

# DialogFlow 2

## CS571: Building User Interfaces

**Warmup:** Signup for an [ngrok account](#) and [clone today's starter code](#).

**Cole Nelson**

# Announcements

- HW12 is "due" Monday, 5/1, no-cost late week of 5/8. Cannot be extended further.
- Bonus Quiz worth 1 pt for the final week of class.
  - Final week of class' content will *only* be on the Bonus Quiz, not on the Final Exam.

# Announcements

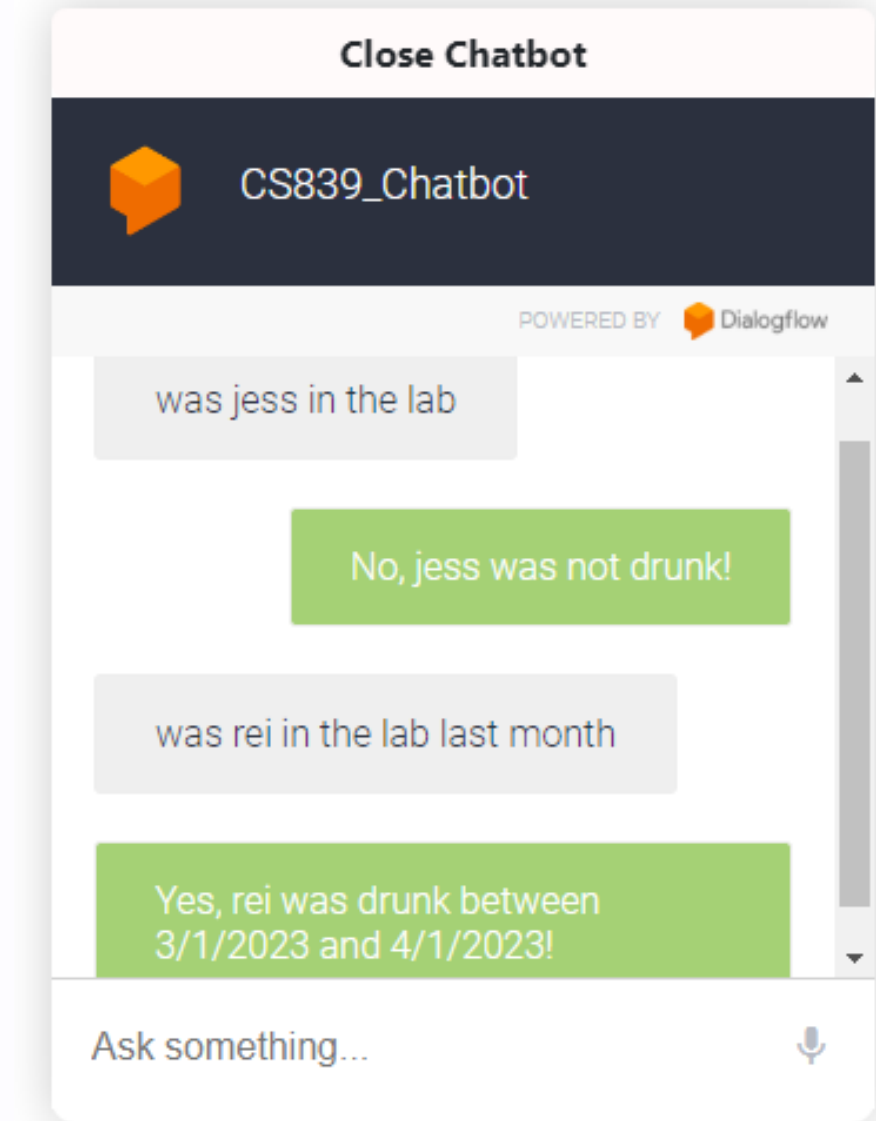
- End of semester survey is open until May 5th at [aefis.wisc.edu](http://aefis.wisc.edu).
  - All responses are anonymous.
  - Optional, but highly encouraged!
- **NPM library usage** due Friday, May 5th.
  - No late weeks can be used!
  - You may go back and modify a previous homework.
  - Can be used on DialogFlow as well!

