

A  
PROJECT REPORT  
ON

**A CHATBOT**

SUBMITTED BY  
**SUTAPA SEN**

FOR DATA SCIENCE  
INTERNSHIP  
IN  
**COINCENT**

# **Abstract**

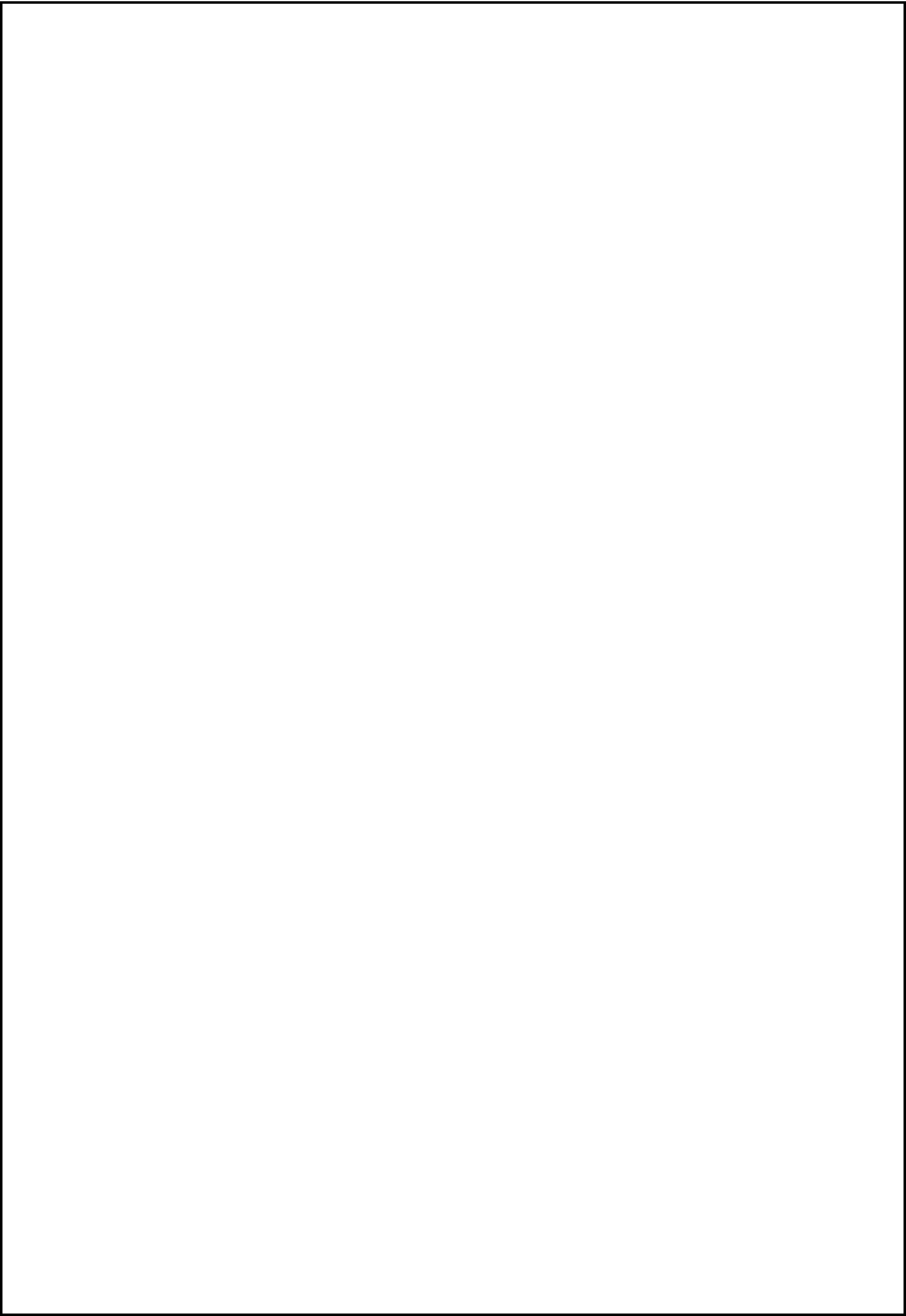
People interact with systems more and more through voice assistants and chatbots. The days of solely engaging with a service through a keyboard are over. These new modes of user interaction are aided in part by advancements in Artificial Intelligence and Machine Learning technology.

This project aimed to implement a web-based chatbot to assist with online banking, using tools that expose artificial intelligence methods such as natural language understanding. Allowing users to interact with the chatbot using natural language input and to train the chatbot using appropriate methods so it will be able to generate a response. The chatbot will allow users to view all their personal banking information all from within the chatbot.

The produced prototype was found to be a very useful tool to justify the need of a modern method of interaction to be integrated within many services offered by banks and financial businesses. In an industry with low user satisfaction rates and limited technology to increase accessibility. It is clear the chatbot overcomes the challenges banks face to increase the use of their services and gain a competitive edge over leading competitors.

With many people adopting Smart Assistant Devices such as Google Home or Amazon's Alexa. The chatbot was tested across a range of devices such as Google

Home and Assistant on android devices to outline the key differences between the two modes of interaction , spoken and text dialog. These test were carried out to identify the value in integrating such technology surrounding the recent interest in chatbots and conversational interfaces. Proving chatbots can be applied to a specific domain to enhance. .



# **OBJECTIVE**

Save effort and time for user. It's very interesting to chat with chatbot.

Provide detailed information about so many current things.

Easy access to information.

# **INTRODUCTION**

Chatbot is a computer program that humans will interact with in natural spoken language and including artificial intelligence techniques such as NLP (Natural language processing) that makes the chatbot more interactive and more reliable.

Based on the recent epidemiological situation, the increasing demand and reliance on electronic education has become very difficult to access to the university due to the curfew imposed, and this has led to limited access to information for academics at the university.

This project aims to build a chatbot for answering random question, every person who asks about the current affairs and chat like a friend.

# **METHODOLOGY**

It's been a while since Flutter is out there and its upsurge these days is phenomenal. The unparalleled performance of the framework is one of the reasons for its fondness among App developers other than the fact that it helps build cross-platform apps.

Here how to build a ChatBot in Flutter using the Dialog Flow development suite is clearly written. So, first let's see the points that:

- Brief about Dialogflow
- Setup and Understand the console
- Train your Agent
- Integrate Dialogflow in Flutter App

Let's move on with the first part where we should know about Dialogflow which is the building block of the ChatBot.

## What is Dialogflow?

Dialogflow is a development suite that incorporates Google's machine learning expertise to build end-to-end, deploy-everywhere interfaces for websites, mobile applications, and IoT devices.

