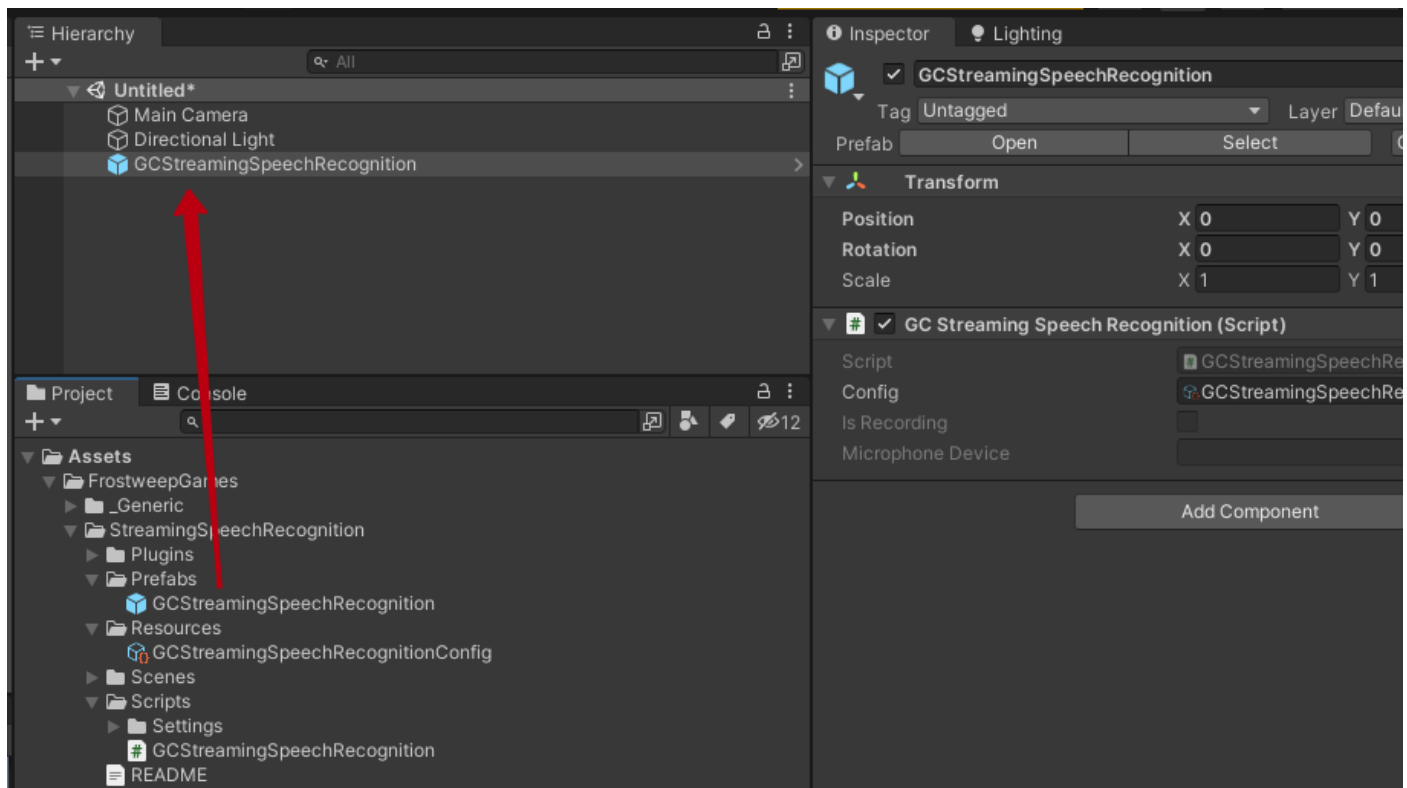


Google Cloud Streaming Speech Recognition

How to use

Lets create a new scene.

Drag and drop GCStreamingSpeechRecognition prefab from Prefabs folder of our asset.



Now you can use API of an asset.

Create a Canvas and add few **buttons**: StartRecord, StopRecord, RefreshMicrophones.

Add an **InputField** for the Context phrases. And also add two **Dropdowns** for selecting language and microphone device.

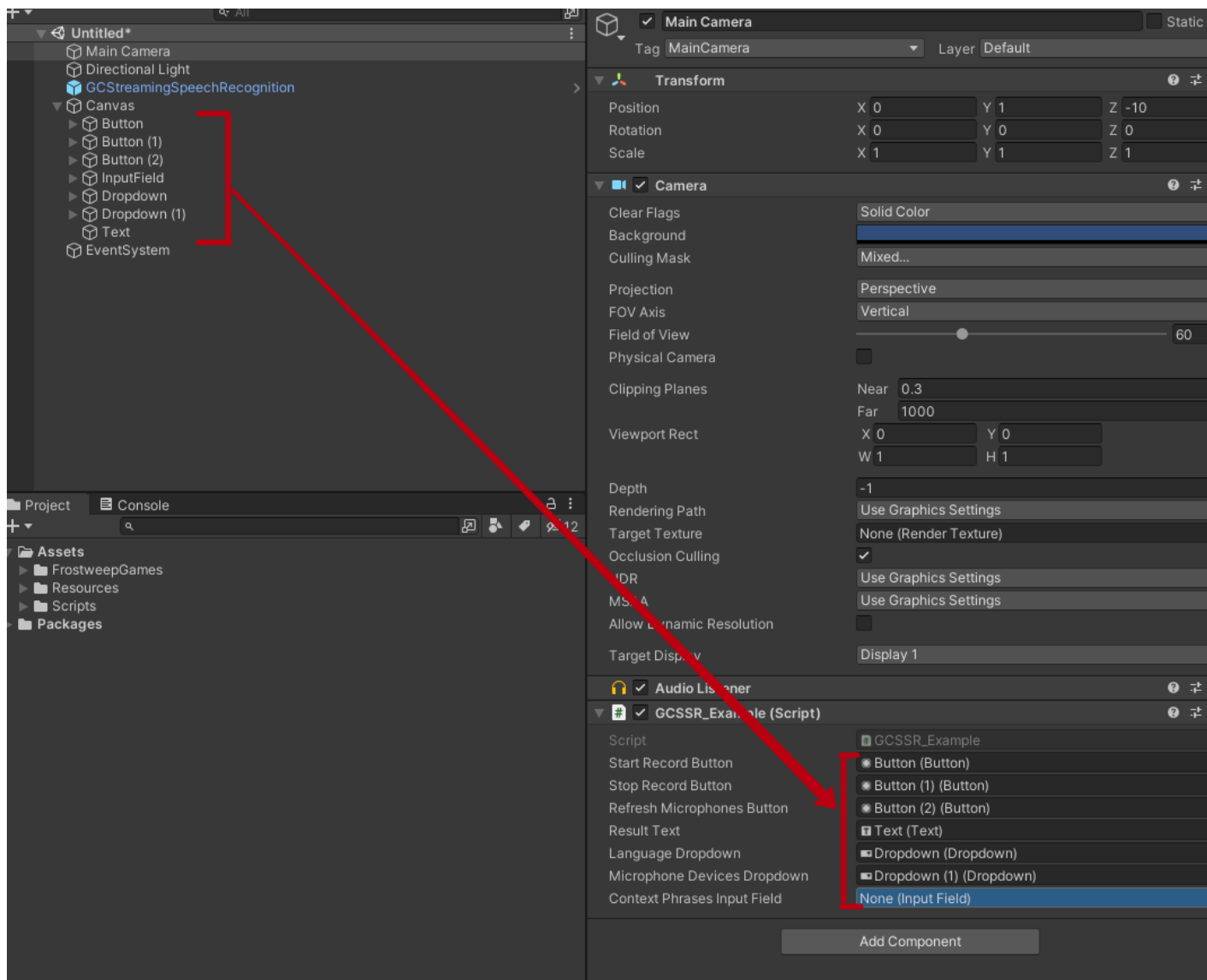
Then add a **Text** to show the results.

