Building User Interfaces

Dialogflow 2

Intermediate Concepts

Professor Yuhang Zhao

Announcements

- **This week:** Midterm 2
- **Next week:**
  - Tuesday: last in-person class!
  - Thursday: bonus lecture (video lecture)
- **End of semester:**
  - Last assignment - Dialogflow
  - Due on May 6
  - We will offer office hours and discussion groups until May 6

What we will learn today?

— Introduction to Dialogflow, Recap
— Dialogflow Building Blocks, Part 2
— Fulfillment

Introduction to Dialogflow, Recap

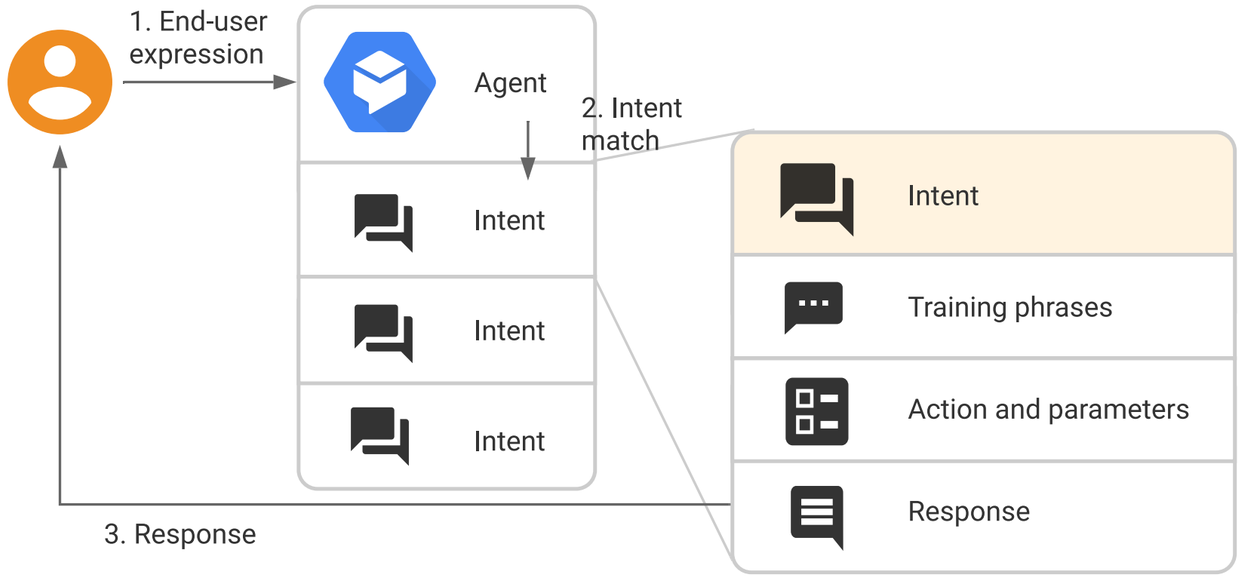
What is Dialogflow?

Dialogflow is an end-to-end, build-once deploy-everywhere development suite for conversational interfaces for websites, mobile applications, and IoT devices (e.g., smart speakers).

How does Dialogflow work?[8]

The process within Dialogflow involves:

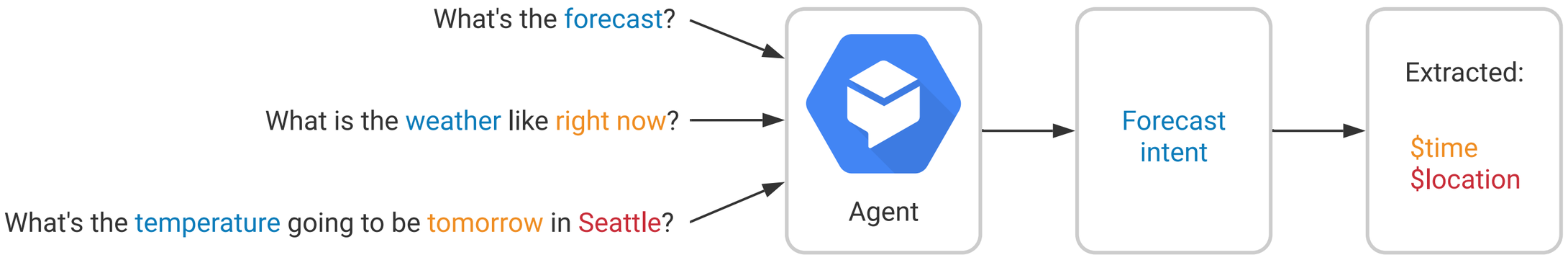
1. User expression
2. Intent matching
3. System response



[8] Image source

What is an *agent*?

