

Siratee K.

212 Followers

About

Follow

Sign in

Get started



Line Messaging API



Firebase

“LINE Messaging API” x

“Firebase (Cloud Function + Firestore)”



Siratee K. Sep 28, 2019 · 10 min read

Level of this tutorial: Developer

{Intro}

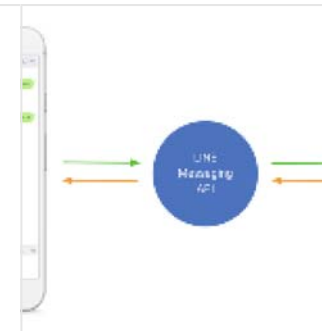
Hi developers!, According to the passed tutorial I talk about “Get started using LINE Messaging API” for beginner and the steps for getting ready before developing so in this tutorial we will start coding the chatbot making it reply message to the user which has **Firebase as a webhook**. so let get start!

[หากต้องการอ่านบทความที่มีเนื้อหาคล้ายๆกับในบทความนี้ และเขียนด้วยภาษาไทย ผมแนะนำบทความนี้เลยครับ เขียนโดย Jirawatee (พีดี)]

สร้าง LINE Bot ด้วย Messaging API และ Cloud Functions for Firebase

ยุคนี้หลายคนคงรู้จัก Bot หรือ Chatbot บนแพลตฟอร์มของ LINE กันแล้ว เพราะทาง LINE ได้เปิด Messaging API...

medium.com



Note: If you don't know what is LINE Messaging API I suggest you read my "Get started using LINE Messaging API" tutorial in the link below

Get started using "LINE Messaging API"

This is the tutorial that will explain about LINE Messaging API from the zero, It is recommended for the developer who want...

medium.com



* What is Firebase?

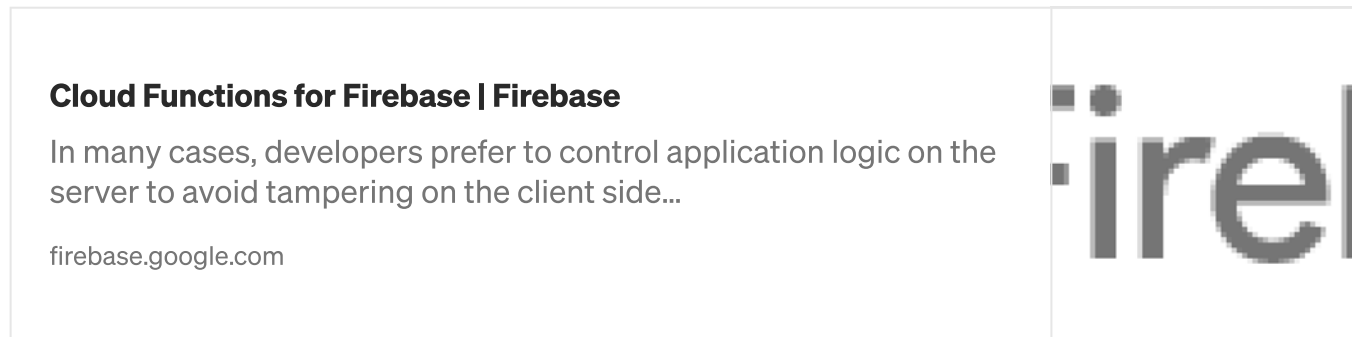


Firebase is a cloud platform run by Google. They have so many services such as CloudFirestore, Authentication, Cloud functions, etc. and it **FREE**

for the starter but if you want more quota for using their service, you can pay for it.

Now, let scope on Cloud Functions for Firebase, This is the service that allows us to create the online function using JavaScript or TypeScript and you can call it from anywhere. We'll use this service for creating a webhook to receive the POST Request from LINE Messaging API.

For more detail:



. . .

{Preparing our project}

Before anything else let talk about our example project goal and features. We will create a chatbot in LINE Messaging API having the cloud function for firebase as a webhook. Our webhook will have the

features to store the chat history and get the customer data from the Firestore Database then reply it back to our user with LINE Messaging API if the user are recognized as our customers. **You are allowed to use the code in this tutorial, I will upload full source code to my GitHub, Link is at the bottom of this tutorial.**

Goal

- We can develop cloud function as a webhook for LINE Messaging API
- We can develop our cloud function to connect to the Firestore Database for storing and querying the data
- We can decode the request sent by LINE Messaging API and get the required data from that request (In this tutorial we need: userId, text, timestamp)

So let get started!