*Build natural and rich conversational experiences*

*Give users new ways to interact with your product by building engaging voice and text-based conversational interfaces, such as voice apps and chatbots, powered by AI. Connect with users on your website, mobile app, the Google Assistant, Amazon Alexa, Facebook Messenger, and other popular platforms and devices.*

### Advantages of Dialogflow:

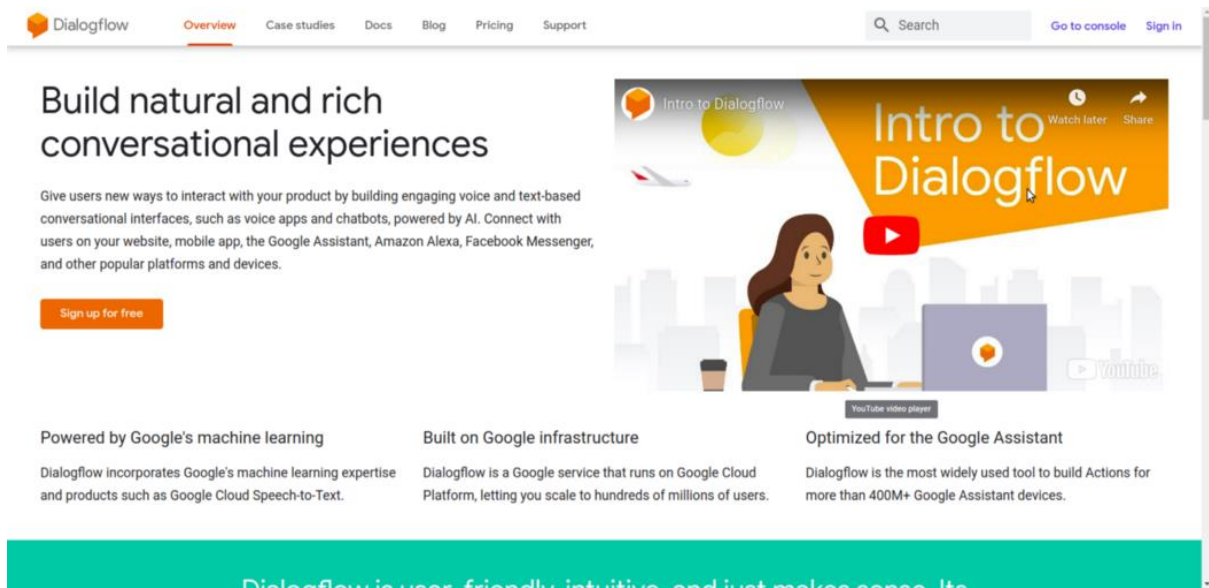
- Powered by Google's Machine learning
- Built on Google's Infrastructure
- Optimized for Google Assistant

To know more about Dialogflow please refer to their video,

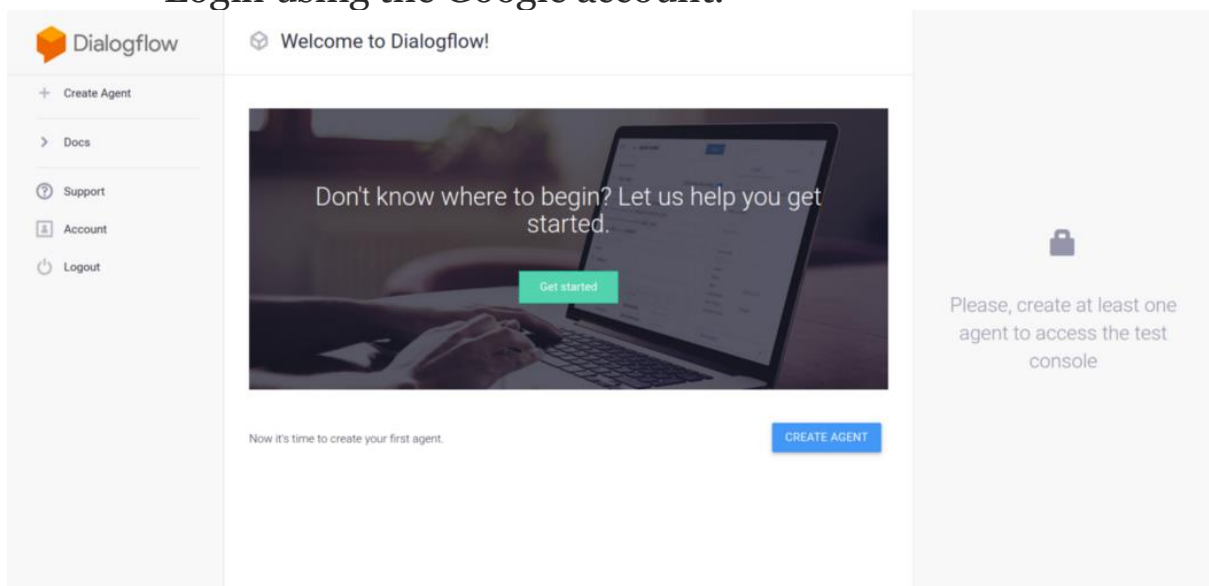
### Setup and Understand Dialogflow console:

Moving on, Lets head over to the Dialogflow website by clicking on the <https://dialogflow.com/>.





- Click on the [Go to Console](#) at the top right corner of the website.
- Login using the Google account.



- Create your first agent. Give it a name, set the time zone and your default language for the agent.

Agent name

CREATE

DEFAULT LANGUAGE ?

English — en

Primary language for your agent. Other languages can be added later.

DEFAULT TIME ZONE

(GMT+6:00) Asia/Almaty

Date and time requests are resolved using this timezone.

