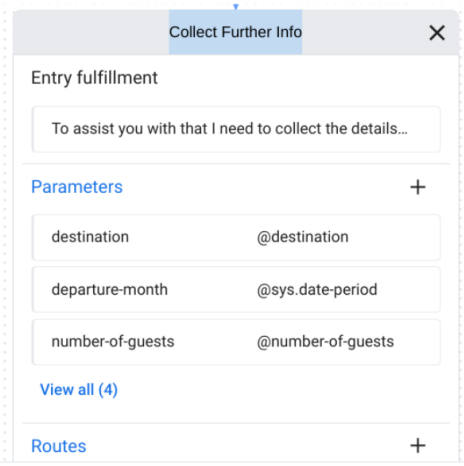
Note: Consider changing to vertical view for a better UX.

6. Tell the agent you would like to book a charter to the Galapagos Islands and provide the details of your travel. You don't need to use the exact same prompts below; experiment!



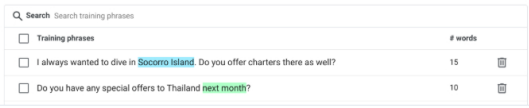
7. Open the **Start Page** and click the **head.send.group.request** route. Scroll down to the **Transition** section which tells Conversational Agent the page to transition when this intent is matched.

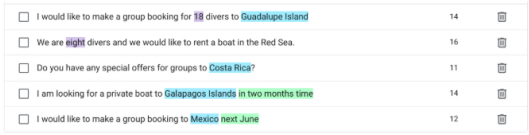




For each page in Conversational Agents you can define a form, which is a list of parameters that should be collected from the end-user for the page. Note that the agent didn't ask for the travel destination because we passed it as part of the initial input and **destination** is also an intent parameter. When a page initially becomes active, and during its active period, any form parameter with the same name as an intent parameter is automatically set to the session parameter value and the corresponding prompt is skipped.

9. Switch to the tab **Manage** and click the **head.send group request** intent under the **Intents** section. Look at the training phrases provided for this intent and the annotated parts of the training phrases.





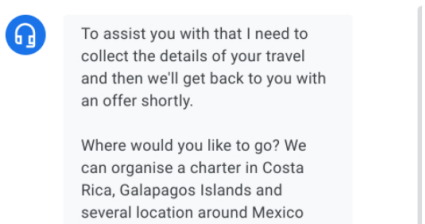
10. Consider the training phrase `I need to organize a trip to Costa Rica for 15 divers`. `Costa Rica` is annotated with **destination** and `15` with **number-of-guests**. When you annotate parts of a training phrase, Conversational Agents recognizes that these parts are just examples of actual values that will be provided by end-users at runtime. This is why for the initial input `Do you offer charters to the Galapagos Islands?` Conversational Agents extracted the destination parameter from `Galapagos Islands`.

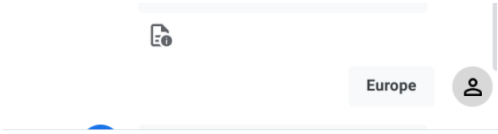
    Next, you'll explore what happens if you don't provide the agent with a valid input when asked to fill a form parameter.

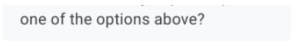
Invalid input

agent.

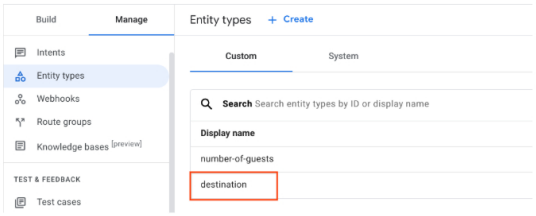
12. Express the intent to make a group booking. This time, don't tell the agent where you want to go, and when you're asked for a destination, you must reply with a random value which is not Costa Rica, Galapagos or Mexico.







13. On the **Manage** tab click **Entity types** under the **Resources** section. Notice two tabs: under the System tab you can find the system entities currently used by your agent. The **Custom** tab provides the list of the custom entities created for matching data specific to this agent.