# What will we learn today?

- A Review of DialogFlow
- How can we perform DialogFlow fulfillment?
- What are other neat DialogFlow features?

# Today's Goal

Be able to create an embeddable, usable voice agent!



# Key Concepts in DialogFlow

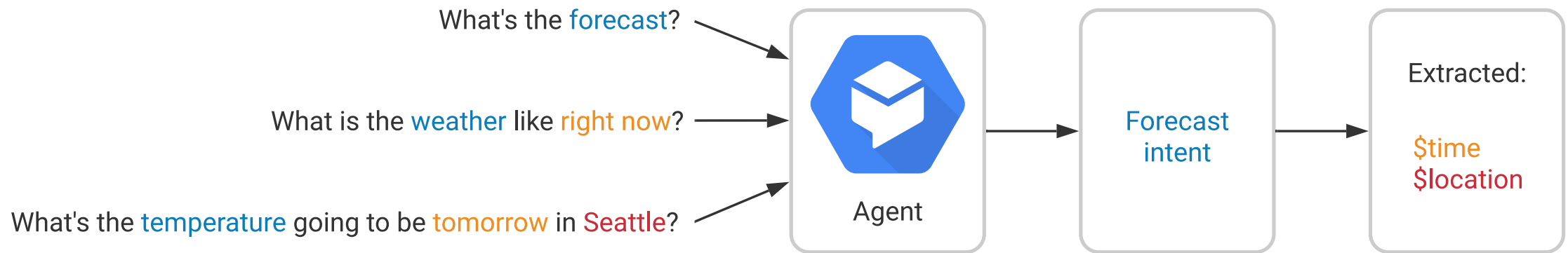
- Agents
  - Intents
    - Training Phrases
    - Parameters
    - Responses
  - Entities

# Intents

**Training Phrases:** Things the user may say to express an intent. DialogFlow recommends having many!

**Parameters:** Things that may vary in an expression, e.g. time, quantity, location.

**Responses:** How the system responds to the expression. Can include text, buttons, links, etc!



## Image Source



# Parameters, Entity Types, and Entities

These allow for more specificity of requests without exploding the intent space.

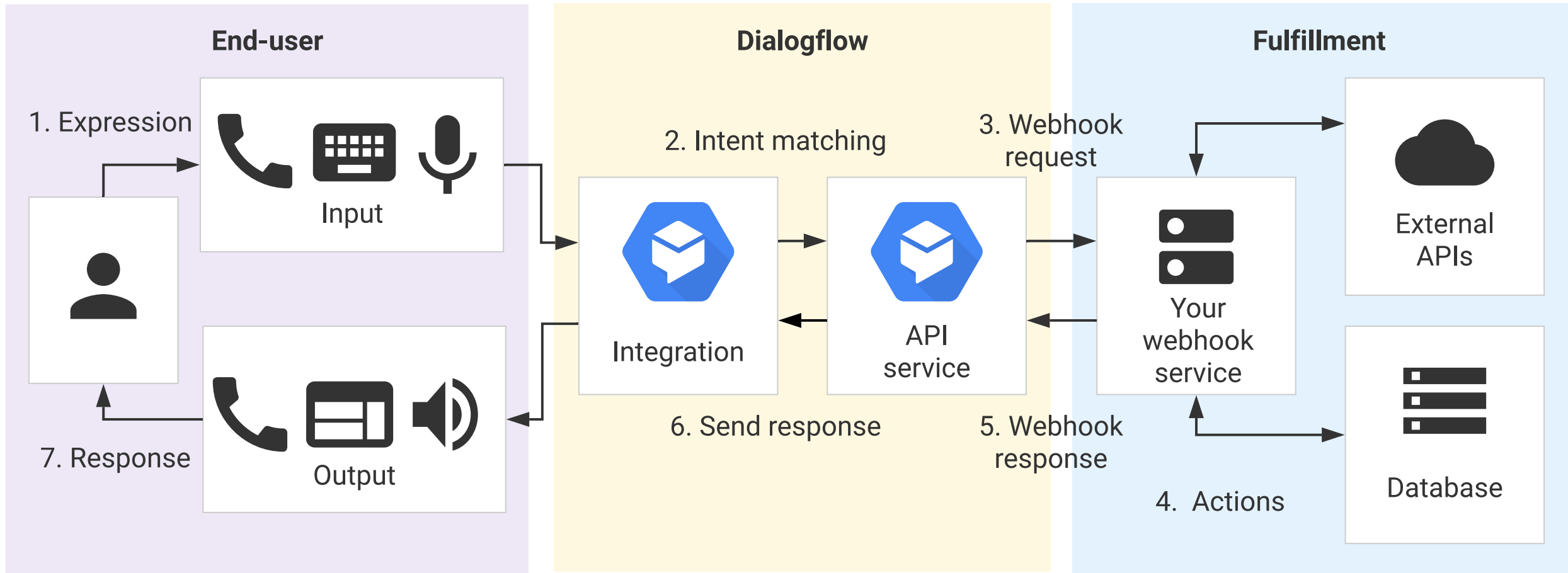
- **Parameter:** A variable to be expressed by the user
- **Entity Type:** The type of variable expressed
- **Entity:** A specific value that can be expressed