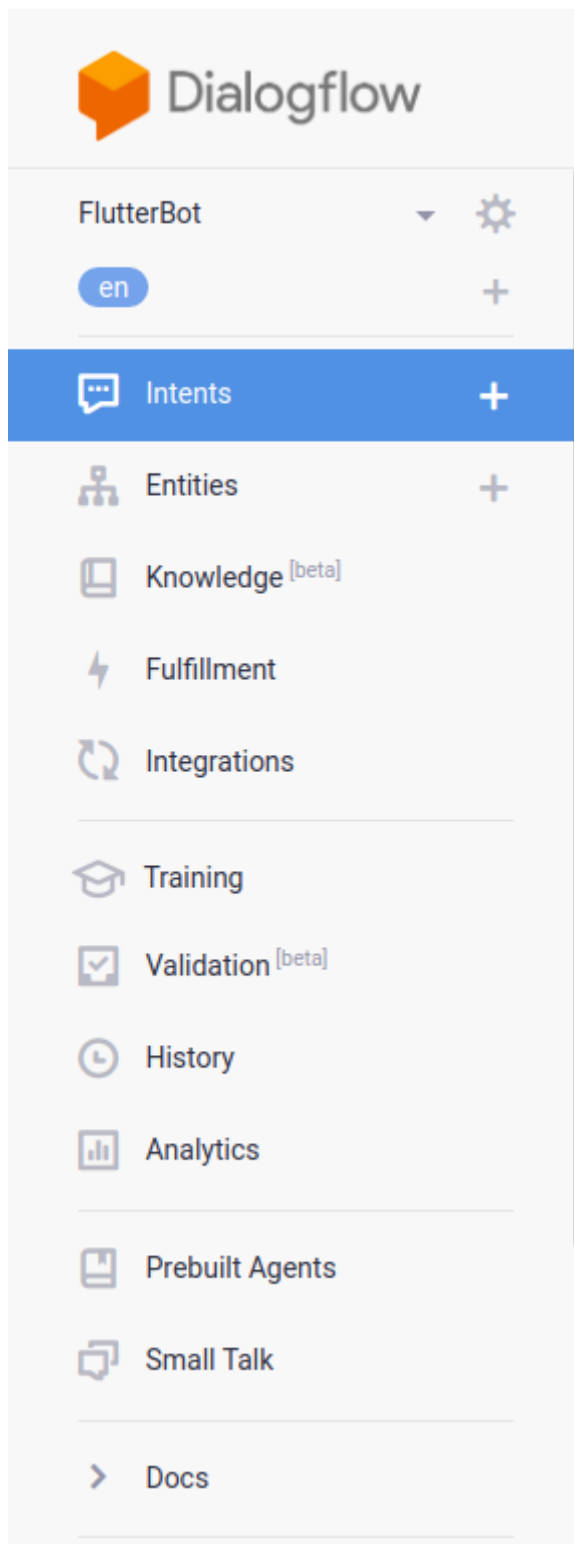
GOOGLE PROJECT

Create a new Google project

Enables Cloud functions, Actions on Google and permissions management.

- After clicking on `Create` , you'll get the console with all the features that you can implement in your Agent.

## Understanding the basic console:



Two things which are the base of an Agent you just created.

- Intents

- Entities

### **Intents:**

An *intent* categorizes an end-user intention for one conversation turn. For each agent, you can define many intents, where your combined intents can handle a complete conversation. When an end-user writes or says something, Dialogflow matches the end-user expression to the best intent in your agent. and based on that intent a response is provided by the agent to the end-user.

*Intents are mappings between a user's queries and actions fulfilled by your software.*

There are two default Intents provided by the Dialogflow, namely:

- **Default Fallback Intent:** These types of intents are used when the user's input expression doesn't match the Intent defined for the Agent.



## Default Fallback Intent



SAVE



### Responses ?



DEFAULT +

#### Text Response



- 1 I didn't get that. Can you say it again?
- 2 I missed what you said. What was that?
- 3 Sorry, could you say that again?
- 4 Sorry, can you say that again?
- 5 Can you say that again?
- 6 Sorry, I didn't get that. Can you rephrase?
- 7 Sorry, what was that?
- 8 One more time?
- 9 What was that?
- 10 Say that one more time?
- 11 I didn't get that. Can you repeat?
- 12 I missed that, say that again?
- 13 Enter a text response variant

- **Default Welcome Intent:** These types of intents are used when the end-user starts the conversation.

- Default Welcome Intent

SAVE



Responses ?



DEFAULT +

Text Response



- 1 Hi! How are you doing?
- 2 Hello! How can I help you?
- 3 Good day! What can I do for you today?
- 4 Greetings! How can I assist?
- 5 Enter a text response variant

ADD RESPONSES

☐ Set this intent as end of conversation ?

## Entities:

Each intent has a parameter and each intent parameter has a type, called the *entity type*, which dictates exactly how data from an end-user expression is extracted.


Dialogflow provides predefined system entities. For example, there are system entities for matching dates, times, colors, email addresses, and so on. You can also create your own developer entities for matching custom data. For example, you could define a *product* entity that can match the types of products available for purchase with an Online store agent.



*Entities are objects your app or device takes action on.*



## Train your Agent:


Now, that you have basic knowledge about the Dialogflow console, all that you'll need to build a basic Agent we can now move forward to train our agent with test cases.


- Create the Intents based on your requirement
- Add the training phrases to the Intents
- Add the response phrases for those training phrases
- Add the Action and Parameters if required for that Intent
- Add the Events which will trigger the Intent on the basis of the event occurrence irrespective of the training phrases, if any.
- Add the [context](#) to the Intent.

**Training phrases** 


Search training phrases  

 Add user expression 

 Tell me about Flutter

 What is Flutter



### Training Phrases for the Intent

**Responses** 

DEFAULT

GOOGLE ASSISTANT

+

**Text Response**  

1


Flutter is a open-source cross-platform mobile app development framework which, as Google defines it, "allows you to build beautiful native apps on iOS and Android from a single codebase".

2

Enter a text response variant

ADD RESPONSES

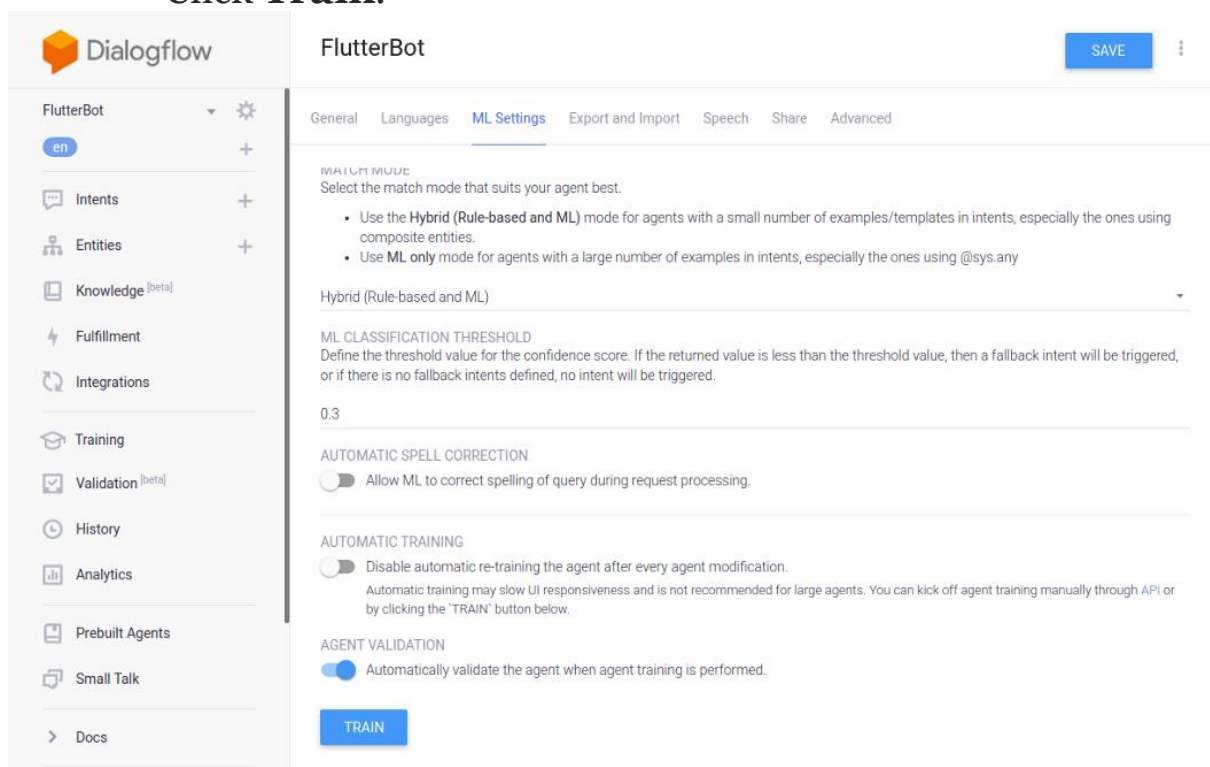
☐

Set this intent as end of conversation 

## Response Phrases for the Intent

Once you create an Intent then you can go on creating different intents for different queries. As of now, Dialogflow allows you to create 780 Intents and after that, you need to update the machine learning algorithm manually which you can do by following the steps:

- Click on the gear icon for your agent.
- Click the **ML Settings** tab.
- Click **Train**.

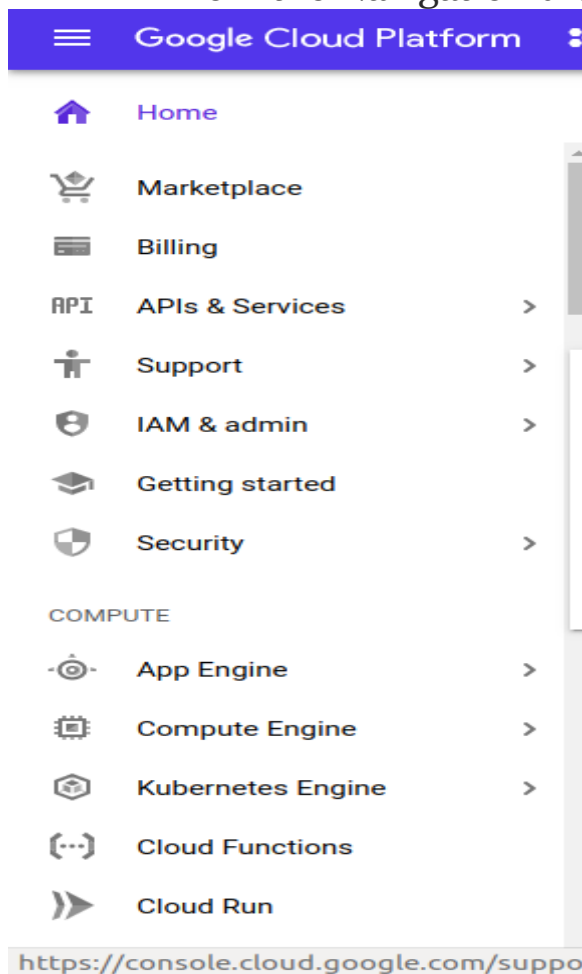


Once you have trained your Agent, now you're ready to integrate your Agent to your Flutter app and before heading over to Flutter, you have to download a JSON file from the console which helps your

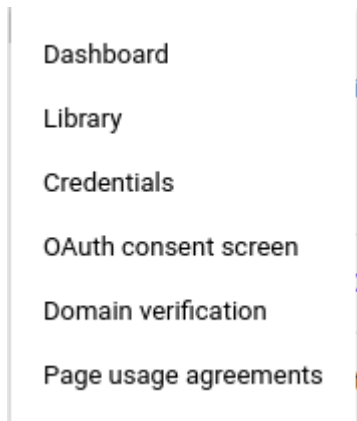


App to connect to the Agent you've just created. To do that follow these steps:

- Open the [Google Cloud Console](#).
- Select your Project.
- From the Navigation drawer click on `API's & Services`

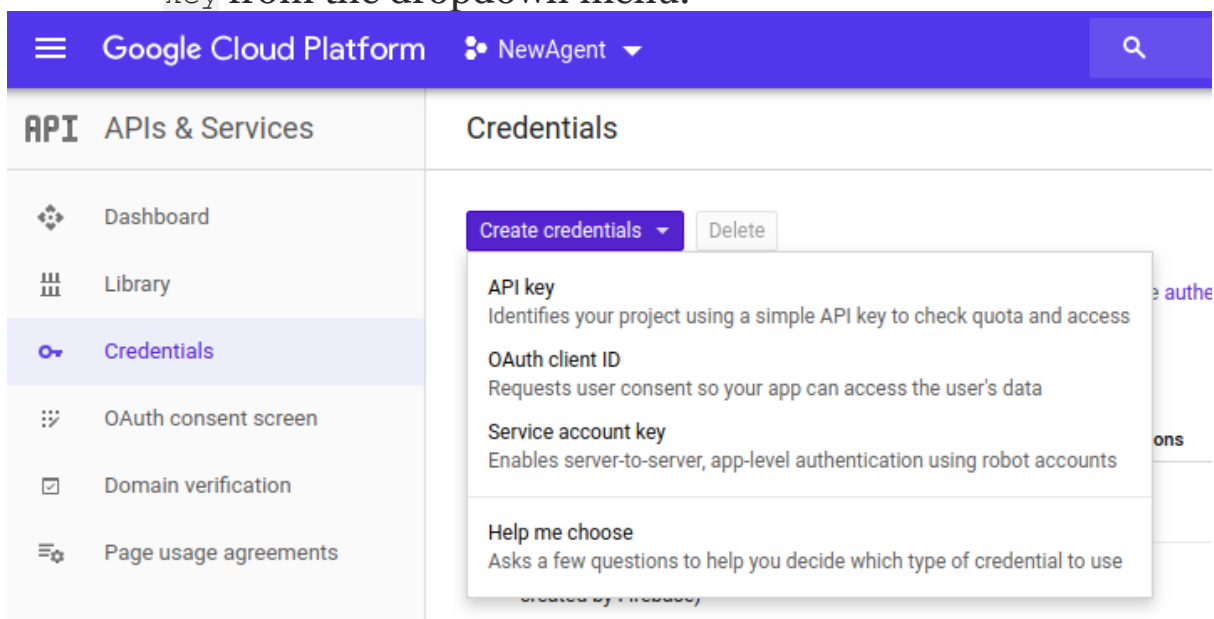


- Choose `Credentials` from the options.