* Create Firebase Project

First, let go to Firebase home page by follow the link below

Firebase

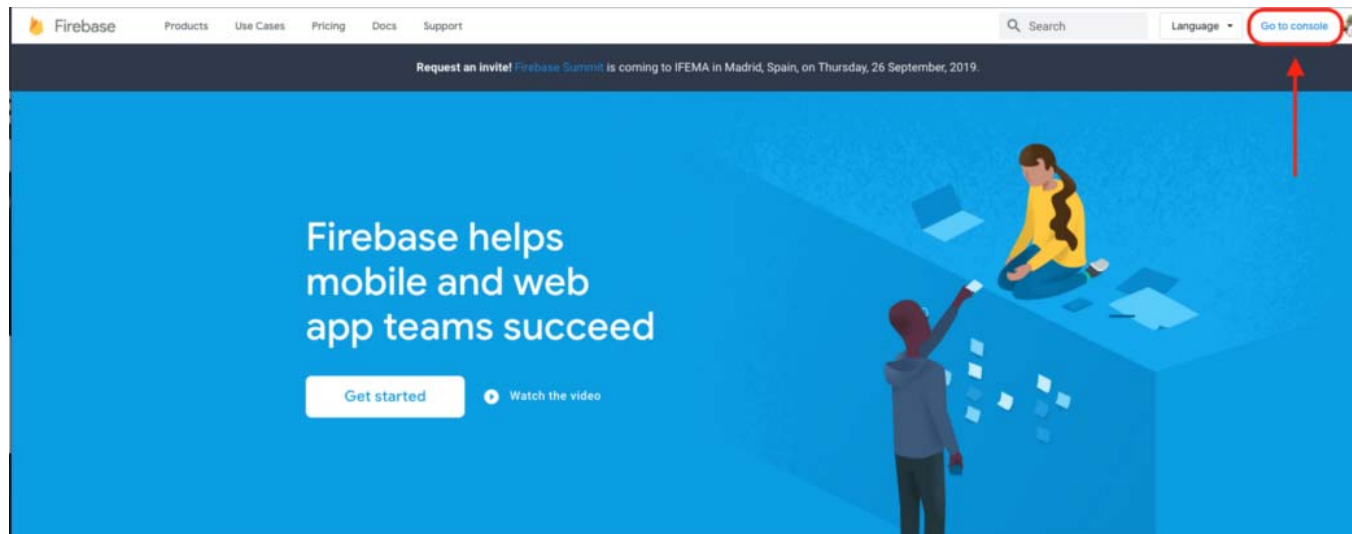
Firebase is Google's mobile platform that helps you quickly develop



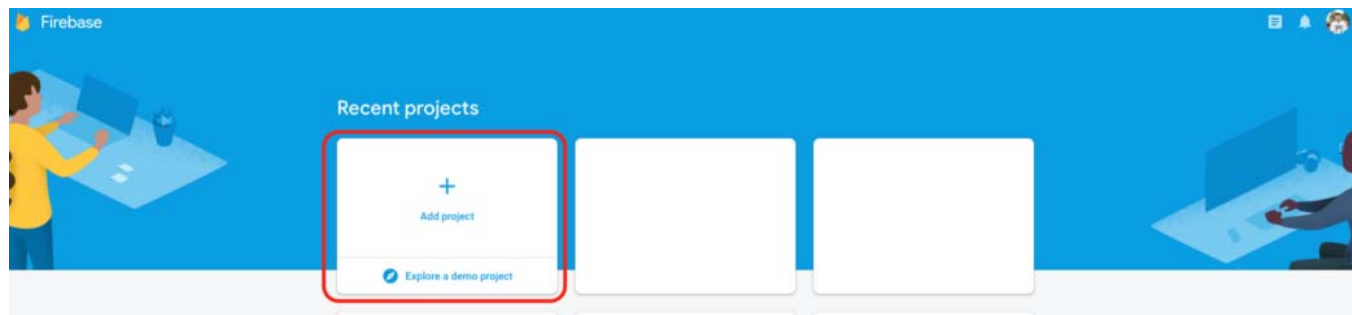
Firebase is Google's mobile platform that helps you quickly develop high-quality apps and grow your business.
firebase.google.com