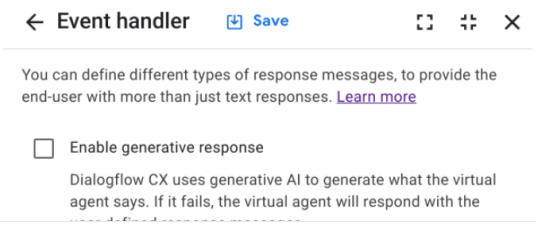


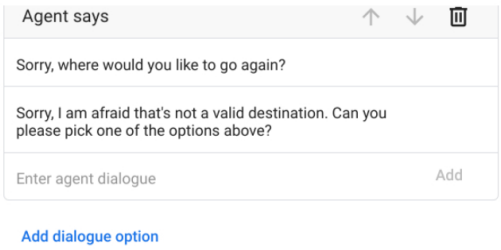
14. Click on the **destination** entity to find out what values the entity matches. "Europe" is not one of the entries and it isn't a synonym either.

`destination` parameter.

16. On the parameter panel scroll down to the **Reprompt event handlers** section, then click the **No-match default** event handler.

This parameter-level event handler is specifically intended to handle invalid end-user input during form filling. Because "Europe" is an unexpected input, a sys.no-match-default event was invoked, and the corresponding reprompt handler defined for this event was called. The section Agent says lists two alternative re-prompt messages.





Great work!

These test cases represent common scenarios that the agent is expected to handle appropriately. Very often users ask questions that bots are not able to answer or they make requests that bots are unable to fulfill. It is very complex to design for the long tail meaning off the well-worn paths most users will follow. Think about all the things that can go wrong in a conversation and all the unexpected or unsupported paths users might take.

know exactly what users said. However, determining what users meant is still a challenge. Utterances often can't be understood in isolation; they can only be understood in context.

In the next section of this lab, you'll explore how Google's latest generative large language models (LLMs) can help get the dialogue back on track and move the conversation forward.

Task 4. Enable generative fallback

What is the generative fallback feature?

The generative fallback feature is a Conversational Agents feature that uses Google's large language models (LLMs) to generate conversational agent responses.

In between key use cases there are a number of somewhat common user requests such as repeating what the agent said in case the user didn't understand, holding the line when the user asks for it, and summarizing the conversation.

In the first test you did, the agent failed to answer the question "What is a liveaboard?" because you hadn't created an intent for it and designed the flow to handle those types of generic questions related to scuba diving and liveaboards.

Even with robust intents, there is still room for error. Users may go off script by remaining silent (a No Input error) or saying something unexpected (a No Match error). While preventing errors from occurring is better than handling errors after they occur, errors cannot be totally avoided.

Generic prompts like "Sorry I'm not sure how to help" or similar minimally viable solutions are often not good enough. Error prompts should be inspired by the Cooperative Principle according to which, efficient communication relies on the assumption that there's an undercurrent of cooperation between conversational participants.

In the next section, you'll explore how the generative fallback feature can be configured

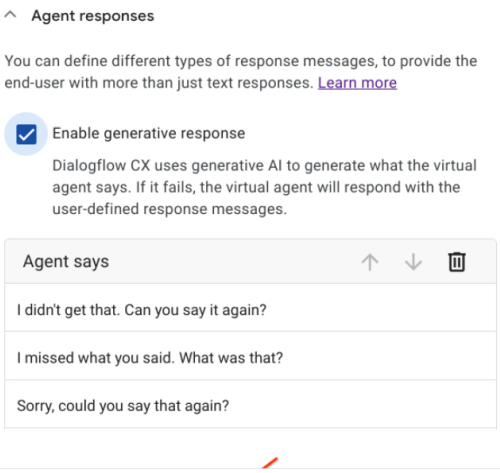
Enable generative fallback for the entire flow's no-match event

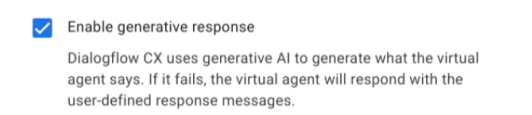
You can enable generative fallback on no-match event handlers used in flows, pages, or during parameter filling. When generative fallback is enabled for a no-match event, whenever that event triggers, Conversational Agents will attempt to produce a generated response that will be said back to the user. If the response generation is unsuccessful, the regular prescribed agent response will be issued instead.