Definition: A Dialogflow agent is a virtual agent that handles conversations with users (similar to a human call agent).[9]

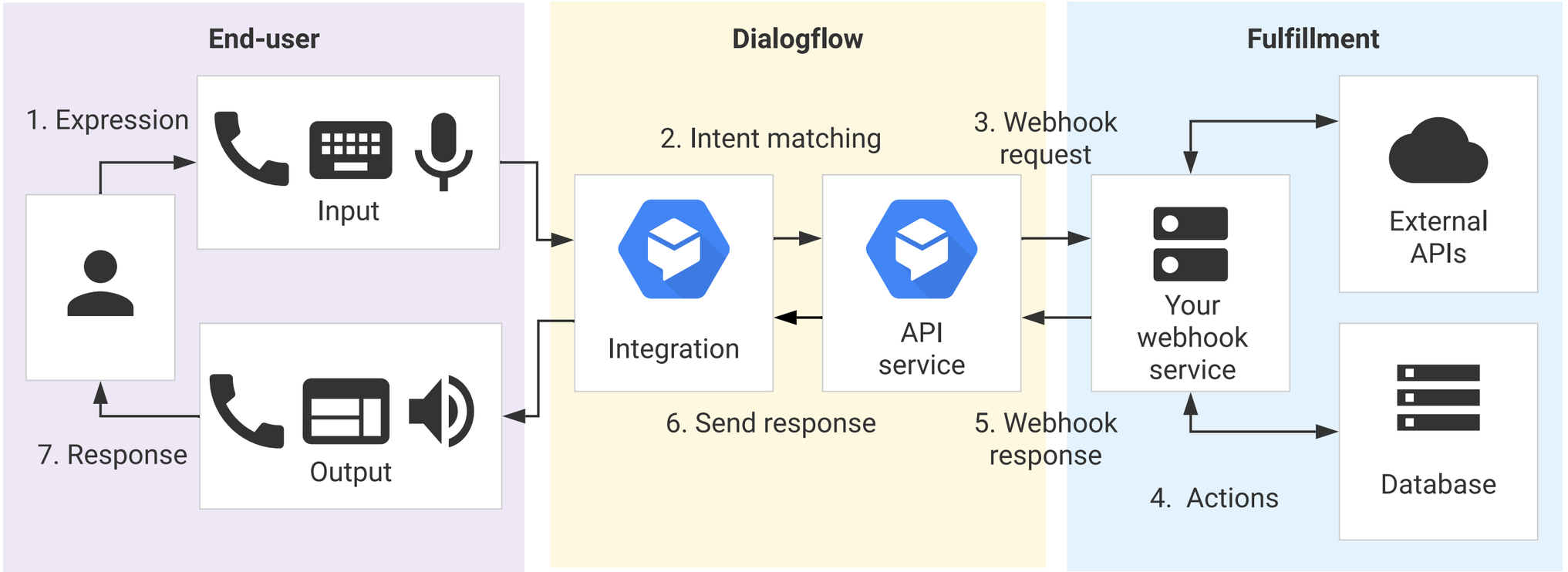


[9] Image source

Agents are high-level containers for a number of building blocks:

— Agent settings
— Intents
— Entities
— Knowledge
— Integrations
— Fulfillment

The End-to-end Dialogflow Workflow[10]



[10] Image source

Dialogflow Building Blocks, Part 2

Contexts

What are *contexts*?