Save the scene.

First of all we have to create the variables for our UI elements and then we will make handlers for their events.

```
GCSSR_Example.cs
Assembly-CSharp
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.UI;
5
6  public class GCSSR_Example : MonoBehaviour
7  {
8      public Button startRecordButton,
9      public Button stopRecordButton,
10     public Button refreshMicrophonesButton;
11
12     public Text resultText;
13
14     public Dropdown languageDropdown,
15     public Dropdown microphoneDevicesDropdown;
16
17     public InputField contextPhrasesInputField;
18
19
20
21     private void Start()
22     {
23     }
24 }
```

Now we could connect these variables with UI elements in the scene. Save this script and back to the Unity Editor and drag & drop UI objects from scene into Inspector window of our script.



Save the scene.

Back to the code editor and write handlers for UI and make logic of control of streaming speech recognition.

First of all lets a handlers for buttons and subscribe on onClick actions:

```
Сообщение Unity | Ссылки: 0
private void Start()
{
    startRecordButton.onClick.AddListener(StartRecord);
    stopRecordButton.onClick.AddListener(StopRecord);
    refreshMicrophonesButton.onClick.AddListener(RefreshMicrophones);
}

ССылка: 1
public void StartRecord()
{
}

ССылка: 1
public void StopRecord()
{
}

ССылка: 1
public void RefreshMicrophones()
{
}
```

Soon we will write logic inside them but now lets create similar code for dropdowns:

```
Сообщение Unity | Ссылки: 0
private void Start()
{
    startRecordButton.onClick.AddListener(StartRecord);
    stopRecordButton.onClick.AddListener(StopRecord);
    refreshMicrophonesButton.onClick.AddListener(RefreshMicrophones);

    languageDropdown.onValueChanged.AddListener(LanguageChanged);
    microphoneDevicesDropdown.onValueChanged.AddListener(MicrophoneChanged);
}

ССылка: 1
public void LanguageChanged(int value)
{
}

ССылка: 1
public void MicrophoneChanged(int value)
{
}

ССылка: 1
public void StartRecord()
{
}
```

Now we are ready for connection an API to the example script.

Lets add a namespace with name: *FrostweepGames.Plugins.GoogleCloud.StreamingSpeechRecognition*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using FrostweepGames.Plugins.GoogleCloud.StreamingSpeechRecognition;

Сообщение Unity | Ссылки: 0
public class GCSSR_Example : MonoBehaviour
{
}
```

By doing this we enable an access to the streaming speech recognition API.

So, create a new variable with type *GCStreamingSpeechRecognition* and name *_speechRecognition* and get an instance of an API script like showed in screenshot:

```

@ Скрипт Unity | Ссылки: 0
public class GCSSR_Example : MonoBehaviour
{
    private GCStreamingSpeechRecognition _speechRecognition;

    public Button startRecordButton,
    > + + + stopRecordButton,
    > + + + refreshMicrophonesButton;

    public Text resultText;

    public Dropdown languageDropdown,
    > + + + microphoneDevicesDropdown;

    public InputField contextPhrasesInputField;

    @ Сообщение Unity | Ссылки: 0
    private void Start()
    {
        _speechRecognition = GCStreamingSpeechRecognition.Instance;
    }
}

```

With that variable you are able to access an API of speech recognition. But first let's subscribe on important events of speech recognition system. For that we have to create new function (handlers) and subscribe on events in Start function:

```

private void Start()
{
    _speechRecognition = GCStreamingSpeechRecognition.Instance;

    _speechRecognition.StreamingRecognitionStartedEvent += StreamingRecognitionStartedEventHandler;
    _speechRecognition.StreamingRecognitionFailedEvent += StreamingRecognitionFailedEventHandler;
    _speechRecognition.StreamingRecognitionEndedEvent += StreamingRecognitionEndedEventHandler;
    _speechRecognition.InterimResultDetectedEvent += InterimResultDetectedEventHandler;
    _speechRecognition.FinalResultDetectedEvent += FinalResultDetectedEventHandler;

    startRecordButton.onClick.AddListener(StartRecord);
    stopRecordButton.onClick.AddListener(StopRecord);
    refreshMicrophonesButton.onClick.AddListener(RefreshMicrophones);

    languageDropdown.onValueChanged.AddListener(LanguageChanged);
    microphoneDevicesDropdown.onValueChanged.AddListener(MicrophoneChanged);
}

ссылка: 1
private void StreamingRecognitionStartedEventHandler()
{
}

ссылка: 1
private void StreamingRecognitionFailedEventHandler(string error)
{
}

ссылка: 1
private void StreamingRecognitionEndedEventHandler()
{
}

ссылка: 1
private void InterimResultDetectedEventHandler(string alternative)
{
}

ссылка: 1
private void FinalResultDetectedEventHandler(string alternative)
{
}

```

By doing that we will have information about recognized speech and when recognition is started and finished and also handle errors.

These are a description of events you're subscribed on:

- **StreamingRecognitionStartedEvent** – fire when streaming recognition successfully started
- **StreamingRecognitionFailedEvent** – fire when streaming recognition receives an errors such as: Forbidden, Limit reached, general network errors, etc.
- **StreamingRecognitionEndedEvent** – fire when streaming recognition finished its work
- **InterimResultDetectedEvent** – fire when intermediate result has received. Fire only when it is allowed in config
- **FinalResultDetectedEvent** – fire when final result of recognized speech is received.

Now lets create a logic for our UI elements and call the API of plugin.

First of all, we have to fill list of languages into our dropdown. For doing that we have to get list of names of language codes from enumerator and then add them to list (you could clear options before add new ones):