# DialogFlow Advanced Concepts

- WebHook Fulfillment
- Contexts
  - Followup Intents
- Events
- Integrations

# Webhook Fulfillment

Doing something "smart" with our agent.




## Image Source

# Types of Webhook Fulfillment

## Inline Editor (Powered by Cloud Functions for Firebase)

ENABLED 

Build and manage fulfillment directly in Dialogflow via Cloud Functions for Firebase. [Docs](#)

index.js package.json 

```
10
11 exports.dialogflowFirebaseFulfillment = functions.https.onRequest((request, response) => {
12   const agent = new WebhookClient({ request, response });
13   console.log('Dialogflow Request headers: ' + JSON.stringify(request.headers));
14   console.log('Dialogflow Request body: ' + JSON.stringify(request.body));
15
16   function welcome(agent) {
17     agent.add('Welcome to my agent!');
18   }
19
20   function fallback(agent) {
21     agent.add('I didn\'t understand');
22     agent.add('I\'m sorry, can you try again?');
23   }
24
```

DEPLOY

## Fulfillment


 Enable webhook call for this intent

 Enable webhook call for slot filling

## Webhook

ENABLED 

Your web service will receive a POST request from Dialogflow in the form of the response to a user query matched by intents with webhook enabled. Be sure that your web service meets all the [webhook requirements](#) specific to the API version enabled in this agent.

URL*	<input type="text" value="Enter URL"/>	
BASIC AUTH	<input type="text" value="Enter username"/>	<input type="text" value="Enter password"/>
HEADERS	<input type="text" value="Enter key"/>	<input type="text" value="Enter value"/>
	 Add header	
SMALL TALK	<input type="text" value="Disable webhook for Smalltalk"/>	

# DialogFlow Webhook

DialogFlow makes a  
POST *request* to `/`.

[Explore Docs](#)

```
{
  "responseId": "response-id ✎",
  "session": "projects/project-id ✎/agent/sessions/s",
  "queryResult": {
    "queryText": "End-user expression",
    "parameters": {
      "param-name": "param-value"
    },
    "allRequiredParamsPresent": true,
    "fulfillmentText": "Response configured for match",
    "fulfillmentMessages": [
      {
        "text": {
          "text": [
            "Response configured for matched intent"
          ]
        }
      }
    ],
    "outputContexts": [
      {
        "name": "projects/project-id ✎/agent/session",
        "lifespanCount": 5,
        "parameters": {
          "param-name": "param-value"
        }
      }
    ]
  }
}
```

# DialogFlow Webhook

DialogFlow expects a *response* of a certain format.