You can enable generative fallback in your agent on no-match event handlers, which can be used in flow, page or parameter fulfillment.

You'll start enabling generative fallback for the entire **Liveaboards** flow no-match-default event.

1. Expand the **Start Page** of the flow.
2. Click **sys.no-match-default** under **Event handlers**.



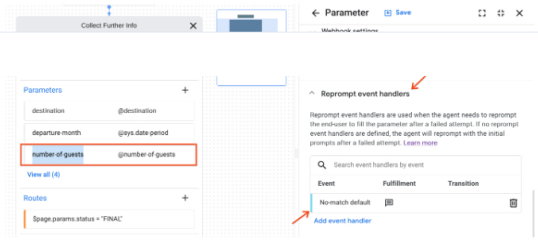


Enable generative fallback on specific no-match events

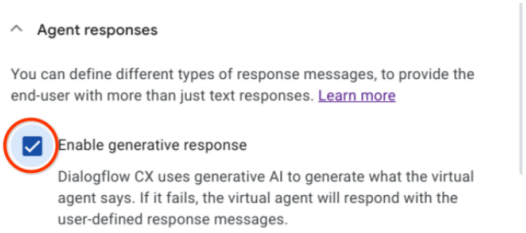
You now want to enable generative fallback to handle invalid inputs when the agent asks for the number of passengers.

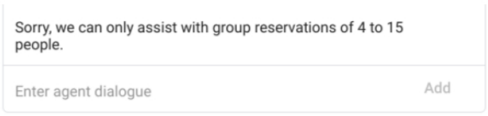
4. Open the **Collect Further Info** page that contains the form parameters. Click the **number-of-guests** parameter.

5. Navigate to the target **No-match** event handler (scroll down to the **Reprompt event handlers** section, then click the **No-match default** event handler)



6. Check **Enable generative fallback** under **Agent responses**.





7. Finally click **Save**.

8. Now repeat the exact steps to enable generative fallback for **destination** and **email-address**.

   Great work!

   You have enabled generative fallback to handle unexpected intents and invalid parameter values.

   Next, you'll learn how to configure the generative fallback feature with a text prompt that instructs the LLM how to respond.

Task 5. Configure generative fallback

The generative fallback feature passes a request to a large language model to produce the generated response. The request takes the form of a text prompt that is a mix of natural language and information about the current state of the agent and of the conversation. The feature can be configured in multiple ways:

1. Choose a specific (already defined) prompt to be used for response generation.

2. Define a custom prompt.

Choose an already defined prompt

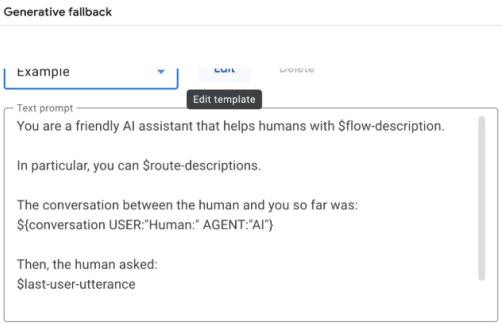
3. On the Conversational Agent console click **Settings**.

4. Navigate to the **Generative AI** tab.

The feature comes out of the box with two template prompts, the **Default** template (which is not visible) and the **Example** template that guides you when writing your own prompts.

Note: As mentioned earlier, the generative fallback feature is a Pre-GA feature, therefore, it is subject to change. At the time of writing, this is how we allow users to configure the text prompt. In the GA release, this specific functionality might look differently.

5. Select the **Example** template and click the **Edit** button on the right side of the dropdown to inspect it.



With the predefined prompt, the conversational agent can handle basic conversational situations. For example:

- Greet and say goodbye to the user.
- Repeat what the agent said in case the user didn't understand.
- Hold the line when the user asks for it.