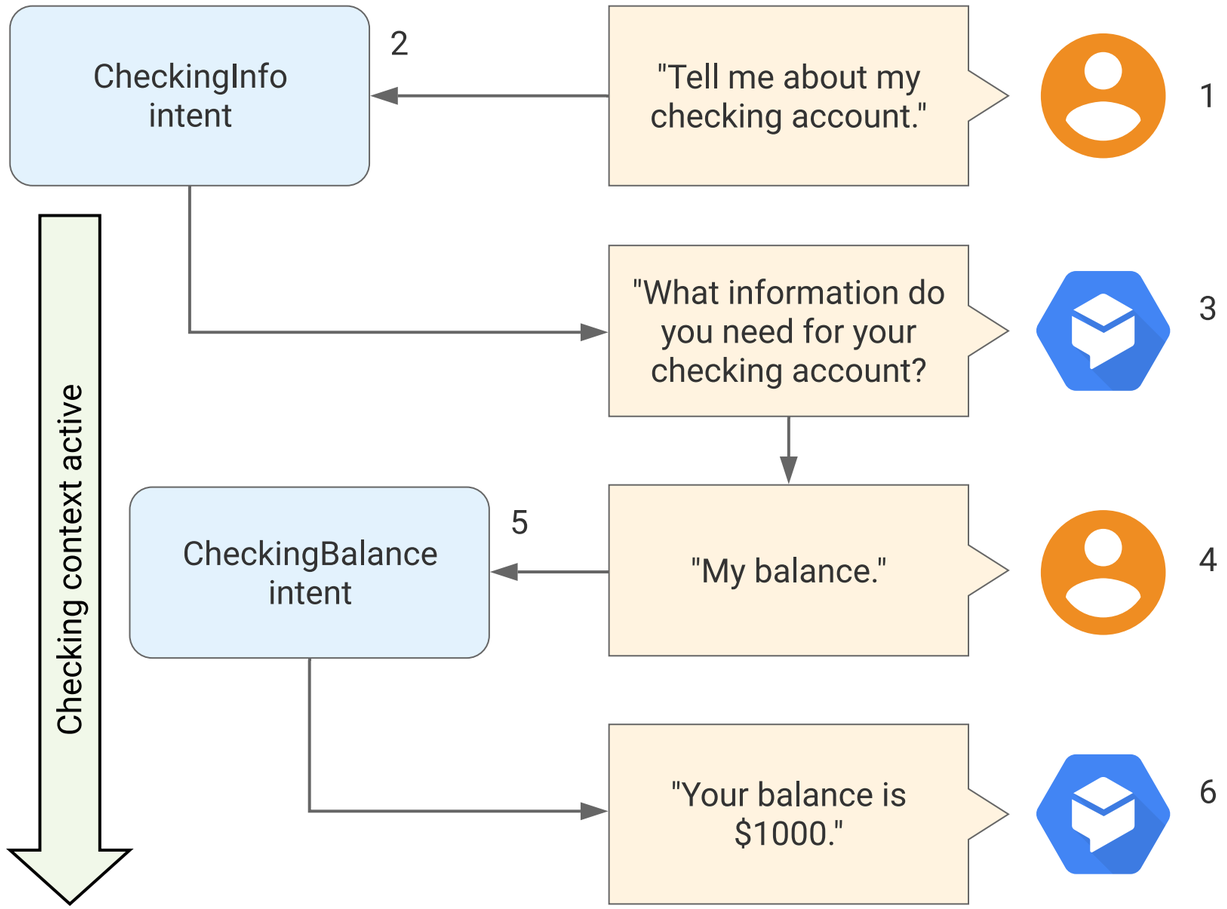
Definition: Context is the *common ground* that communicators have about the conversation that allows them to infer ambiguous speech, such as "pass me that."

In Dialogflow, contexts provide the agent with common ground on the *intent* and respond to ambiguous speech.

Contexts allow us to control the order of intent matching and define different behaviors for intents with the same training phrases.

Let's Look at An Example[5]

A `checking` context is defined for the `CheckingInfo` intent and is active, so that the agent is more likely to recognize the `CheckingBalance` intent.



[5] Image source

Defining Contexts

Contexts for intents are defined as *input context* and *output context*.

Context Example

A: Welcome to WiscShop!
U: Hello!
A: What are you looking for?
U: Hats.