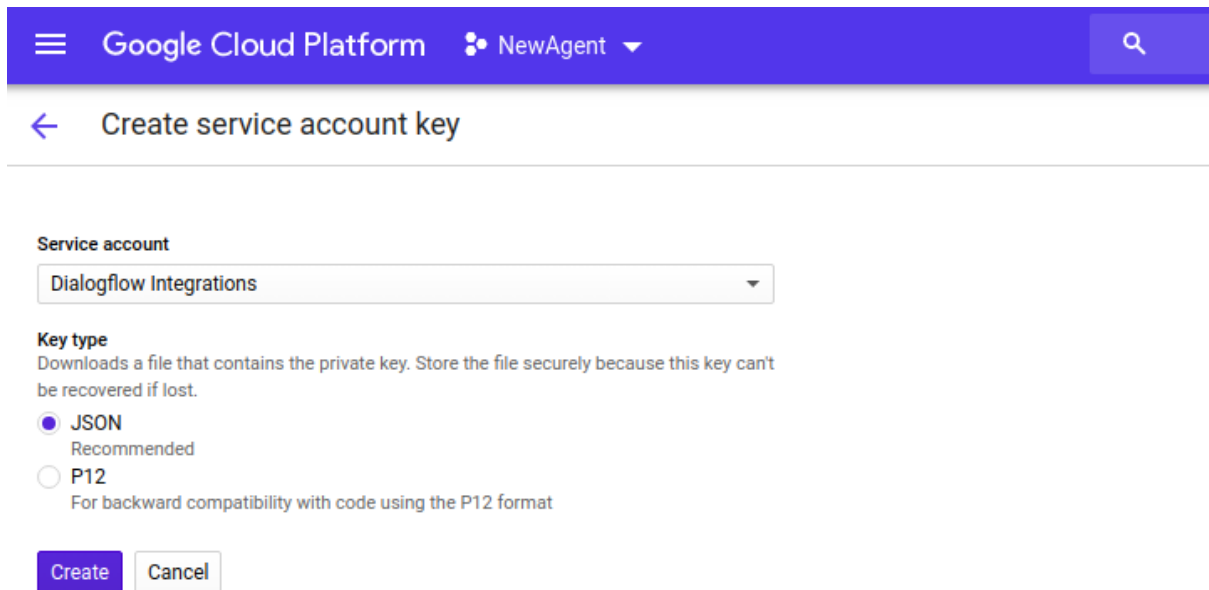


- From the Credentials window, Select `Service account key` from the dropdown menu.

- From the Credentials window, Select `Service account key` from the dropdown menu.



- Next, select the `Dialogflow integration` option from the Service Account dropdown menu and select `JSON` from the radio button below that.



Google Cloud Platform NewAgent

← Create service account key

Service account

Dialogflow Integrations

Key type

Downloads a file that contains the private key. Store the file securely because this key can't be recovered if lost.

☒ JSON  
Recommended

☐ P12  
For backward compatibility with code using the P12 format

Create Cancel

That's it, we'll have a JSON file downloaded which we'll need in the next steps.

## Integrate Dialogflow in Flutter App

Now, that you have your agent ready and trained you're all set to integrate it into your Flutter app.

- First, add the dependency to your pubspec.yaml file. Please make sure to the latest version.

```
dependencies:  
flutter_dialogflow: ^0.1.3
```

- Create a folder in your project with the name `assets` and move the JSON file that you've downloaded from the Google Cloud Console into it.

- Open the pubspec.yaml file and add,

```
flutter:  
  uses-material-design: true
```

```
assets:  
  - assets/[name_of_your_json_file].json
```

I'll let you know the part where you setup the code for Dialogflow in the app.

```
AuthGoogle authGoogle = await AuthGoogle(fileJson:  
"assets/[your_json_file_name].json").build();  
Dialogflow dialogFlow =  
Dialogflow(authGoogle: authGoogle, language: Language.english);  
AIResponse response = await dialogFlow.detectIntent(query);
```

*Note: use the same filename as mentioned in the pubspec.yaml file.*

That's all we'll need to get our Agent to respond to our questions. The `response` will have our Agent responses based on our query.

# CODE

## main.dart

```
import 'package:chatbot/Messages.dart';
import 'package:dialog_flowtter/dialog_flowtter.dart';
import 'package:flutter/material.dart';

void main() => runApp(Myapp());

class Myapp extends StatelessWidget {
  const Myapp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'SSBot',
      theme: ThemeData(brightness: Brightness.dark),
      home: Home(),
    );
  }
}

class Home extends StatefulWidget {
  const Home({super.key});

  @override
  State<Home> createState() => _HomeState();
}

class _HomeState extends State<Home> {
  late DialogFlowtter dialogFlowtter;
  final TextEditingController _controller = TextEditingController();

  List<Map<String, dynamic>> messages = [];

  @override
  void initState() {
    DialogFlowtter.fromFile().then((instance) => dialogFlowtter = instance);
    super.initState();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('SSBot'),
```

```

    ),
    body: Container(
      child: Column(
        children: [
          Expanded(child: MessagesScreen(messages: messages)),
          Container(
            padding: EdgeInsets.symmetric(horizontal: 14, vertical: 8),
            color: Colors.deepPurple,
            child: Row(
              children: [
                Expanded(
                  child: TextField(
                    controller: _controller,
                    style: TextStyle(color: Colors.white),
                  )),
                IconButton(
                  onPressed: () {
                    sendMessage(_controller.text);
                    _controller.clear();
                  },
                  icon: Icon(Icons.send))
              ],
            ),
          ),
        ],
      ),
    ),
  );
}

sendMessage(String text) async {
  if (text.isEmpty) {
    print('Message is empty');
  } else {
    setState(() {
      addMessage(Message(text: DialogText(text: [text])), true);
    });

    DetectIntentResponse response = await dialogFlowtter.detectIntent(
      queryInput: QueryInput(text: TextInput(text: text));
    if (response.message == null) return;
    setState(() {
      addMessage(response.message!);
    });
  }
}

addMessage(Message message, [bool isUserMessage = false]) {

```

```
    messages.add({'message': message, 'isUserMessage': isUserMessage});  
  }  
}
```

## messages.dart

```
import 'package:flutter/material.dart';  
  
class MessagesScreen extends StatefulWidget {  
  final List messages;  
  const MessagesScreen({super.key, required this.messages});  
  
  @override  
  State<MessagesScreen> createState() => _MessagesScreenState();  
}  
  
class _MessagesScreenState extends State<MessagesScreen> {  
  @override  
  Widget build(BuildContext context) {  
    var w = MediaQuery.of(context).size.width;  
    return ListView.separated(  
      itemBuilder: (context, index) {  
        return Container(  
          margin: EdgeInsets.all(10),  
          child: Row(  
            mainAxisAlignment: widget.messages[index]['isUserMessage']  
              ? MainAxisAlignment.end  
              : MainAxisAlignment.start,  
            children: [  
              Container(  
                padding: EdgeInsets.symmetric(vertical: 14, horizontal:  
14),  
                decoration: BoxDecoration(  
                  borderRadius: BorderRadius.only(  
                    bottomLeft: Radius.circular(  
                      20,  
                    ),  
                    topRight: Radius.circular(20),  
                    bottomRight: Radius.circular(  
                      widget.messages[index]['isUserMessage'] ? 0 :  
20),  
                    topLeft: Radius.circular(  