```
private void Start()
{
    languageDropdown.ClearOptions();
    for (int i = 0; i < System.Enum.GetNames(typeof(Enumerators.LanguageCode)).Length; i++)
    {
        languageDropdown.options.Add(new Dropdown.OptionData(((Enumerators.LanguageCode)i).Parse()));
    }
}
```

Similar logic you have to do in RefreshMicrophones button handler to have list of available devices:

```
private void RefreshMicsButtonOnClickHandler()
{
    _speechRecognition.RequestMicrophonePermission();
    _microphoneDevicesDropdown.ClearOptions();
    for (int i = 0; i < _speechRecognition.GetMicrophoneDevices().Length; i++)
    {
        _microphoneDevicesDropdown.options.Add(new Dropdown.OptionData(_speechRecognition.GetMicrophoneDevices()[i]));
    }
}
```

Here you could see few new API function which we used: **RequestMicrophonePermission** and **GetMicrophoneDevices** which helps us to get access to a list of devices and get list of devices at all.

These are description of functions:

- **GetMicrophoneDevices** – returns array of connected microphone devices names. Empty, if no devices
- **RequestMicrophonePermission** – prompt user for granting access to microphone devices

So, now we have filled list of languages and microphone devices now we are ready to use that.

When microphone device is selected or changed, we have to handle that and say to plugin that device was changed. For doing that we have to back to microphone devices dropdown handle and write logic for telling API to change device:

```
public void MicrophoneChanged(int value)
{
    if (!_speechRecognition.HasConnectedMicrophoneDevices())
        return;
    _speechRecognition.SetMicrophoneDevice(_speechRecognition.GetMicrophoneDevices()[value]);
}
```

Here you could two new functions and one old which is described above.

These are description of functions:

- **HasConnectedMicrophoneDevices** – returns true if at least one device is connected and ready to use
- **SetMicrophoneDevice** – configures plugin to use microphone device with name provided (be careful and don't provide custom names. Use only ones which is available in devices list from GetMicrophoneDevices)

Once you are configured microphone device you are able to run speech recognition API. For doing that you have to call **StartStreamingRecognition** function from API. It takes two parameters: language code and speech context. Lets add it to StartRecord button handler:

```
public void StartRecord()
{
    _speechRecognition.StartStreamingRecognition((Enumerators.LanguageCode)languageDropdown.value, null);
}
```

In this screenshot you could see that we took value from language dropdown and converted it to *LanguageCode* (this is possible to do because dropdown was filled from elements of *LanguageCode* enumerator and both starts from zero index). Also, we don't provide context there so we have to simply write **null** in second parameter.

These are description of functions:

- **StartStreamingRecognition** – starts streaming recognition. Takes two parameters: *LanguageCode*, *Context*

So, we had started streaming recognition, talking something in microphone and waiting for recognition, but we don't see any results... yes, we have to show them by writing logic in recognition API handlers. Find these functions and add results to our UI Text component:

```
private void InterimResultDetectedEventHandler(string alternative)
{
    resultText.text += $"<b>Alternative:</b> {alternative}\n";
}

ссылка:1
private void FinalResultDetectedEventHandler(string alternative)
{
    resultText.text += $"<b>Final:</b> {alternative}\n";
}
```

We just simply add received text to our component.

Okay, we see results, but how to stop recognition? – For that we have special function in API which stops recording from microphone and stops recognition. Find StopRecording button handler and write new logic like in screenshot:

```
public async void StopRecord()
{
    await _speechRecognition.StopStreamingRecognition();
}
```

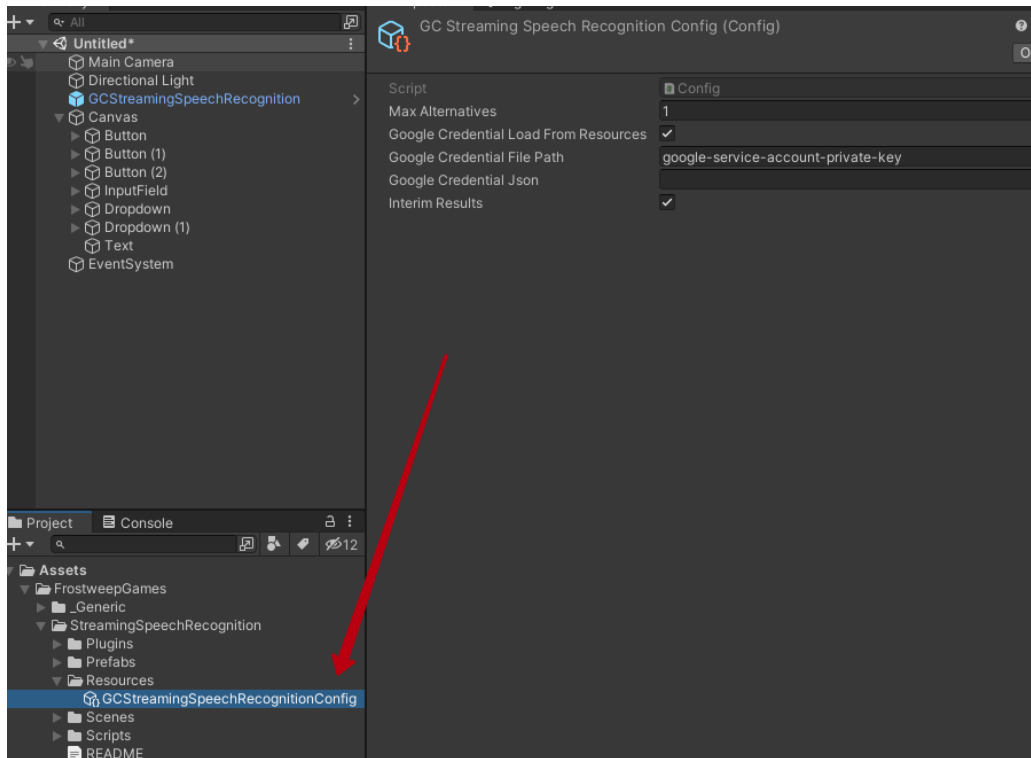
This is a bit harder construction. But don't worry we just do that asynchronously. **async** keyword says to compiler that this function will run asynchronously and **await** keyword says to compiler that this function should wait till **StopStreamingRecognition** function will end their work before end its own flow. Of course, it could be called in sync mode but it will cause main thread freeze, so **we recommend to use it always in async mode**.

These are description of functions:

- **StopStreamingRecognition** – stops streaming recognition by asynchronously finishing their active tasks. Recommended to use always asynchronously by using async-await construction.

Okay, now save script and back to the Unity Editor as because we are finished with programming and ready to finish configuration of plugin before using it.

Our plugin uses own config file with parameters:



It helps to configure open parameters.

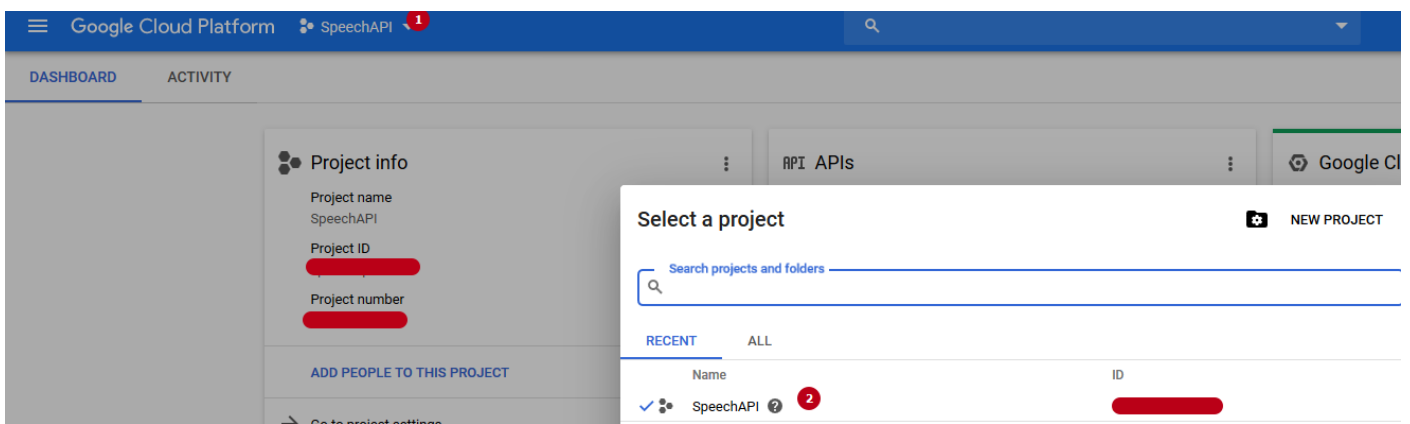
These are description of parameters:

- **Max Alternatives** – number of alternative results possible from Google Service during recognition. Up to 10
- **Google Credential Load From Resources** – says to plugin from where it has to load Json data of Google Service account (Json: <https://www.json.org>) (how to get service account data described below)