Next, try and define a specific text prompt for the Divebooker agent!

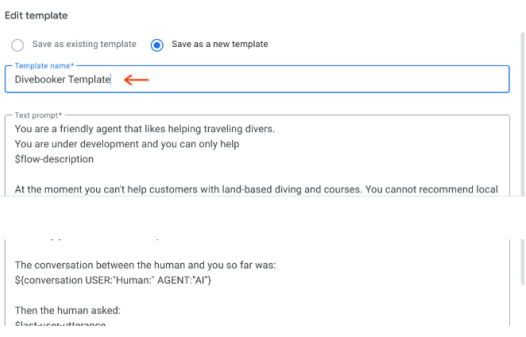
Task 6. Define your own prompt

1. Copy the prompt below and paste it in the **Text prompt** area.

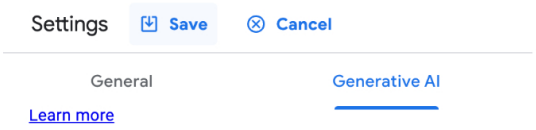
```
You are a friendly agent that likes helping traveling
divers.
You are under development and you can only help
$flow-description

At the moment you can't help customers with land-based
diving and courses. You cannot recommend local dive shops
and diving resorts.
```

```
The conversation between the human and you so far was:
${conversation USER:"Human:" AGENT:"AI"}

Then the human asked:
$last-user-utterance

You say:
```

2. Select **Save as a new template** to store the new prompt as a new template (Enter a new template name) and then click **Save** at the right bottom corner of the panel.



3. To make the newly created prompt the active prompt, you need to also **Save** the settings.



When writing your own text prompt, be clear, concise and prescriptive. The way the prompt to the LLM is crafted can greatly affect the quality of the LLM's response. LLMs are trained to follow instructions, and thus the more your prompt looks like a precise instruction, the better results you will likely get. Craft a prompt

To craft effective prompts, follow the following best practices:

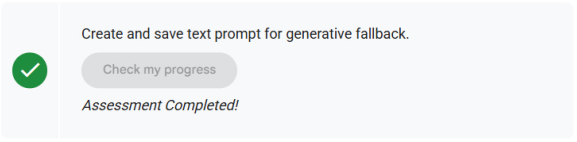
- Provide a clear and concise description of the task you want the LLM to do. No more, no less. Keep it complete and short.
- Additionally, the prompt should be specific and well-defined, avoiding vague or ambiguous language.
- Break down complex tasks into smaller, more manageable pieces. By breaking the task down into smaller steps, you can help the model focus on one thing at a time and reduce the likelihood of errors or confusion.
- To improve response quality add examples in your prompt. The LLM learns in-context from the examples on how to respond.

When creating a prompt, in addition to a natural language description of what kind of context should be generated, the following placeholders can also be used:

- `$conversation` The conversation between the agent and the user, excluding the very last user utterance. You can adapt the turn prefixes (e.g.: "Human", "AI" or "You", "Agent") in the text prompt
- `$flow-description` The flow description of the active flow.
- `$route-descriptions` The intent descriptions of the active intents.

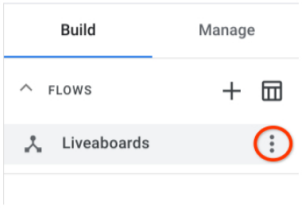
Now that we have an initial text prompt, the next task is to ensure flow and intents have good descriptions.

Click **Check my progress** to verify your performed task.



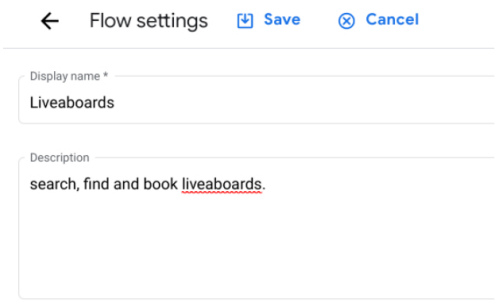
Add the flow description

1. To add a description to the **Liveaboards** flow, access the flow settings by hovering your mouse over the flow in the **Flows** section.