[Explore Docs](#)

```
{
  "fulfillmentMessages": [
    {
      "text": {
        "text": [
          "Text response from we"
        ]
      }
    }
  ]
}
```

# Accessing Our Webhook

We are running a server on localhost... how can we tell DialogFlow where to POST ?

Solution: [ngrok](#) (or any reverse-proxy service)

```
https://088b-146-151-104-188.ngrok.io -> http://localhost:53705
```

**Pro-Tip:** You can use this with your React apps!



# Intent Mapping

```
const intentMap = {
  "GetNumUsers": getNumUsers,
  "GetNumMessages": getNumMsgs,
  "GetChatroomMessages": getChatMsgs
}

app.post('/', (req, res) => {
  const intent = req.body.queryResult.intent.displayName;
  if (intent in intentMap) {
    intentMap[intent](req, res);
  } else {
    console.error(`Could not find ${intent} in intent map!`)
    res.status(404).send(JSON.stringify({ msg: "Not found!" }));
  }
})
```

# Responding to Request

```
function getNumUsers(req, res) {  
  // TODO Fetch data from API  
  res.status(200).send({  
    fulfillmentText: "Hello from getNumUsers!"  
  });  
}
```

# Responding to Request - Alternative

```
function getNumUsers(req, res) {  
  // TODO Fetch data from API  
  res.status(200).send({  
    fulfillmentMessages: [  
      {  
        text: {  
          text: [  
            `Hello from getNumUsers!`  
          ]  
        }  
      }  
    ]  
  });  
}
```

# Responding to Request - Card

```
function getNumUsers(req, res) {  
  // TODO Fetch data from API  
  res.status(200).send({  
    fulfillmentMessages: [  
      {  
        card: {  
          title: "Hello",  
          subtitle: "World",  
          buttons: [  
            {  
              text: "Click Me",  
              postback: "https://example.com/"  
            }  
          ]  
        }  
      }  
    ]  
  });  
}
```

# Let's Build a Better Agent!

**Your turn!** Signup for an [ngrok account](#) and [clone today's starter code](#).

# A Review of Async

**Definition:** not happening or done at the same time.

We `promise` something will happen in the future.

If we fulfill our promise, `then` we do something.

Otherwise, we `catch` and handle the error.

# async / await

- equivalent way to handle asynchronous behavior
- `await` must be inside of an `async` function
- `await` waits for right-hand-side to complete
- every `async` function returns a `Promise`
- a synchronous function may spawn `async` behavior
- an `async` function always happens asynchronously

# Equivilant Handling!

```
function getLogos() {  
  fetch("https://www.example.com/logos?amount=20")  
    .then(res => res.json())  
    .then((newLogos) => {  
      setLogos(newLogos);  
    })  
}
```

```
async function getLogos() {  
  const resp = await fetch("https://www.example.com/logos?amount=20");  
  const newLogos = await resp.json();  
  setLogos(newLogos);  
}
```



# Equivilant Handling - Errors

```
function getLogos() {  
  fetch("https://www.example.com/logos?amount=20")  
    .then(res => res.json())  
    .then((newLogos) => {  
      setLogos(newLogos);  
    })  
    .catch(e => alert("Something went wrong!"))  
}
```

# Equivilant Handling - Errors

```
async function getLogos() {  
  try {  
    const resp = await fetch("https://www.example.com/logos?amount=20");  
    const newLogos = await resp.json();  
    setLogos(newLogos);  
  } catch (e) {  
    alert("Something went wrong!")  
  }  
}
```

# Let's Build a Better Agent!

Using `async` / `await` ...