```

```

                                widget.messages[index]['isUserMessage'] ? 20 :
0),
                                ),
                                color: widget.messages[index]['isUserMessage']
                                    ? Colors.grey.shade800
                                    : Colors.grey.shade900.withOpacity(0.8)),
                                constraints: BoxConstraints(maxWidth: w * 2 / 3),
                                child:
                                    Text(widget.messages[index]['message'].text.text[0])),
                                ],
                                ),
                                );
                                },
                                separatorBuilder: (_, i) => Padding(padding: EdgeInsets.only(top:
10)),
                                itemCount: widget.messages.length);
                                }
                                }

```

## widget\_test.dart

```

// This is a basic Flutter widget test.
//
// To perform an interaction with a widget in your test, use the WidgetTester
// utility in the flutter_test package. For example, you can send tap and
scroll
// gestures. You can also use WidgetTester to find child widgets in the widget
// tree, read text, and verify that the values of widget properties are
correct.

// ignore_for_file: unused_import

import 'package:flutter/material.dart';
import 'package:flutter_test/flutter_test.dart';

import 'package:chatbot/main.dart';

void main() {
  testWidgets('Counter increments smoke test', (WidgetTester tester) async {
    // Build our app and trigger a frame.
    await tester.pumpWidget(const MyApp());

    // Verify that our counter starts at 0.
    expect(find.text('0'), findsOneWidget);

```



```

expect(find.text('1'), findsNothing);

// Tap the '+' icon and trigger a frame.
await tester.tap(find.byIcon(Icons.add));
await tester.pump();

// Verify that our counter has incremented.
expect(find.text('0'), findsNothing);
expect(find.text('1'), findsOneWidget);
});
}

```

## pubspec.yaml

```

name: chatbot
description: A new Flutter project.

# The following line prevents the package from being accidentally published to
# pub.dev using `flutter pub publish`. This is preferred for private packages.
publish_to: 'none' # Remove this line if you wish to publish to pub.dev

# The following defines the version and build number for your application.
# A version number is three numbers separated by dots, like 1.2.43
# followed by an optional build number separated by a +.
# Both the version and the builder number may be overridden in flutter
# build by specifying --build-name and --build-number, respectively.
# In Android, build-name is used as versionName while build-number used as
# versionCode.
# Read more about Android versioning at
https://developer.android.com/studio/publish/versioning
# In iOS, build-name is used as CFBundleShortVersionString while build-number
# is used as CFBundleVersion.
# Read more about iOS versioning at
#
https://developer.apple.com/library/archive/documentation/General/Reference/InfoPlistKeyReference/Articles/CoreFoundationKeys.html
# In Windows, build-name is used as the major, minor, and patch parts
# of the product and file versions while build-number is used as the build
# suffix.
version: 1.0.0+1

environment:
  sdk: '>=2.18.6 <3.0.0'

# Dependencies specify other packages that your package needs in order to
# work.
# To automatically upgrade your package dependencies to the latest versions