- **Google Credential File Path** – path to file in *Resources* folder in a project used if previous parameter set to true. (You don't need to write absolute path, use relative under *Resources* folder. Also don't need to write extension of a file.)
- **Google Credential Json** – if you prefer to don't use file from resources you could use string representation. To enable using of it you have to set **Google Credential Load From Resources** parameter to false and then insert Json data in this input field
- **Interim Results** – enables interim results during recognition instead of receiving only final.

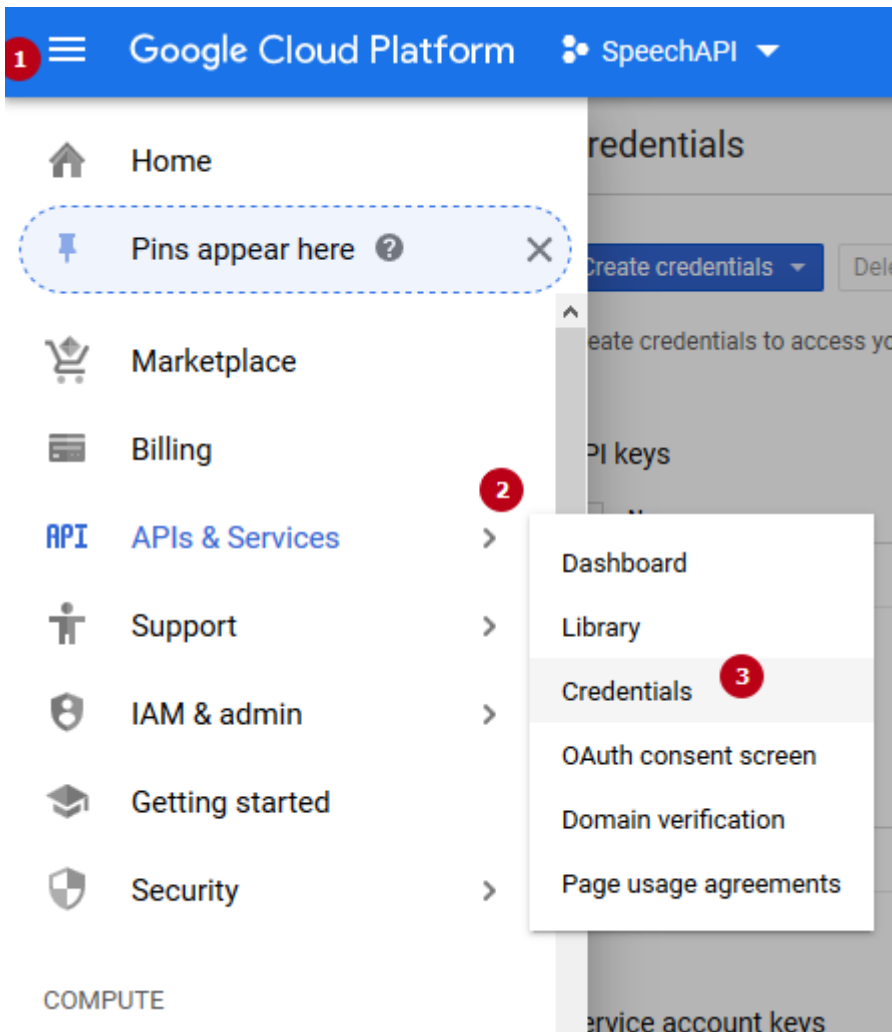
Now you are almost finished. You just need to configure project in Google Cloud Console and create a Service Account (to receive access key file).

Lets move to the **Google Service** and setup dashboard:

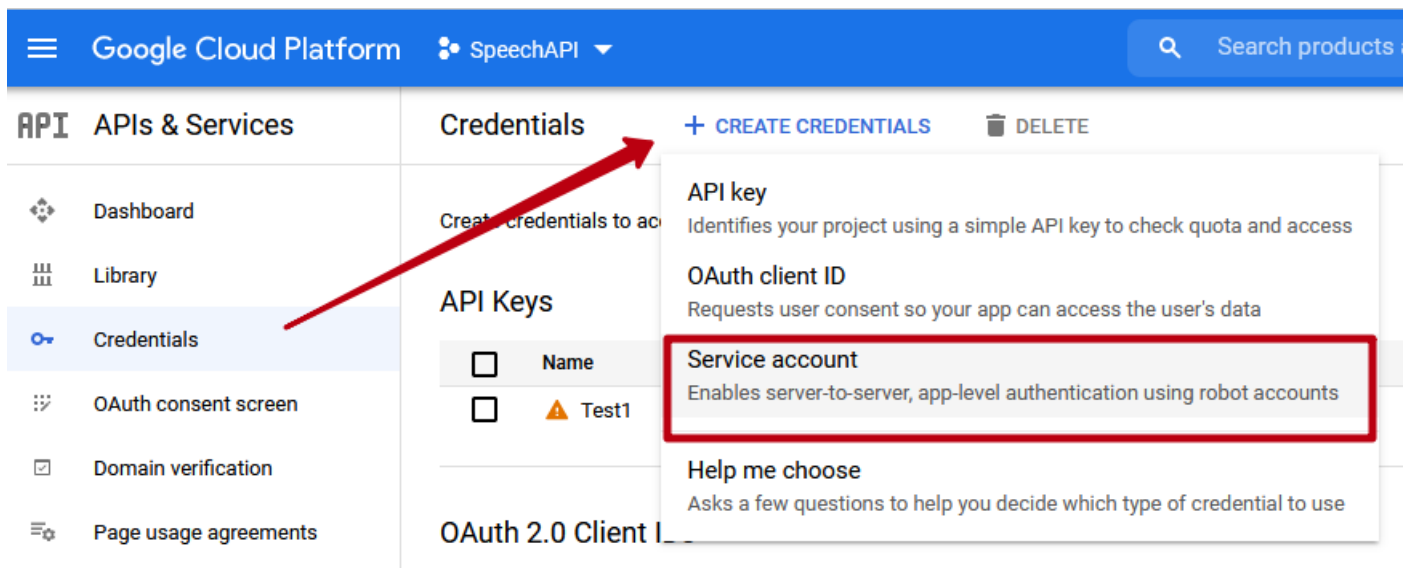
Open the <https://console.cloud.google.com/> create a first project (or use already created) and select it.



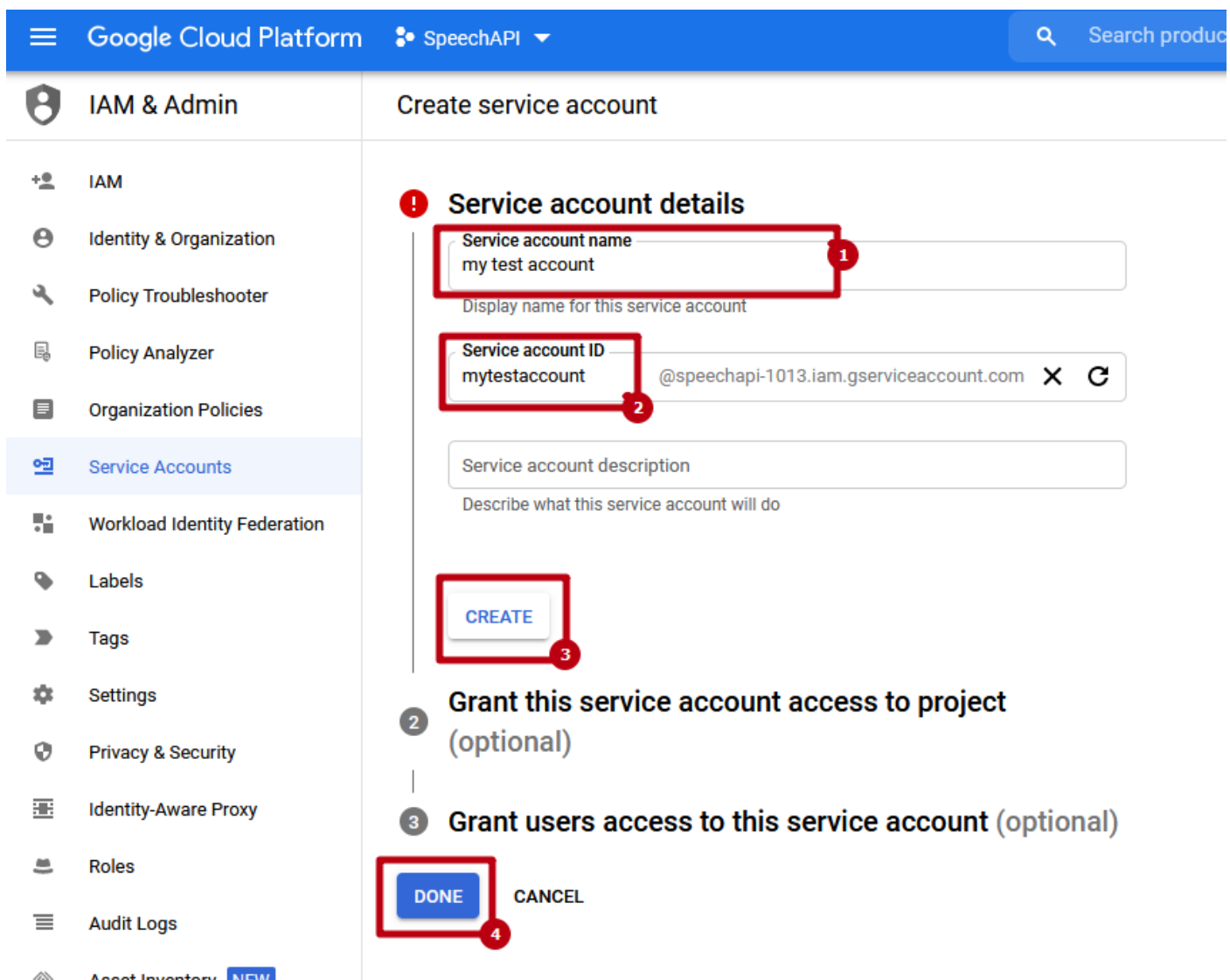
Then select **Menu**, then **Api & Services**, then **Credentials**



After that in opened screen **select Create credentials** and select **API Key**




In opened screen you have to fill fields and create a new service account (you could also grant accesses to you project. For test purposes you could set it to **Owner** role):