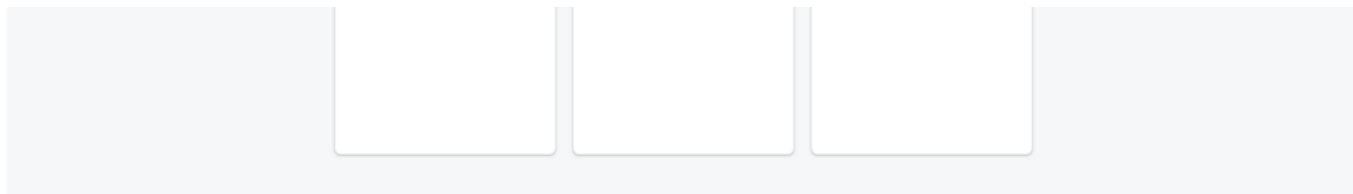


Log in with your Google account and then click on “Go to console” button



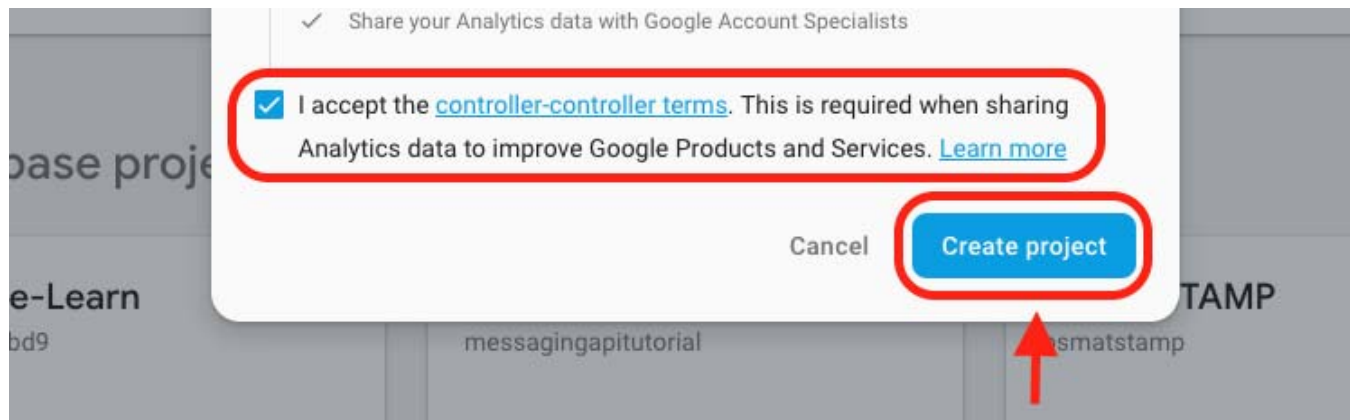
And then click on “Add project” button



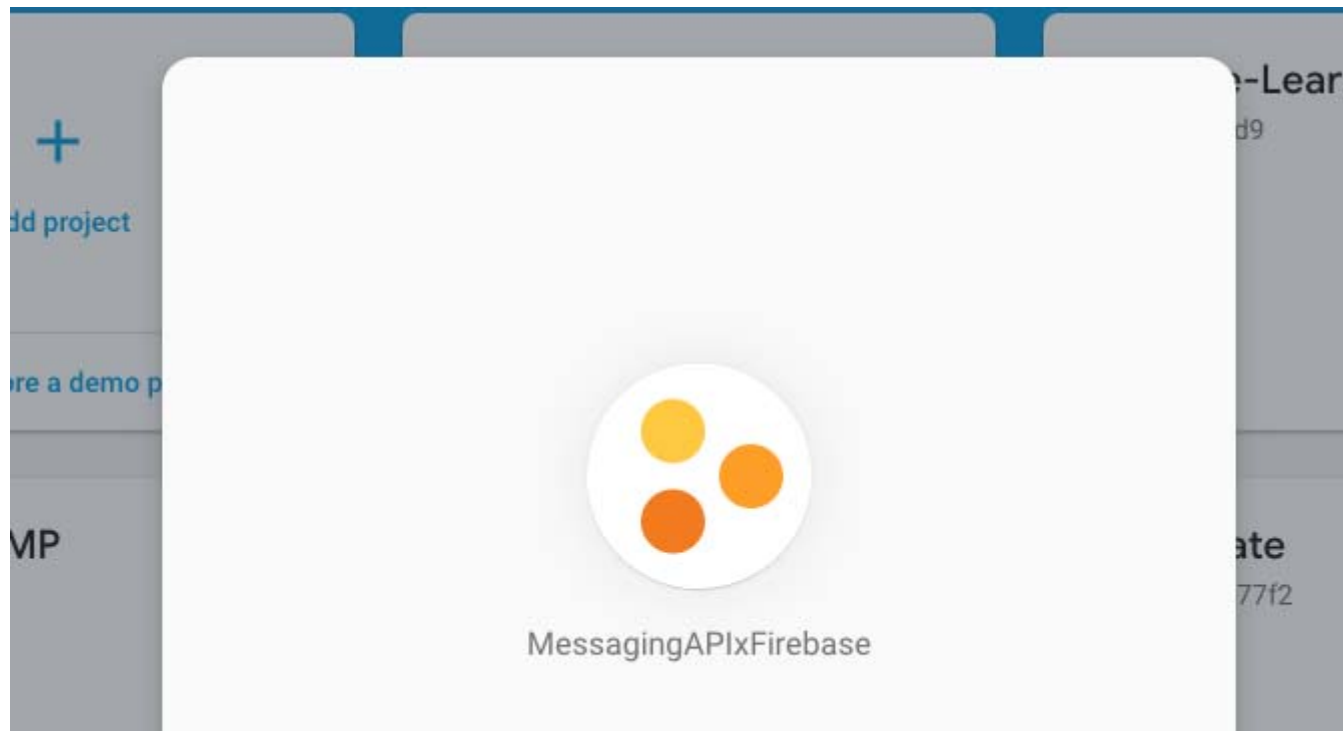


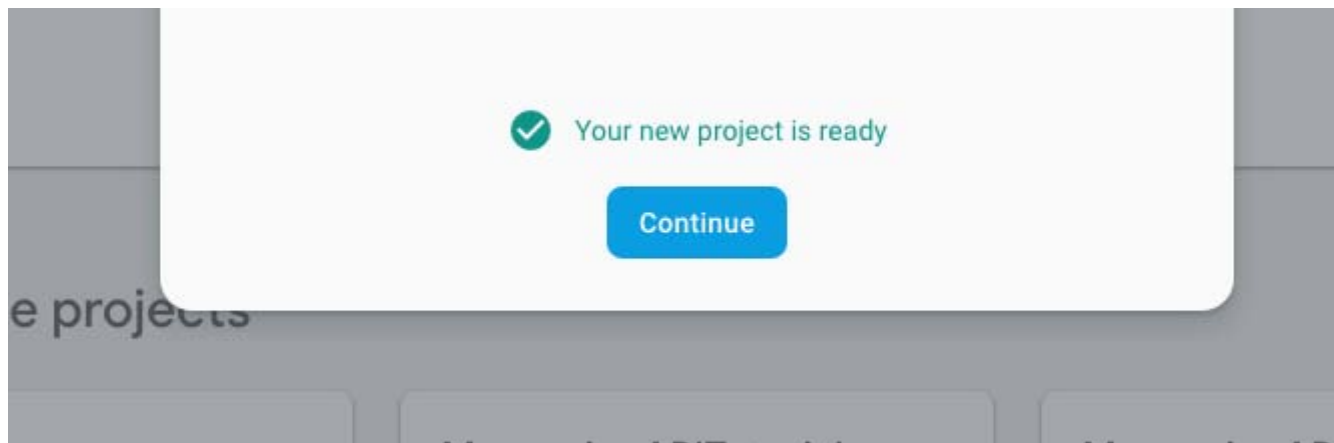
Note: If you have other Firebase Project, it'll appear in this page as in the picture(In the blank block)