```

```
# consider running `flutter pub upgrade --major-versions`. Alternatively,
# dependencies can be manually updated by changing the version numbers below
to
# the latest version available on pub.dev. To see which dependencies have
newer
# versions available, run `flutter pub outdated`.
dependencies:
  flutter:
    sdk: flutter
  dialog_flowtter:

  # The following adds the Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^1.0.2

dev_dependencies:
  flutter_test:
    sdk: flutter

  # The "flutter_lints" package below contains a set of recommended lints to
  # encourage good coding practices. The lint set provided by the package is
  # activated in the `analysis_options.yaml` file located at the root of your
  # package. See that file for information about deactivating specific lint
  # rules and activating additional ones.
  flutter_lints: ^2.0.0

# For information on the generic Dart part of this file, see the
# following page: https://dart.dev/tools/pub/pubspec

# The following section is specific to Flutter packages.
flutter:

  # The following line ensures that the Material Icons font is
  # included with your application, so that you can use the icons in
  # the material Icons class.
  uses-material-design: true

  # To add assets to your application, add an assets section, like this:
  assets:
    - assets/dialog_flow_auth.json

    # - images/a_dot_ham.jpeg

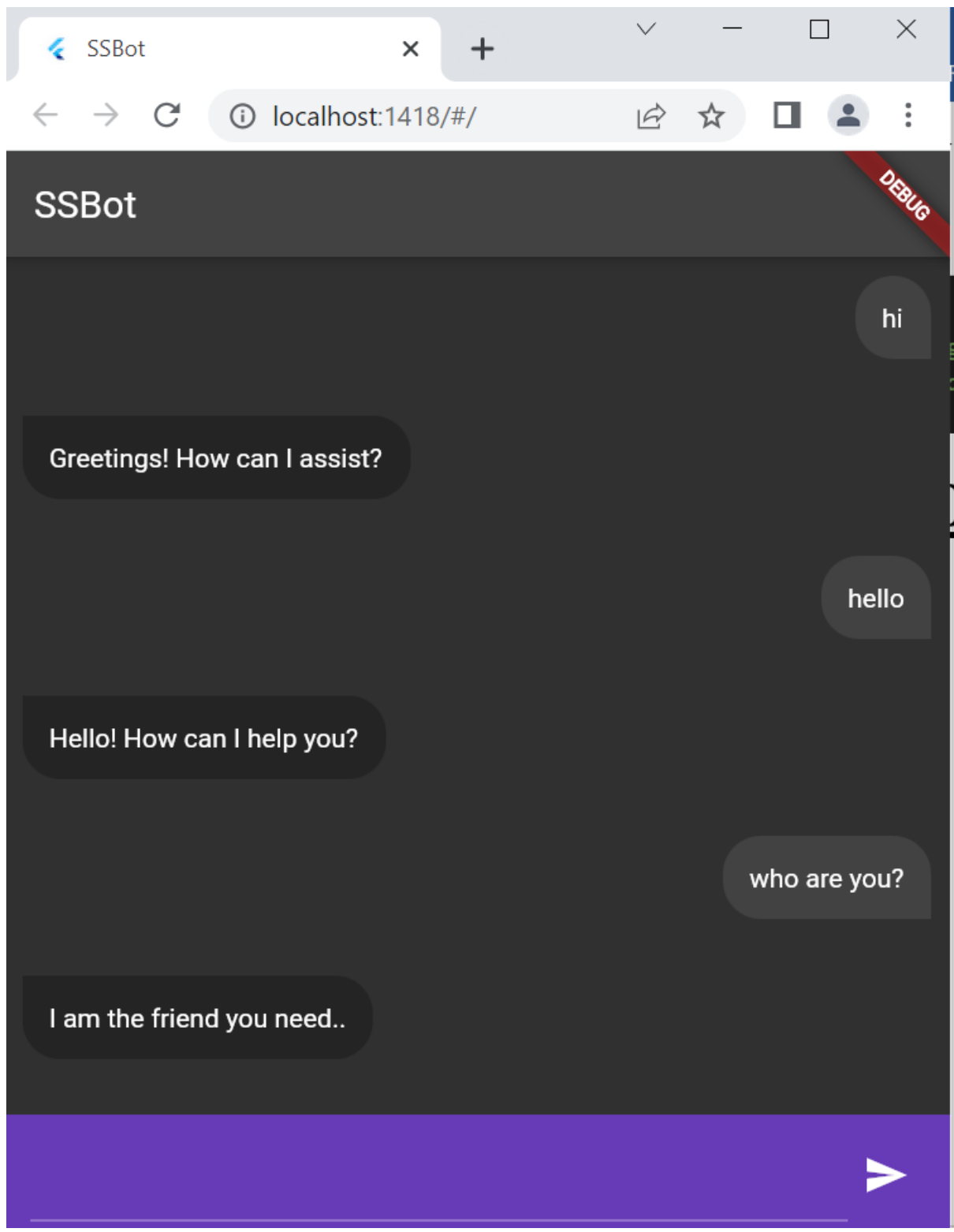
  # An image asset can refer to one or more resolution-specific "variants",
  see
  # https://flutter.dev/assets-and-images/#resolution-aware

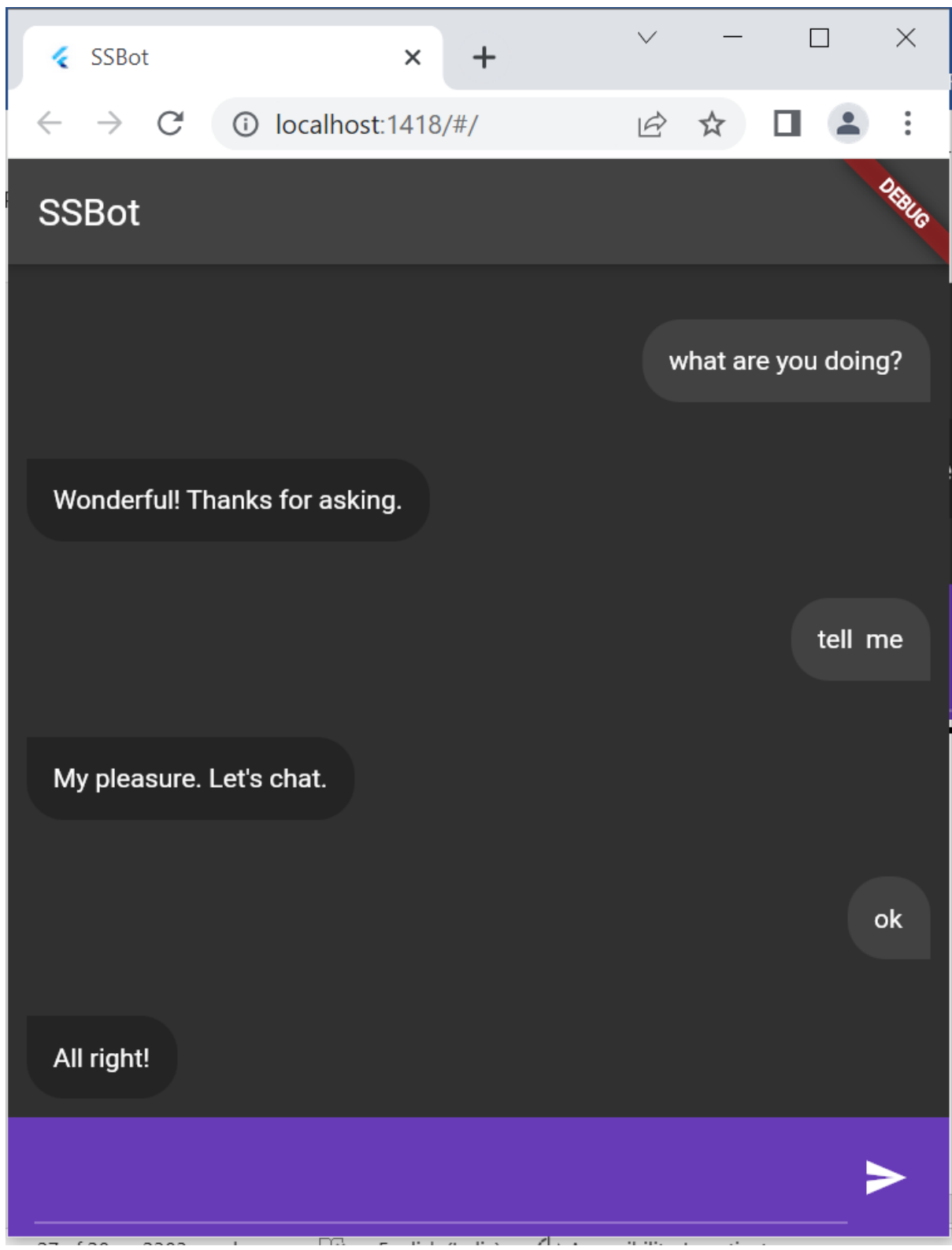
  # For details regarding adding assets from package dependencies, see
```