✓ Service account details

2 Grant this service account access to project (optional)

Grant this service account access to SpeechAPI so that it has permission to complete specific actions on the resources in your project. [Learn more](#)

<div>Role</div> <div>Owner</div> <div>Full access to all resources.</div>	<div>Condition</div> <div>Add condition</div>	
---	---	---

+ ADD ANOTHER ROLE

CONTINUE

3 Grant users access to this service account (optional)

DONE

CANCEL

Once account was created you will be redirected to Credentials page. There you will see your newly created service account. Tap on pen button to open settings:

Google Cloud Platform

SpeechAPI

Search products and res

IAM & Admin

my test account

DETAILS PERMISSIONS KEYS METRICS LOGS

Service account details

Name

my test account

SAVE

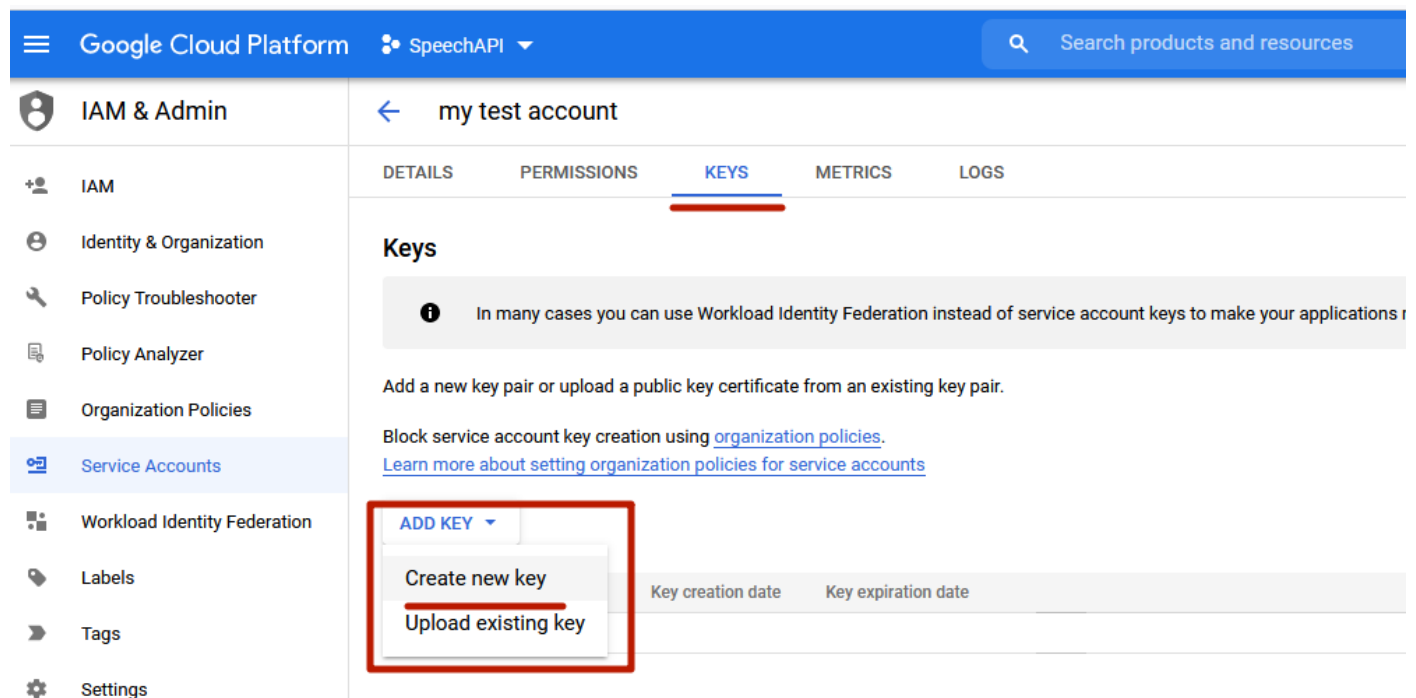
Description

SAVE

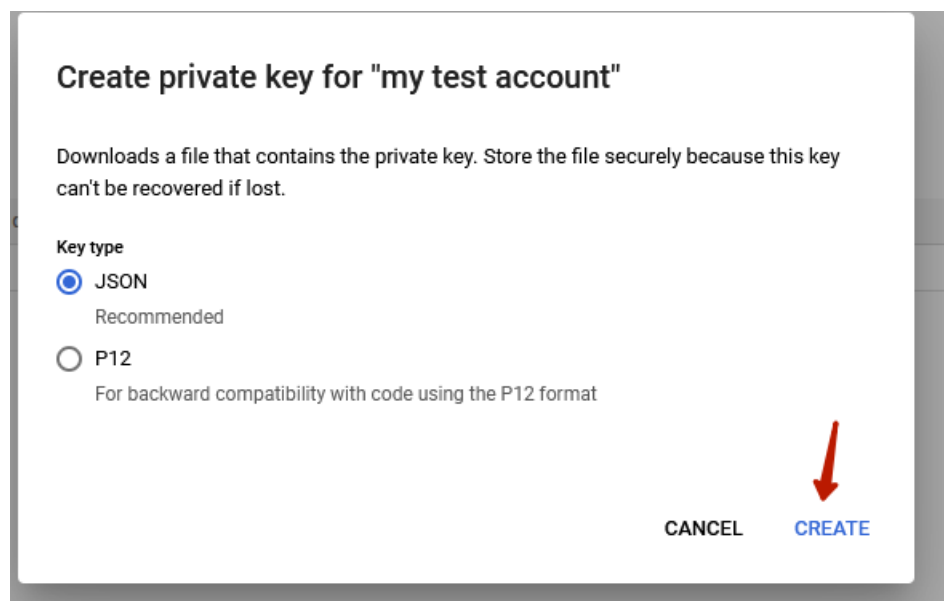
Email

mytestaccount@speechapi-1013.iam.gserviceaccount.com

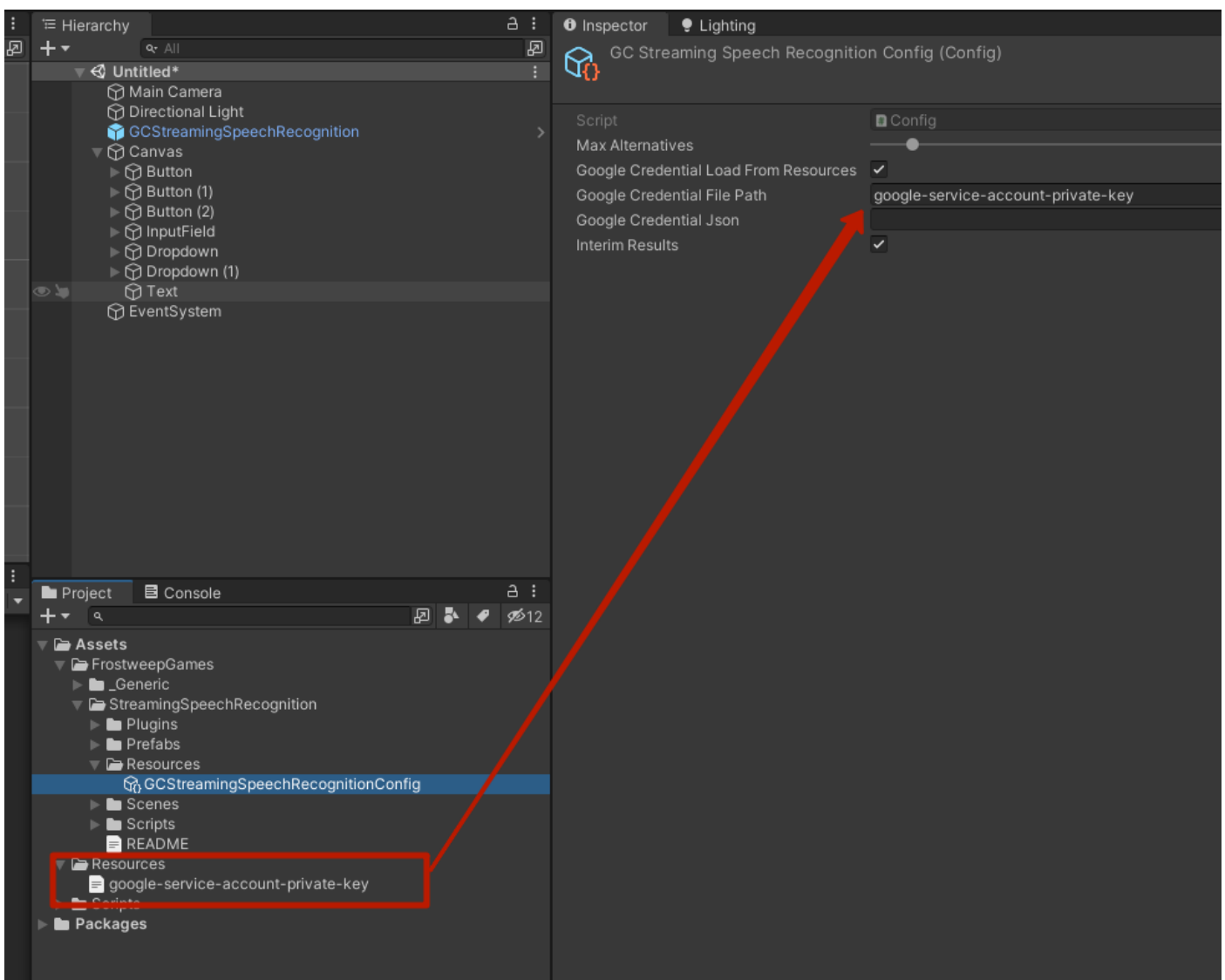
There you have to open KEYS tab and create a new Key file:



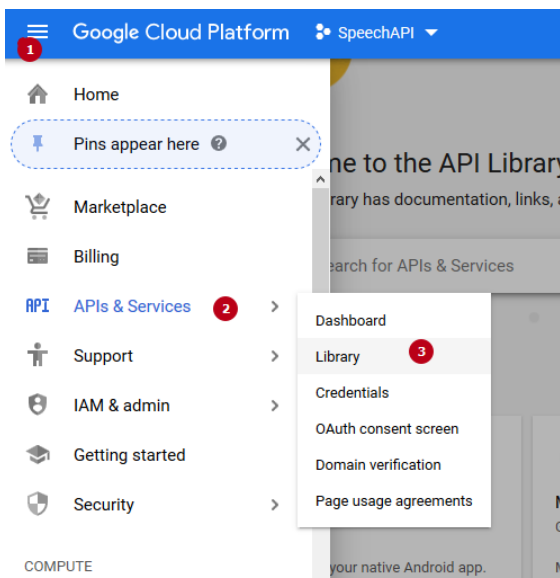
In opened window select JSON and click on Create button:

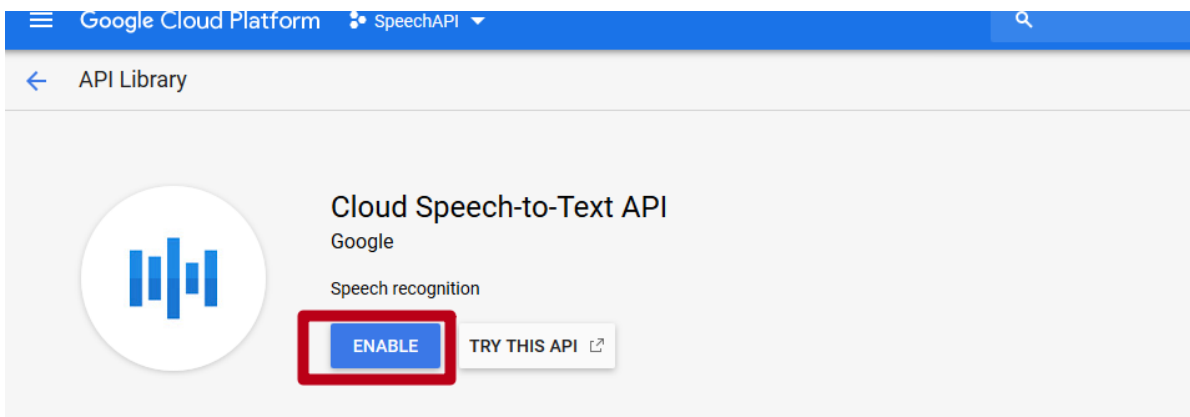
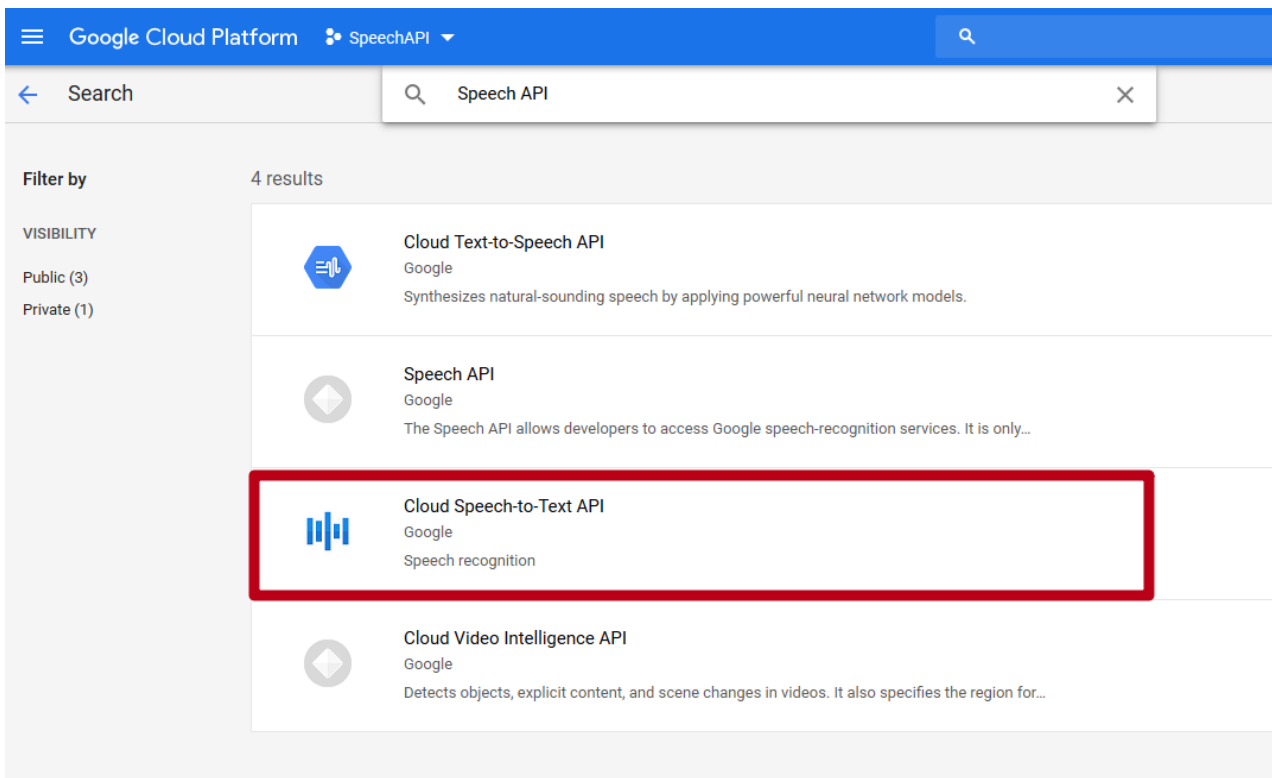