Then type your awesome project name and don't forget to check "I accept terms of use" and then click Create Project

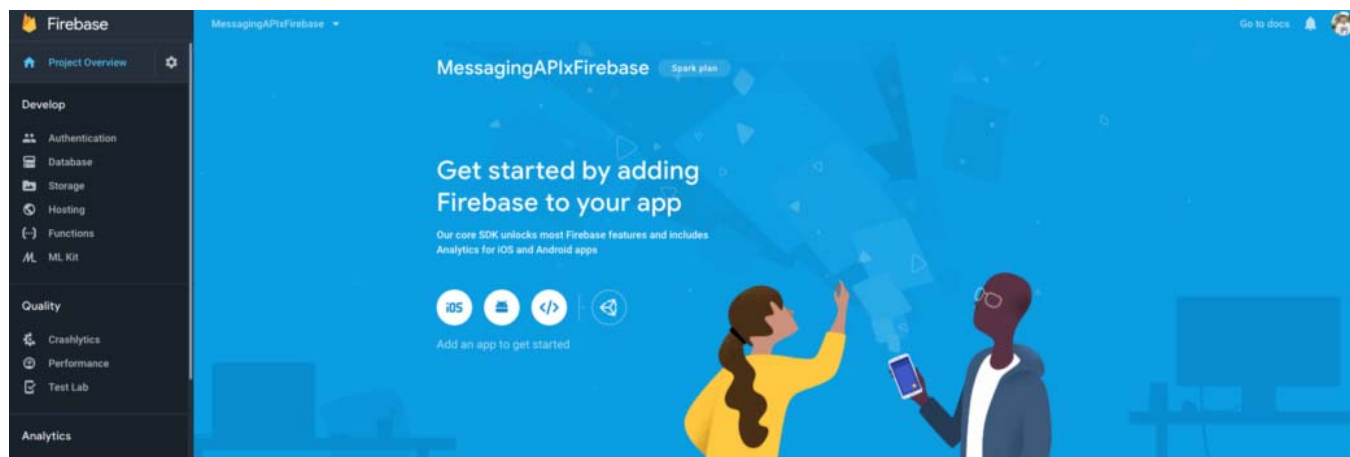


And it gonna take a few seconds for creating your project and you'll see this page when your project has been created, Then click Continue and it will automatically redirect you to your Project Console