Agent activates the `looking-for-hats` output context.

A: Great. How can I help?
U: Show me what's on sale.

Agent matches `ShowSaleItems`, input context `looking-for-hats`.

Input Context

Input context are added to intents, and they increase the likelihood of an intent being matched when the context is active.

Output Context

Output control *active* contexts. When an intent is matched, any configured output contexts for that intent become active.

Multiple output contexts can be applied to an intent for finer intent matching.

The lifespan of an output context determines how long the context will be active for.

Context Lifespan

Contexts are designed to expire after some time.

— By default, they expire after 5 requests or 20 minutes once they are activated.
— The lifespan can be specified in terms of number of turns.

Follow-up Intents

Follow-up intents are children to parent intents and automatically set to inherit the context from the parent intent.

Intent name	Training phrase	Input context	Output context	Intent response
Appointment	Hello		appointment-followup	Would you like to make an appointment?
↳ Appointment - yes	Yes	appointment-followup	appointment-yes-followup	Would you like a haircut?
↳↳ Haircut - yes	Yes	appointment-yes-followup		Your appointment is set.
↳↳ Haircut - no	No	appointment-yes-followup		Goodbye.
↳ Appointment - no	No	appointment-followup		Goodbye.

Events

What Are Events?

Definition: Events trigger intents based on external events rather than user queries.

We can use *platform events* or create our *custom events*.

Platform Events

Definition: Events that are triggered by actions users take on platforms that Dialogflow interacts with, such as Google Assistant, Slack, and Facebook Messenger.

Platform	Event	Description
Multiple	WELCOME	Generic welcome event for any platform.
Actions on Google	GOOGLE_ ASSISTANT_WELCOME	Triggered when the user starts a conversation with your action.
Slack	SLACK_WELCOME	Triggered when the user starts a conversation with your Slack bot.
Facebook	FACEBOOK_WELCOME	Triggered when the user starts a conversation with your Facebook Messenger bot.

Custom Events

Definition: Events defined for communication that cannot be captured through text or voice, such as button clicks, authorization, or timeouts.

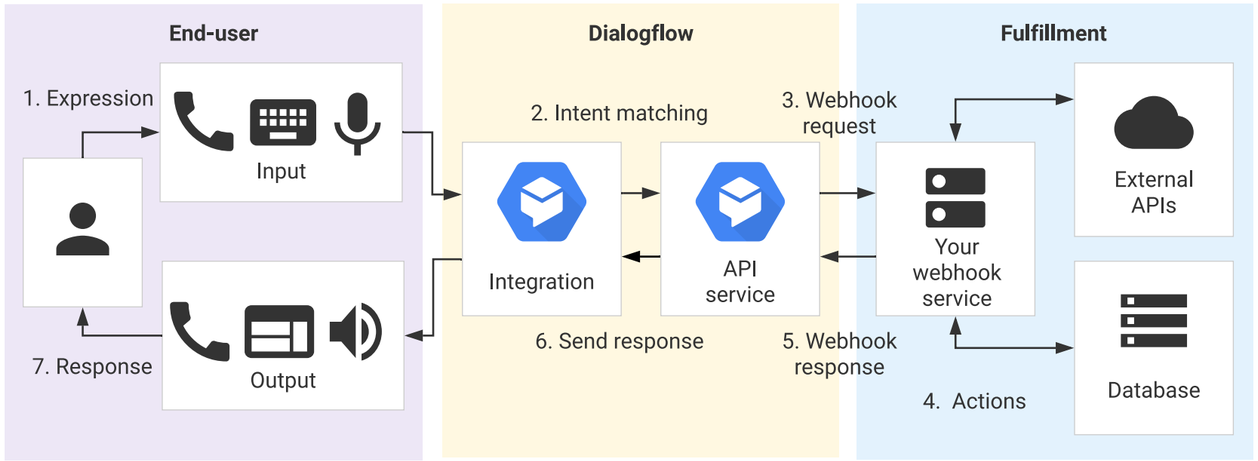
Fulfillment

What Is Fulfillment?[6]

Definition: Fulfillment allows the integration of Dialogflow agents with web services.