# Context

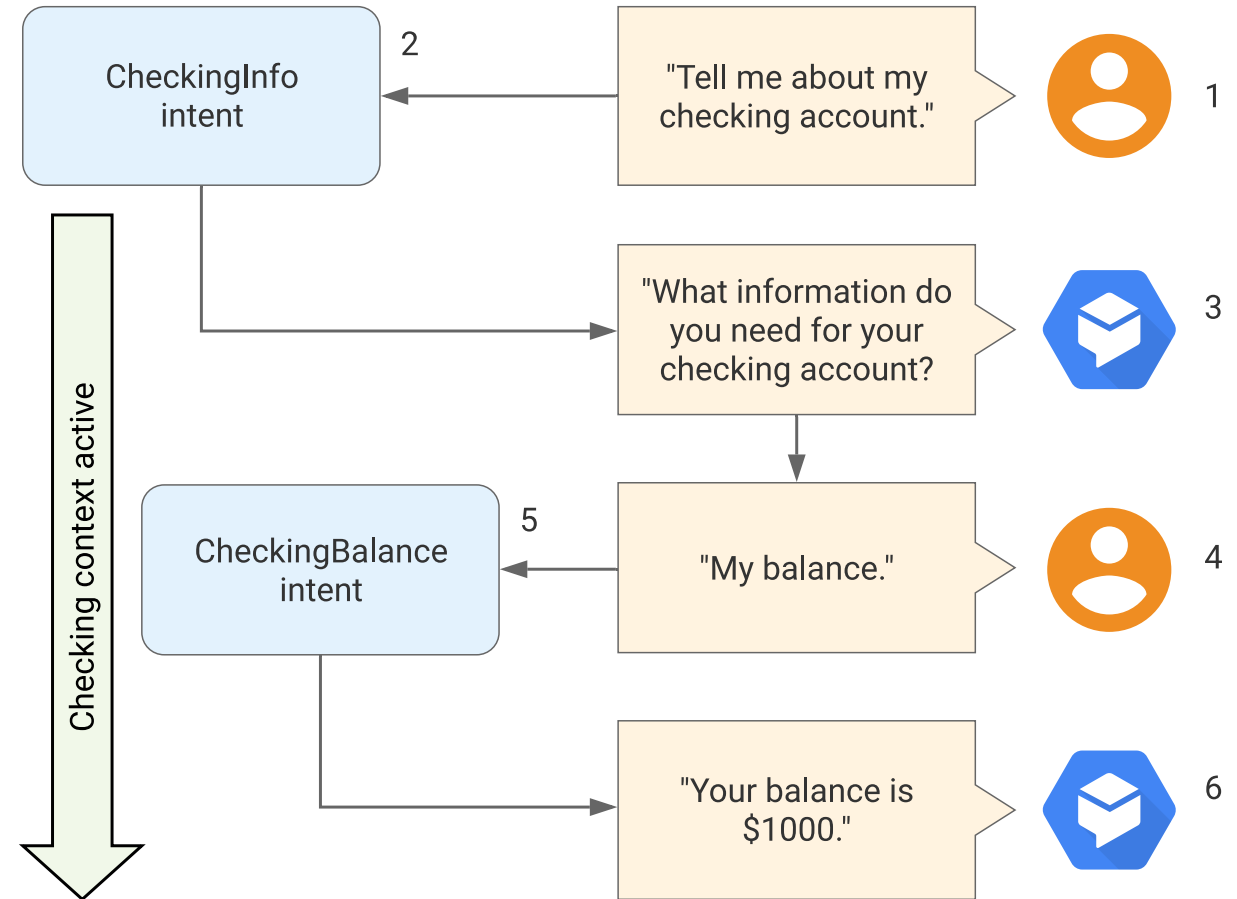
Context is the common ground that communicators have about the conversation that allows them to infer ambiguous speech such as "pass me that."

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

# Context

The training phrase "My balance" only makes sense in the context of CheckingInfo.

Image Source



# Contexts

Intents have **input** and **output** contexts.

A context has a **lifespan**. The lifespan may be defined in terms of **time** or **turns**.