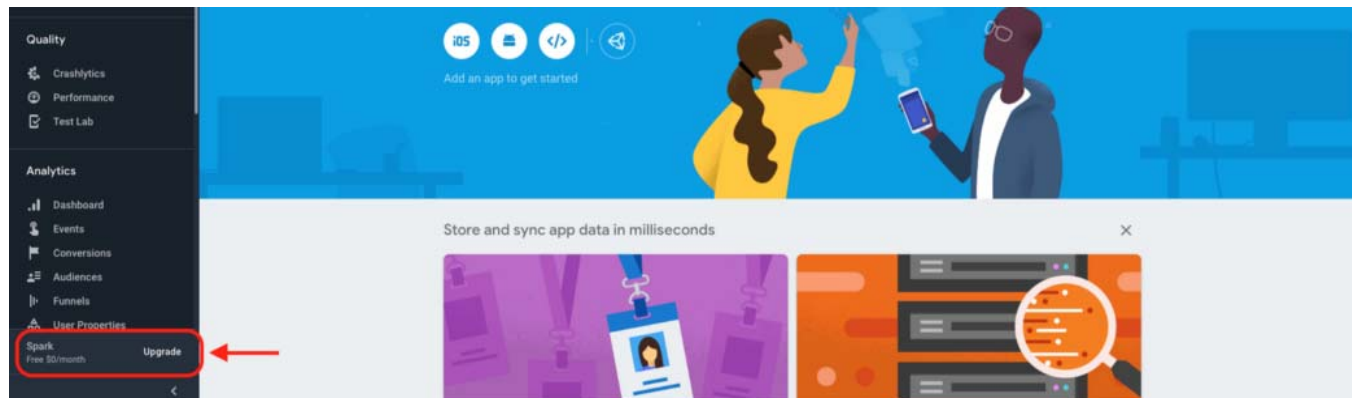
This is your project console



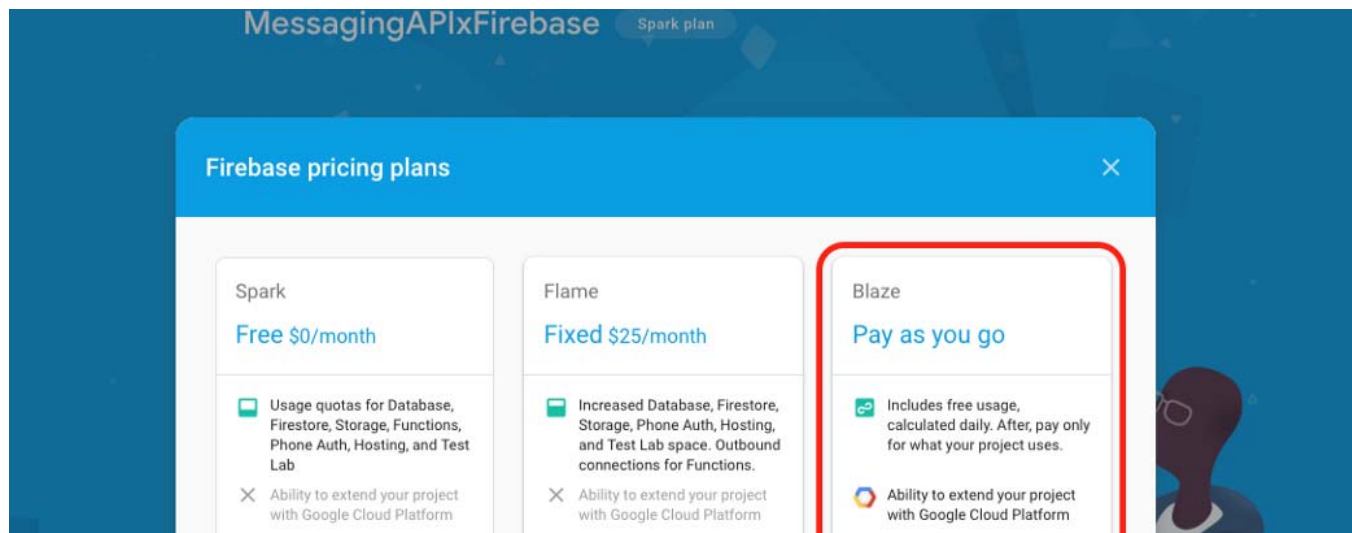
* Changing Firebase pricing plan

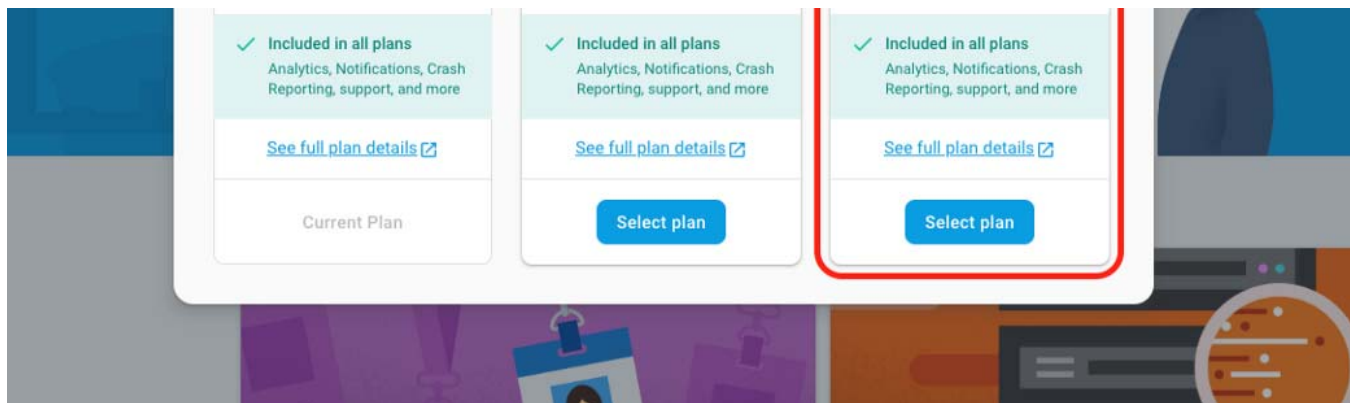
Because we will use Cloud Function for connecting the external network(Connect to LINE Server).In the Spark plan, we cannot use Cloud

Function to connect to any external network due to the restriction so we gonna change it to Blaze plan(or Pay as you go) with this plain you can build the cloud function without any restriction and you won be charged if you use all services in the free quota limit.