You will be prompted to load newly create key file. So, download it and copy it to your project folder somewhere in **Resources** folder (if you don't have this folder, create a new one and copy file there). Then copy file name and paste it in Config file parameter input field(described above)

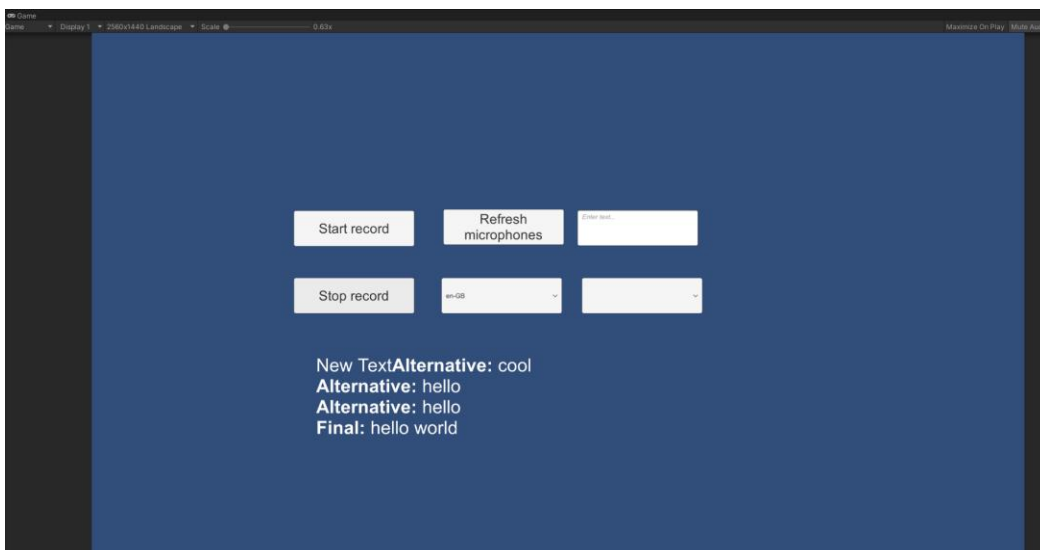


After that back to the google console and open APIs and Services page to enable Speech to Text API:





Now you ready to use plugin. Open back Unity Editor and click on Play button. Then click on refresh microphones button, select microphone device from list and click on Start Record button. To stop recognition – click on Stop Record button:



Troubleshooting

Unity Editor

- Multiple plugins with the same name 'Newtonsoft.Json' found in a project. – to fix that issue you have to delete same library from you project and use only one. Our plugin uses Newtonsoft.Json version special for Unity from: <https://github.com/jilleJr/Newtonsoft.Json-for-Unity/releases/> . For other libraries which are included in our plugin should be included special release version of Newtonsoft.Json to prevent errors and linking. Latest uses version described under Plugin folder in *dependencies.manifest* file.
 - This issue could appear in new version of Unity where added Collaborate package which is also uses Newtonsoft.Json library. To prevent build errors, you have to delete Collab package via Package Manager (editor can be opened from *Window/Package Manager* menu). Issue related thread: <https://forum.unity.com/threads/conflict-between-assets-newtonsoft-and-packagemanagers-newtonsoft.843898/#post-7194568>