Intents are *enabled* for fulfillment.

Once enabled, intents will communicate with external APIs using a *webhook*.



[6] Image source

What Is A Webhook?

Definition: A *webhook* is a web server endpoint that allows applications to interact with the server API.

<URL>/endpoint

Webhooks in Dialogflow

Dialogflow fulfillment will follow the following course of action:

1. When an intent is enabled for fulfillment, it will make an HTTP `POST` request to the associated webhook with a JSON object about the matched event.
2. The webhook is designed to respond to your request appropriately, e.g., by looking up information in a database.
3. The webhook responds back with instructions for what Dialogflow should do next.

Standard `POST` request format:

```
POST https://my-service.com/action

Headers:
//user defined headers
Content-type: application/json

POST body: {
  // JSON
}
```

You have to design your webhook to be able to process the JSON that's passed by the `POST` and to respond in a way that Dialogflow can process.

In the assignment, we did this for you — we are using a webhook and external code to connect your agent with our server.

If you are using Dialogflow to connect with Google services (more common, preferred by Dialogflow), you can use the Inline Editor. (Our webhook works like the Dialogflow Inline Editor.)

We will cover some of the basics next.

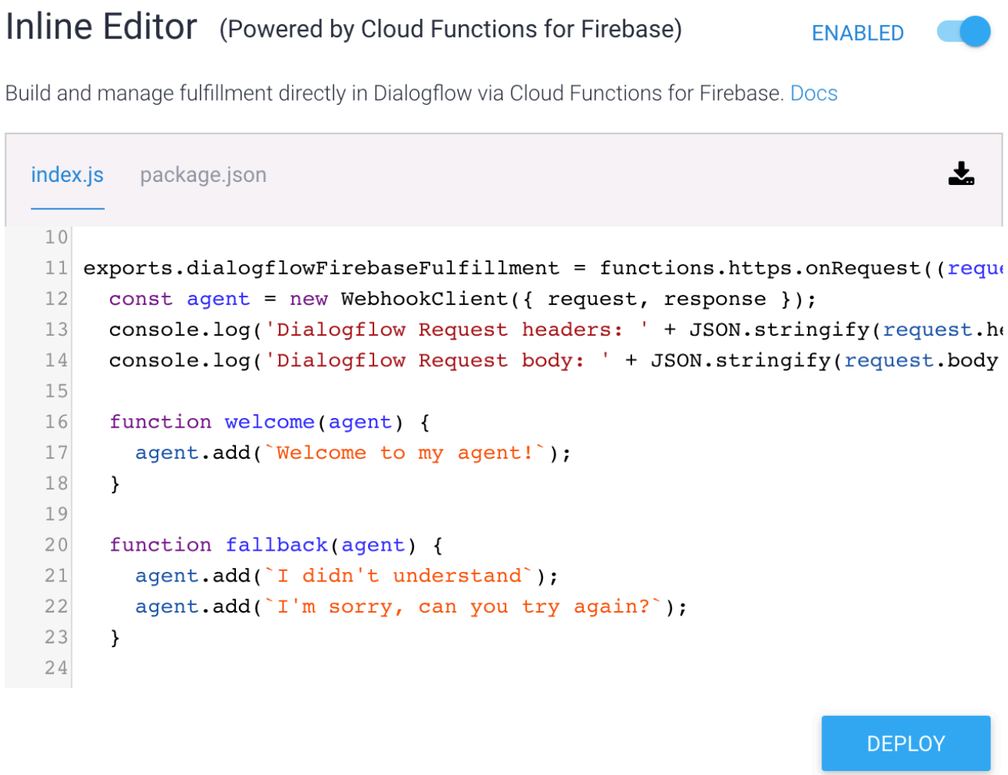
Dialogflow Fulfillment

Process for creating a custom fulfillment:

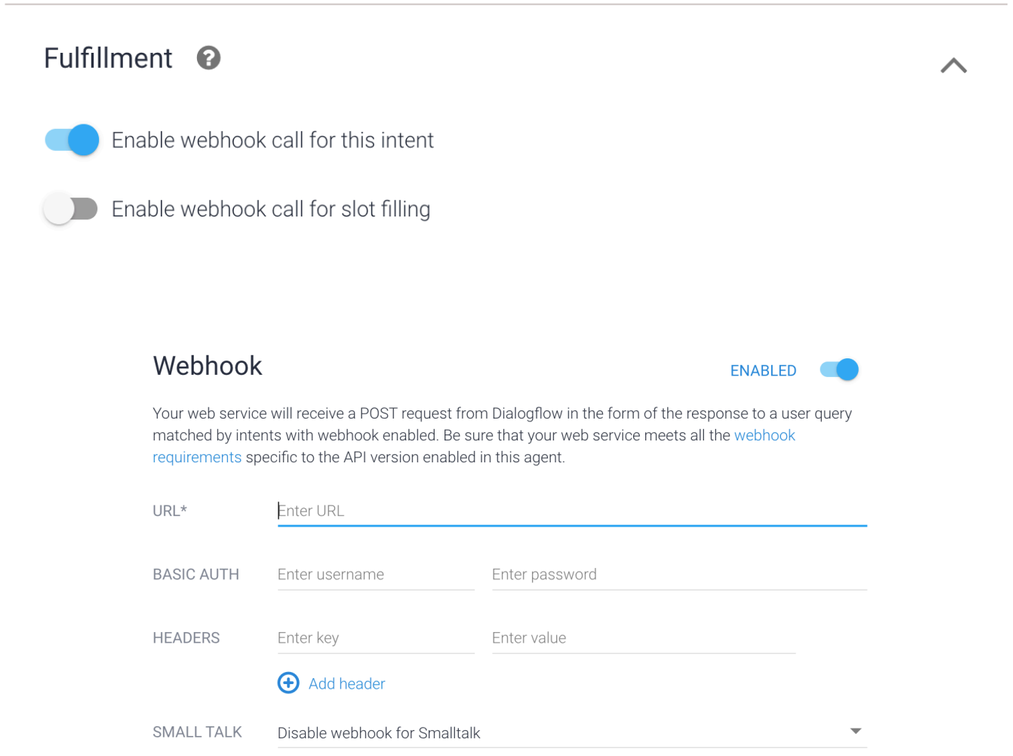
1. Enable fulfillment for the intent
2. Go to the Inline Editor and locate where the request is made
3. Define a function for your intent
4. Connect the intent to the function
5. Deploy

Step 1. Enable fulfillment for the intent

Standard procedure



Our procedure



Step 2. Locate where the request is made

Standard procedure

Our procedure

```
var express = require('express')
var app = express()
app.post('/', express.json(), (req, res) => {
  const agent = new WebhookClient({
    request : req, response: res
  })
    // Do your work here
})
```

Step 3. Define a function for your intent

Define a function for each intent:

```
function welcome() {
  agent.add('Welcome to our service!')
}
```

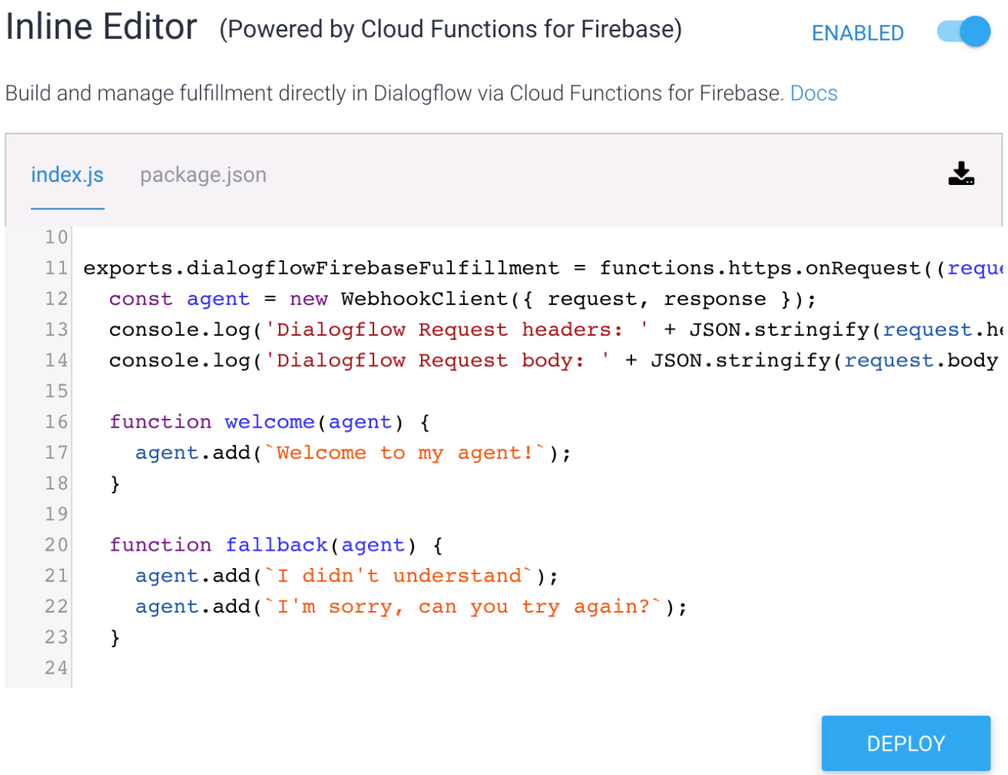
Step 4. Connect the intent to the function