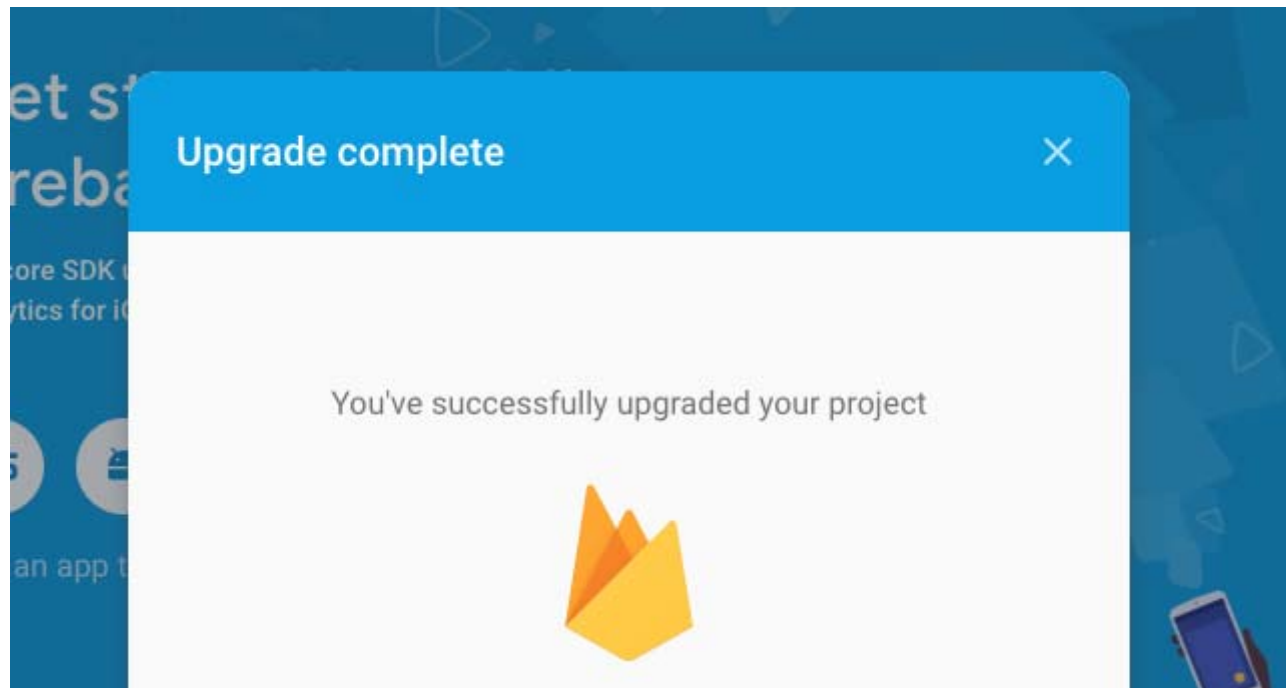
Next, choose Blaze plain

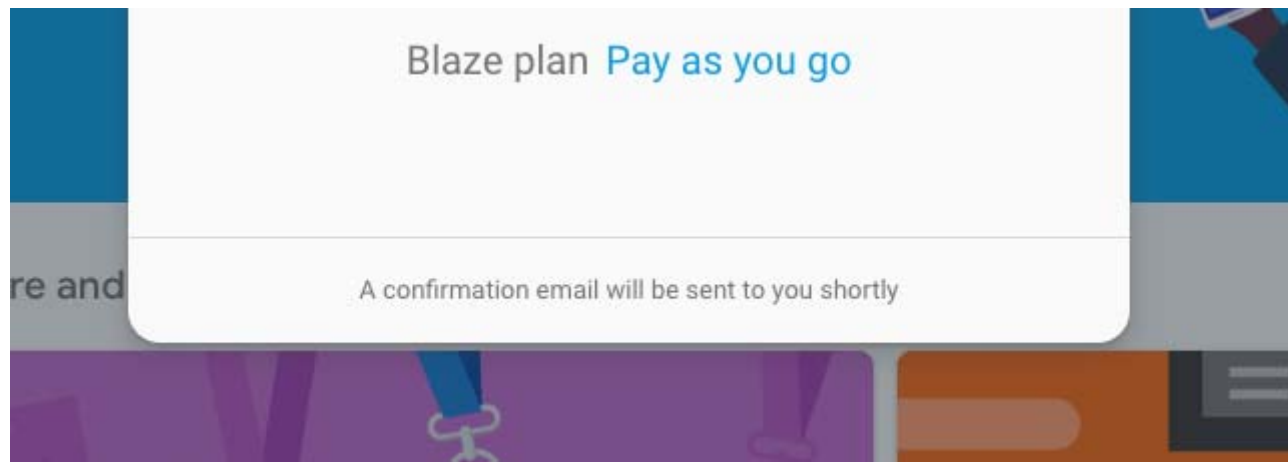




Note: This gonna require your credit card

Done!. (It may required you to specify your billing account in Google Cloud Project so continue to follow it to the end and back to change your plan to Blaze plan again)





* Setting up Firebase tools

Firebase tool is a CLI used to talk with the Firebase server from your terminal. In this tutorial, we'll use it for deploying your Cloud Function to Firebase.