A context can be **added** or **removed** by an intent.

Can also be used to make *followup intents*.

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

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

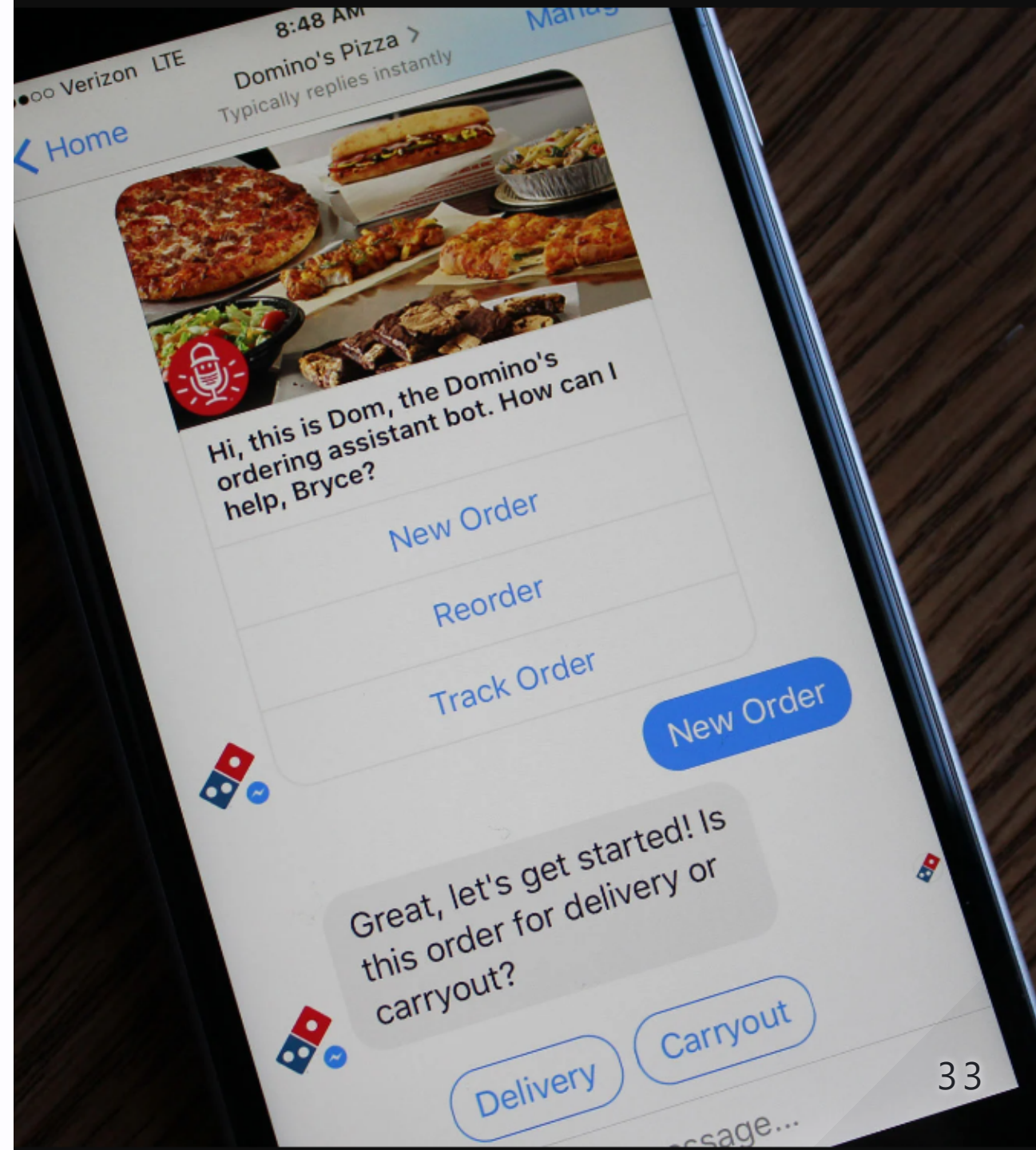
Platform	Event	Description
Multiple	WELCOME	Generic welcome event for any platform.
Actions on Google	GOOGLE_ASSISTANT_WELCOME	Triggered when the user starts a conversation with your action.
Slack	SLACK_WELCOME	Triggered when the user starts a conversation with your Slack bot.
Facebook	FACEBOOK_WELCOME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