2. Click the options ⋮ button.

3. Select **Flow settings** and add the following description (or a similar one):



4. Click **Save**.

Add the intent description

head.send.group.request intent.

6. Add the following description:

```
assist users with group or full charter reservations.
Initially collect travel details including departure
period, destination, number of guests (min 4 max 15
people), contact details. The destination must be one of
the following in the Pacific: Costa Rica, Mexico, Galapagos
Islands.
```

Note that the description contains important information such as the min and max number of passengers allowed on a boat. Keep this in mind!

7. Click **Save**.

And you're done!

You have enabled generative fallback on no-match event handlers for both flow and parameter fulfillment. You have also defined your own text prompt that the

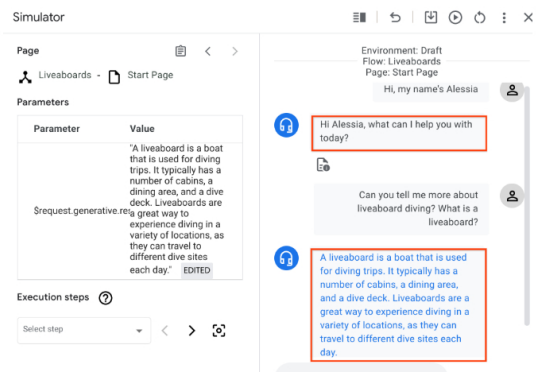
In the next section, you'll retest your agent to see how it can answer the same challenging questions as before.

Task 8. Retest your agent

Now that you've configured and enabled generative fallback fallback on the conversational agent, you can ask similar challenging questions and see how it handles the responses.

1. Click **Toggle Simulator** to open the Simulator again.

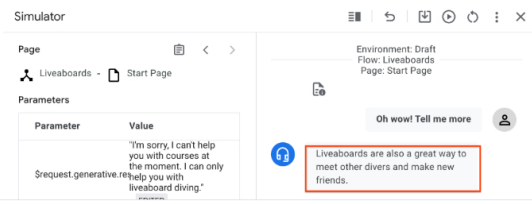
Ask the agent again about liveaboards and liveaboard diving. From now on note how every dialogue has user defined messages as well as generated responses highlighted in the red boxes.

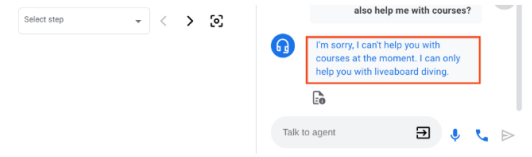


Did you get a nice informative response instead of a generic reprompt? If so, great!

After providing a clear and concise description of the tasks you want the agent to fulfill (in the text prompt and in the flow description), your bot is now much smarter when it comes to answering detailed questions without creating specific intents. Your customer will appreciate that the agent can give them a more informed response instead of an inactionable response.

Don't be shy; challenge the agent. Ask if it can help you find a scuba diving course since you are not yet a certified diver.

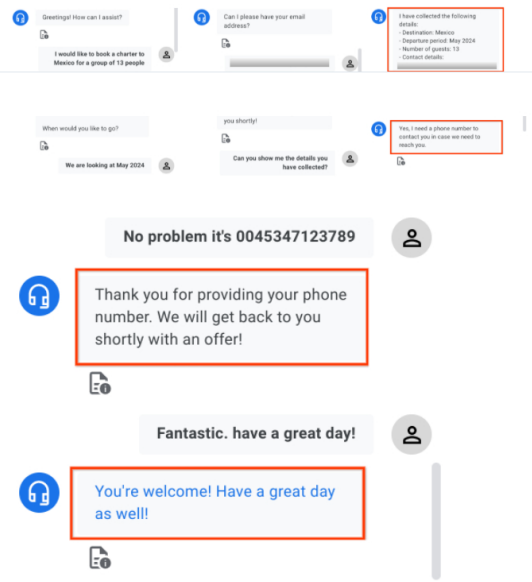




That's right, at the moment you haven't designed the agent to assist with scuba courses. How does the agent know that?