Connect the function and intent through the intent map:

```
let intentMap = new Map()
intentMap.set('<IntentName>', welcome)
agent.handleRequest(intentMap)
```

Step 5. Deploy

Standard procedure



Our procedure

You don't need to do anything!

Now, let's get to the `// Do your work` part:

— Accessing user input
— Accessing user entities
— Setting and getting context

Everything is done through your `agent` object.

```
const { WebhookClient } = require("dialogflow-fulfillment");
const agent = new WebhookClient({ request: req, response: res });
```

How to Access User Input

To get the entire user query:

`agent.query`

Returns a string with the entire query and `null` if no value.

How to Access User Entities

To get user entities and other information associated with the intent:

`agent.parameters`

Returns a JS object with `key:value` pairs and `null` if no value.

Accessing Specific Parameters

We need to use `$parameter-name` to access a specific parameter. E.g.:

`agent.parameters.color`

Setting and getting context

To set a new outgoing context:

```
agent.context.set(context) // string or object
```

To get a context:

```
agent.context.get(contextName) // string name of context
```

Returns the context object.

An Example

Imagine you are on the product page on WiscShop:

User: I'd like to buy this t-shirt.

intent = purchaseItem; entity = { item: t-shirt, number: null; color: null };

Agent: How many do you want?
User: 3

```
intent = purchaseItem; entity = { item: t-shirt, number: 3; color:
null }
```

Agent: What color would you like?
User: Black

```
intent = purchaseItem; entity = { item: t-shirt, number: 3; color:
black }
```

What do we do now?

In the webhook:

```
app.post('/', express.json(), (req, res) => {
  const agent = new WebhookClient({ request: req, response: res })

  function purchase () {
    //to do
  }

  let intentMap = new Map()
  intentMap.set('purchaseItem', purchase)
  agent.handleRequest(intentMap)
})
```

Find which page they are on to find product ID.

```
fetch('URL/application/', <options>) // GET
```

Returns with a page key, e.g., `"page":"/username/tshirts/products/14"`

Grab 14, use the following three times:

```
fetch('URL/application/products/14', <options>) // POST
```

Formulate your response:

```
agent.add('Ok, I've added ' + agent.parameters.number + ' ' + agent.parameters.item)
```

Add user response and agent response to the messages.

Imagine that the user wanted to hear reviews first before purchasing, once the product was selected, we would want to set the context to that product so that we don't have to ask again.

```
agent.context.set({
  'name':'current-product',
  'lifespan': 5,
  'parameters':{
    'id':14
  }
});
```

```
agent.context.get('current-product');

{
  'name': 'current-product',
  'lifespan': 5,
  'parameters':{
    'id':14
  }
}
```

To Learn More

— Dialogflow documentation

— Online tutorials: e.g., Chatbots Life

What did we learn today?

— Introduction to Dialogflow, Recap
— Dialogflow Building Blocks, Part 2
— Fulfillment