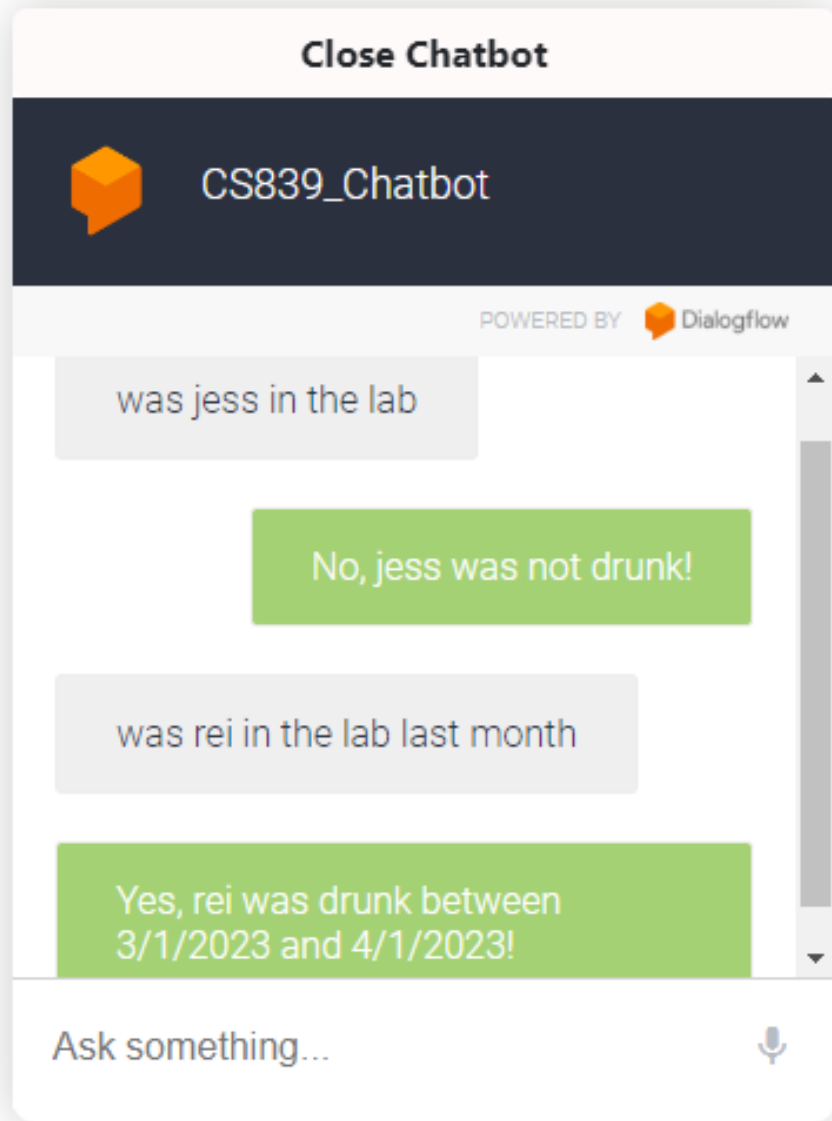


# Integrations

DialogFlow agents can be integrated with other platforms such as Google Assistant, Slack, and Messenger.