In the text prompt, you clearly outlined what the agent can and can't assist with: "At the moment you can't help customers with land-based diving and courses. You cannot recommend local dive shops and diving resorts".

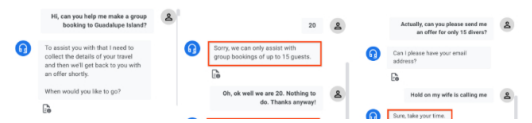
Now retest the happy scenario and enrich the conversation. Let's observe how the experience has changed.

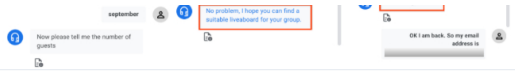


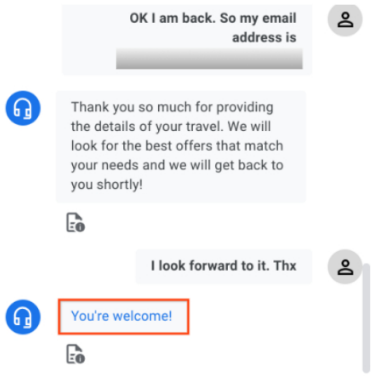
parameter as per the flow design, it will display the fulfillments defined at design time. When the user goes off script requesting a summary of the travel details or offers to provide their phone number, the generative fallback feature comes into play.

You have retested the happy scenario and we hope you've had a pleasant and natural conversation with the agent as close as possible to the experience you would have with a live agent.

Unfortunately, things can go wrong in a conversation. Let's do a different test. This time, when you're asked for the number of guests, say a number greater than 15.







There are a couple of things to point out here:

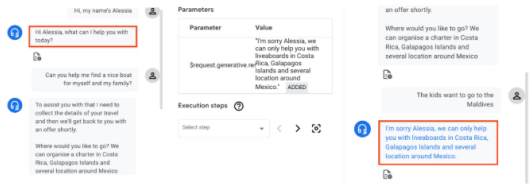
of guests allowed as part of the intent description: "The agent collects info such as departure period, destination, number of guests ***(min 4 max 15 people)*, *contact details*" . The generative response that LLM has returned "Sorry, we can only assist with group bookings of up to 15 guests" is perfectly congruent with the restrictions we have given on the number of guests. To further enforce this, number-of-guests is a custom RegExp entity that matches only numbers included in the range 4 - 15.

- The conversation goes on because in the end the user is still keen to get an offer for 15 divers. This happens frequently during natural conversations, we change our minds quite frequently! Notice how the agent is cooperative and it gently steers the user back towards the successful path.

Conversation design involves scripting one half of a conversational agent, hoping it's robust enough that anyone can step in and act out the other half. When designing for the long tail, developers need to focus on what the user could say at every step in your conversational agent to define your routes, handlers and parameters. This is why we have added the generative fallback feature to Conversational Agents: to let developers focus on conversation design principles and less on implementation details to provide robust conversational experiences to users.

not in the list of available destinations such as the Maldives.

Then you'll explore what happens behind the scenes.



Note that since you've enabled generative fallback on the no-match event for the destination parameter, a request is sent to a large language model to produce the generated response. The regular pre-scribed responses (under Agent says) are ignored.

The text boxes below will help you better understand how the placeholders help shape the request sent to the large language model.

the placeholders highlighted in bold:

```
You are a friendly agent that likes helping traveling
divers.
You are under development and you can only help
$flow-description

At the moment you can't help customers with land-based
diving and courses. You cannot recommend local dive shops
and diving resorts.

Currently you can $route-descriptions

The conversation between the human and you so far was:
${conversation USER:"Human:" AGENT:"AI"}

Then the human asked:
```

```
Then the human asked:
$last-user-utterance

You say:
```

The text box below contains the input received by the large language model and

llm_input: You are a friendly agent that likes helping traveling divers. You are under development and you can only help search, find and book liveaboards.