iOS

Our plugin has own iOS Post Process handler which adds needed libraries and set Build Settings in *Unity-iPhone.xcodeproj* to work and build project properly. However, in project with custom settings you could receive errors. Below we will show possible issues and how to fix them.

- *Libraries/FrostweepGames/StreamingSpeechRecognition/Plugins/Grpc.Core/runtimes/ios/libgrpc_csharp_ext.a(grpc_csharp_ext.o)' does not contain bitcode. You must rebuild it with bitcode enabled (Xcode setting ENABLE_BITCODE), obtain an updated library from the vendor, or disable bitcode for this target. for architecture arm64* - To fix that you have to **disable Bitcode** in all targets and project.
- *Undefined symbols for architecture arm64: "_inflateInit2_", referenced from: zlib_compress(grpc_slice_buffer*, grpc_slice_buffer*, int) in libgrpc.a(message_compress.o) grpc_stream_compression_context_create_gzip(grpc_stream_compression_method) in libgrpc.a(stream_compression_gzip.o)* - To fix that you have to add **libz.tbd** to **UnityFramework** target.

macOS

If plugin doesn't work there, please check that you build you project only with **Intel64** architecture. Our **plugin doesn't support Apple Silicon** CPU yet so you should remove it from supported architecture.

Version Updates

- 1.0
 - First Release