Note: You gonna need NodeJS for installing the Firebase tool

Node.js

Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.

nodejs.org



Install the Firebase tool by typing this command into your terminal

```
$ npm install -g firebase-tools
```

If it doesn't work, you may need to apply the permission for npm by typing "sudo" at the start of the command

Note: This gonna take sometimes

You can check if Firebase tools is installed correctly by typing this command

```
$ firebase -V
```

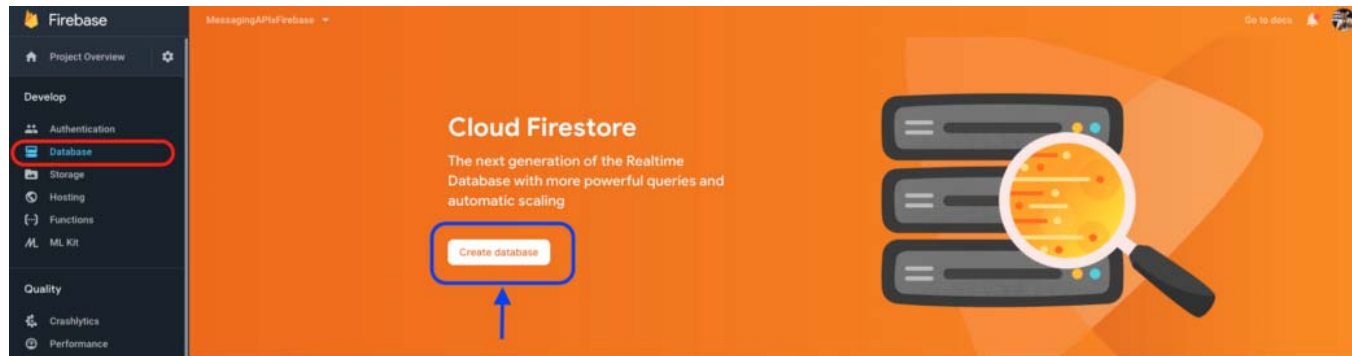
This will return a version of Firebase Tools. Mine is 7.1.0

```
[MaiYaRaps-iMac:~ maiyarapkung$ firebase -V  
7.1.0  
MaiYaRaps-iMac:~ maiyarapkung$ █
```