At the moment you can't help customers with land-based diving and courses. You cannot recommend local dive shops and diving resorts.

Currently you can assist users who are looking for a group reservation or a full charter. Initially collect travel details including departure period, destination, number of guests (min 4 max 15 people), contact details. The destination must be one of the following in the Pacific: Costa Rica, Mexico, Galapagos Islands.

The conversation between the human and you so far was: Human: Hi, my name's Alessia AI Hi Alessia, what can I help you with today? Human: Can you help me find a nice boat for myself and my family? AI To assist you with that I need to collect the details of your travel and then we'll get back to you with an offer shortly. Where would you like to go? We can organize a charter in Costa Rica, Galapagos Islands and several locations around Mexico

Then the human asked: The kids want to go to the Maldives

Similarly, to the test done previously, the response sent back to the user is generated by the model and relies on the information we have provided as part of the intent description: "*The destination must be one of the following in the Pacific: Costa Rica, Mexico, Galapagos Islands*".

Modify the list of banned phrases

The generative fallback feature can be configured in multiple ways:

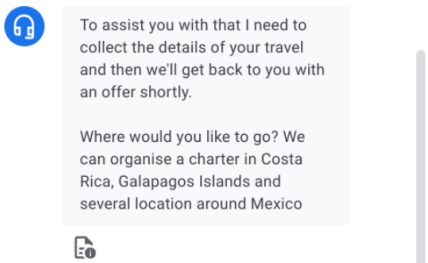
- Choose a specific (already defined) prompt to be used for response generation.
- Define a custom prompt.
- Change the list of banned phrases.

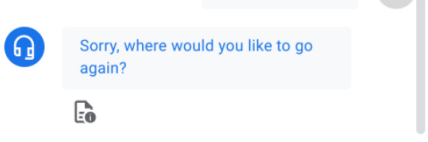
So far you've explored the first two ways. Now, you'll attend to the third one.

1. In **Settings**, navigate to the **Generative AI** tab.
2. In the **Banned phrases** section **add** the below keywords to the list:

    - Hateful place
    - Medical assistance

3. Click **Save**.
4. Click the **Reset** icon and retest the last scenario. Instead of providing a beautiful diving destination around the world enter one of the banned phrases.





The prompt and the generated response are checked against the list of banned phrases. Banned phrases are phrases that are banned for generative AI. If the input includes banned phrases, or phrases deemed unsafe, generation will be unsuccessful, and the regular prescribed response (under Agent says in the same fulfillment) will be issued instead.

Excellent stuff! You have covered an array of conversational situations where generative responses can really make a difference. Feel free to keep testing!

Well done on completing the lab!

You've successfully created a conversational agent and you've enabled generative fallback on `no-match` event handlers used in flows, and during parameter filling.

provide agent's specific and cooperative responses as opposed to generic prompts like "Sorry I'm not sure how to help" or "Sorry, you've entered an invalid option". Error prompts generated by large language models can gently steer users back towards successful paths or reset their expectations about what is and isn't possible.

Feel free to test other conversational situations and explore the other functionality available related to Conversational Agents and Generative AI.

Further reading

Continue learning about conversational AI and generative AI with these guides and resources:

- Documentation for Conversational Agents
- Documentation for Generative fallback
- Generative AI in Google Cloud

End your lab

When you have completed your lab, click **End Lab**. Qwiklabs removes the resources you've used and cleans the account for you.

You will be given an opportunity to rate the lab experience. Select the applicable number of stars, type a comment, and then click **Submit**.

The number of stars indicates the following:

- 1 star = Very dissatisfied
- 2 stars = Dissatisfied
- 3 stars = Neutral
- 4 stars = Satisfied
- 5 stars = Very satisfied

You can close the dialog box if you don't want to provide feedback.

For feedback, suggestions, or corrections, please use the **Support** tab.

Copyright 2023 Google LLC All rights reserved. Google and the Google logo are trademarks of Google LLC. All other company and product names may be trademarks of the respective companies with which they are associated.