[Image Source](#)





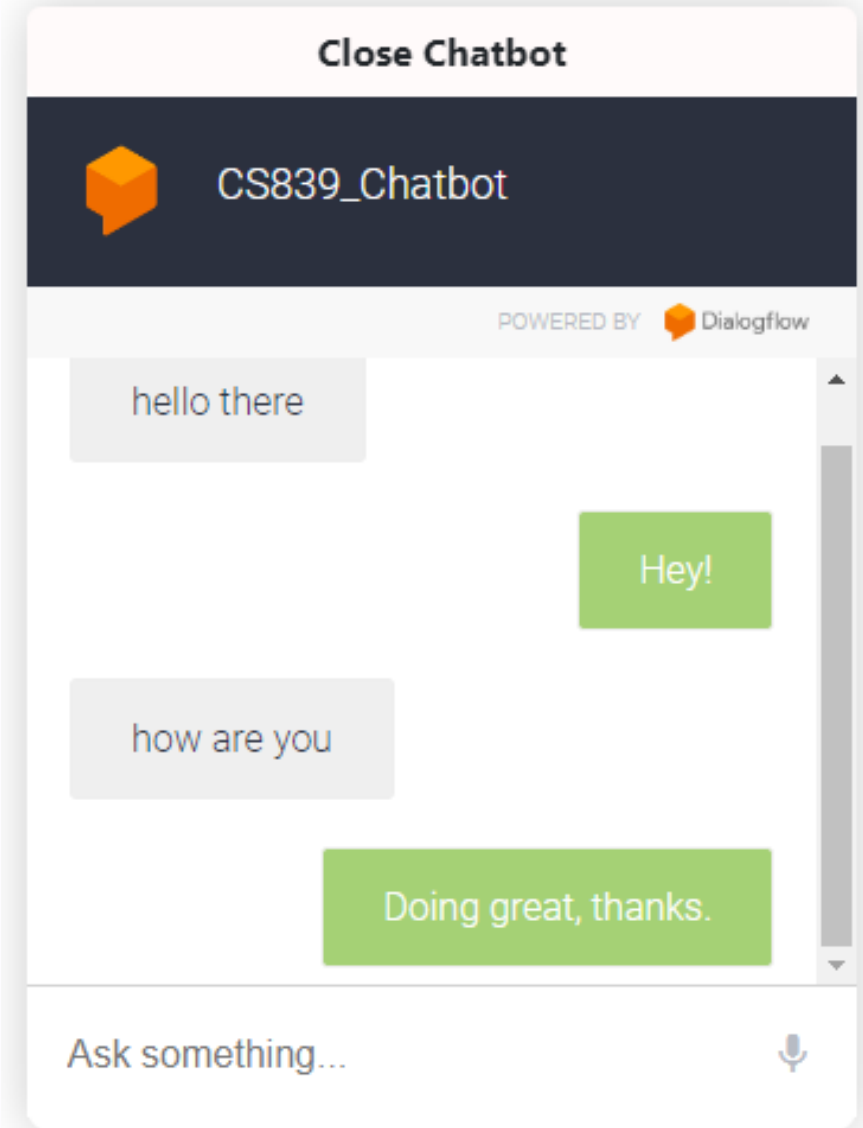
# Integrations

Can also be embedded  
as an `iframe` ...

```
<iframe
  allow="microphone;"
  width="350"
  height="430"
  src="https://console.dialogflow.com/
    api-client/demo/embedded/xxxxxxx">
</iframe>
```

# Integrations

**SmallTalk** is used to provide responses to casual conversation.



# HW12 Sneak Peak

Use DialogFlow to create a voice agent for navigating BadgerChat...

- Read-only
- Use Dialogflow Agent for Steps 1-3
- Use a Webhook for Steps 4-6

You will have completed a web, mobile, and voice application for the same service! 🎉

**Use DialogFlow ES Trial Edition!**

# DialogFlow Library

Looking to get your 3rd Party Library assignment done? Consider using [dialogflow-fulfillment](#)!

# What did we learn today?

- A Review of DialogFlow
- How can we perform DialogFlow fulfillment?
- What are other neat DialogFlow features?

# Questions?