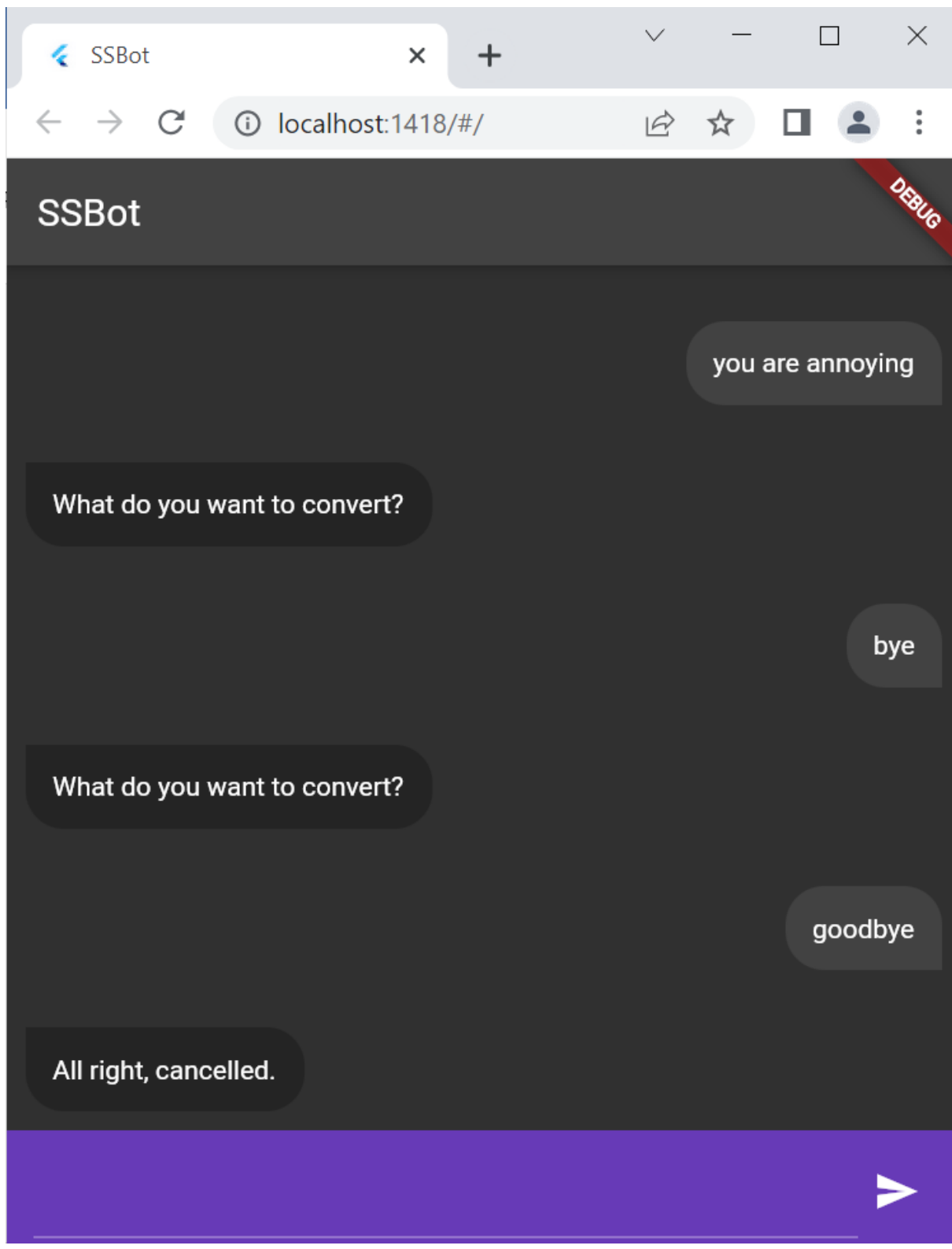
```
# https://flutter.dev/assets-and-images/#from-packages

# To add custom fonts to your application, add a fonts section here,
# in this "flutter" section. Each entry in this list should have a
# "family" key with the font family name, and a "fonts" key with a
# list giving the asset and other descriptors for the font. For
# example:
# fonts:
#   - family: Schyler
#     fonts:
#       - asset: fonts/Schyler-Regular.ttf
#       - asset: fonts/Schyler-Italic.ttf
#         style: italic
#   - family: Trajan Pro
#     fonts:
#       - asset: fonts/TrajanPro.ttf
#       - asset: fonts/TrajanPro_Bold.ttf
#         weight: 700
#
# For details regarding fonts from package dependencies,
# see https://flutter.dev/custom-fonts/#from-packages
```

## SCREENSHOTS







Dialogflow Essentials

Global

Senbot

en

Intents

Entities

Knowledge [beta]

Fulfillment

Integrations

Training

Validation

History

Analytics

Intents

CREATE INTENT

Try it now

Please use test console above to try a sentence.

Search intents

Can you tell me more about Sutapa?

currency-converter

Default Fallback Intent

Default Welcome Intent

help

What are the hobbies of her?

Dialogflow Essentials

Global

Integrations

Training

Validation

History

Analytics

Prebuilt Agents

Small Talk

Docs

Trial Free Upgrade

Dialogflow CX [new]

Small Talk

SAVE

QUESTION

Who are you?

ANSWER

1

I am God

2

Don't you know?

3

I am the friend you need..

4

Enter a Answer variant

QUESTION

How old are you?

ANSWER

1

I am a newborn bot

2

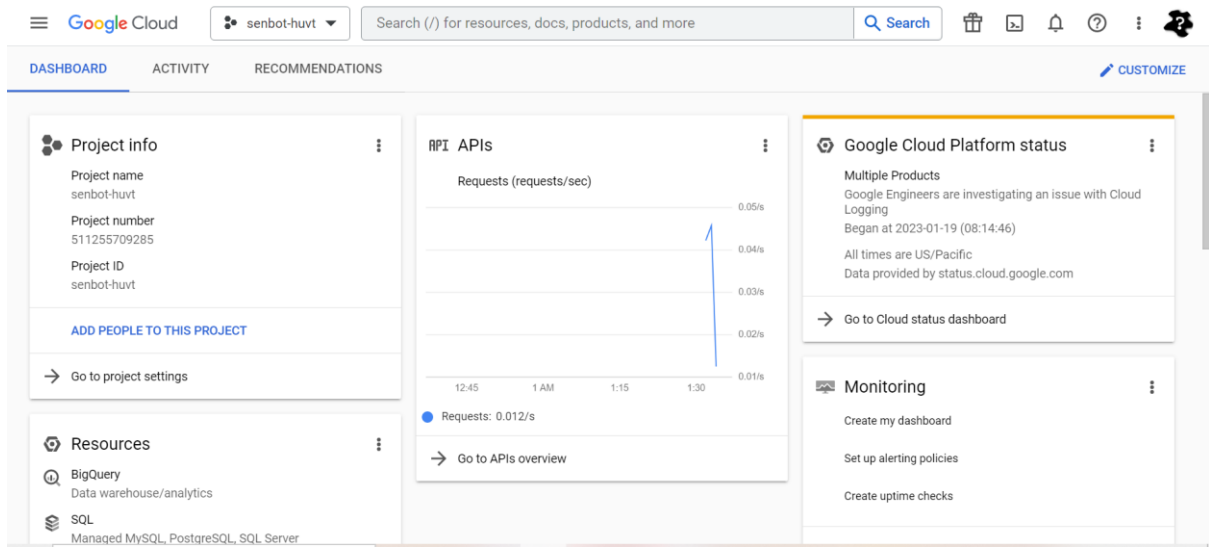
What you have to do?

3

I won't tell you!

4

Enter a Answer variant



The image shows the Visual Studio Code editor with the 'main.dart' file open. The file is part of a Flutter project named 'FLUTTER\_PROJECTS\_MINE'. The code is as follows:

```
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
```

```
chatbot > lib > main.dart > ...
setState(() {
  addMessage(Message(text: DialogText(text: {text})), true);
});

DetectIntentResponse response = await dialogFlowtter.detectIntent(
  queryInput: QueryInput(text: TextInput(text: text));
if (response.message == null) return;
setState(() {
  addMessage(response.message!);
});
}

addMessage(Message message, [bool isUserMessage = false]) {
  messages.add({'message': message, 'isUserMessage': isUserMessage});
}
}
```



## **CONCLUSION**

There is one solution primed to satisfy the modern customer, and that is a [chatbot](#). With a chatbot, any organization can easily offer high-quality support and conflict resolution any time of day, and for a large quantity of customers simultaneously.

According to [Microsoft](#), **90%** of consumers expect an online portal for customer service. As a significant aspect of business evolution, the need for [AI-powered](#) chatbots will only continue to rise.

**THANK YOU!!**