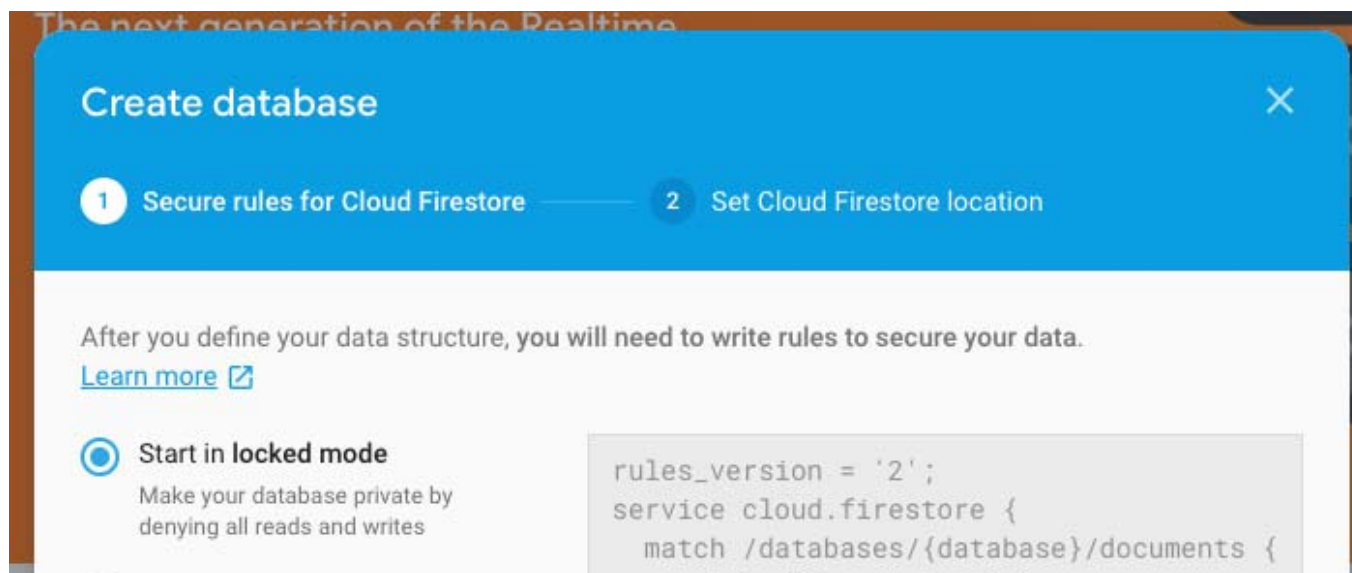
*** Creating the Firestore Database**

Firestore is the NoSQL database which we will use in this project

First, let go to the database tab and then click “Create database” button

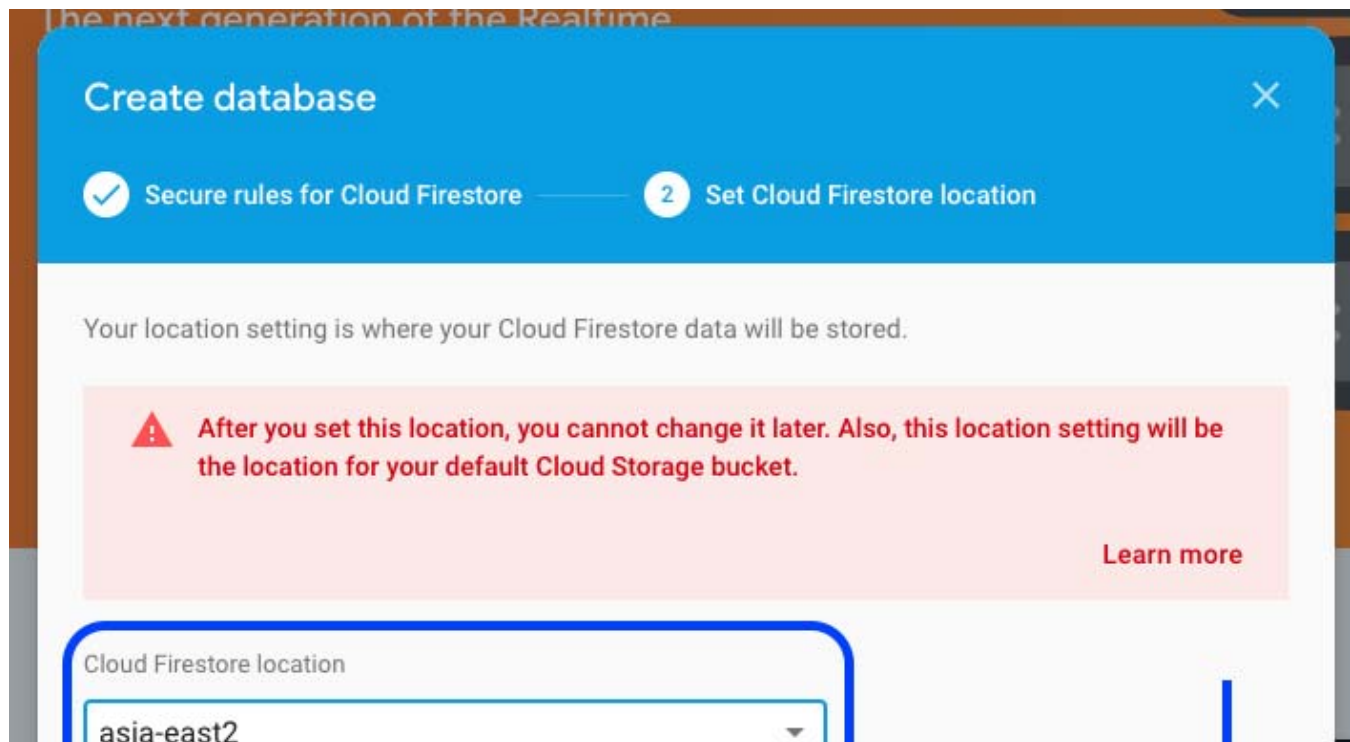


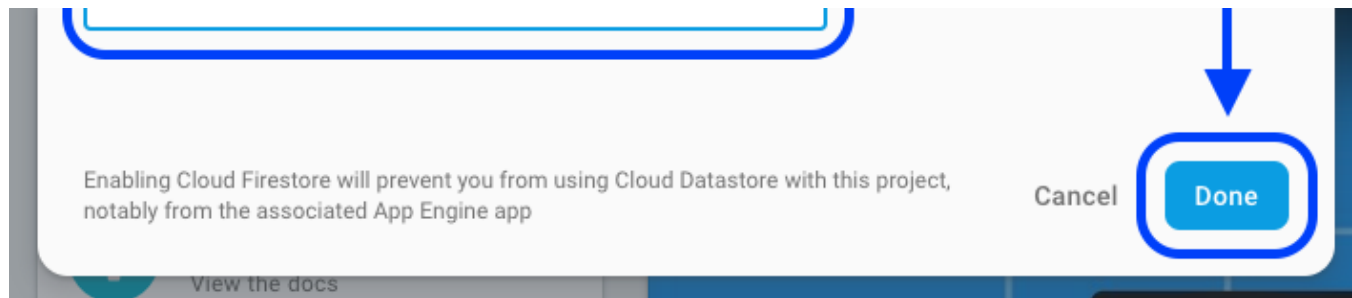
We will start our database in the locked mode for improving the database’s security so leave it default.



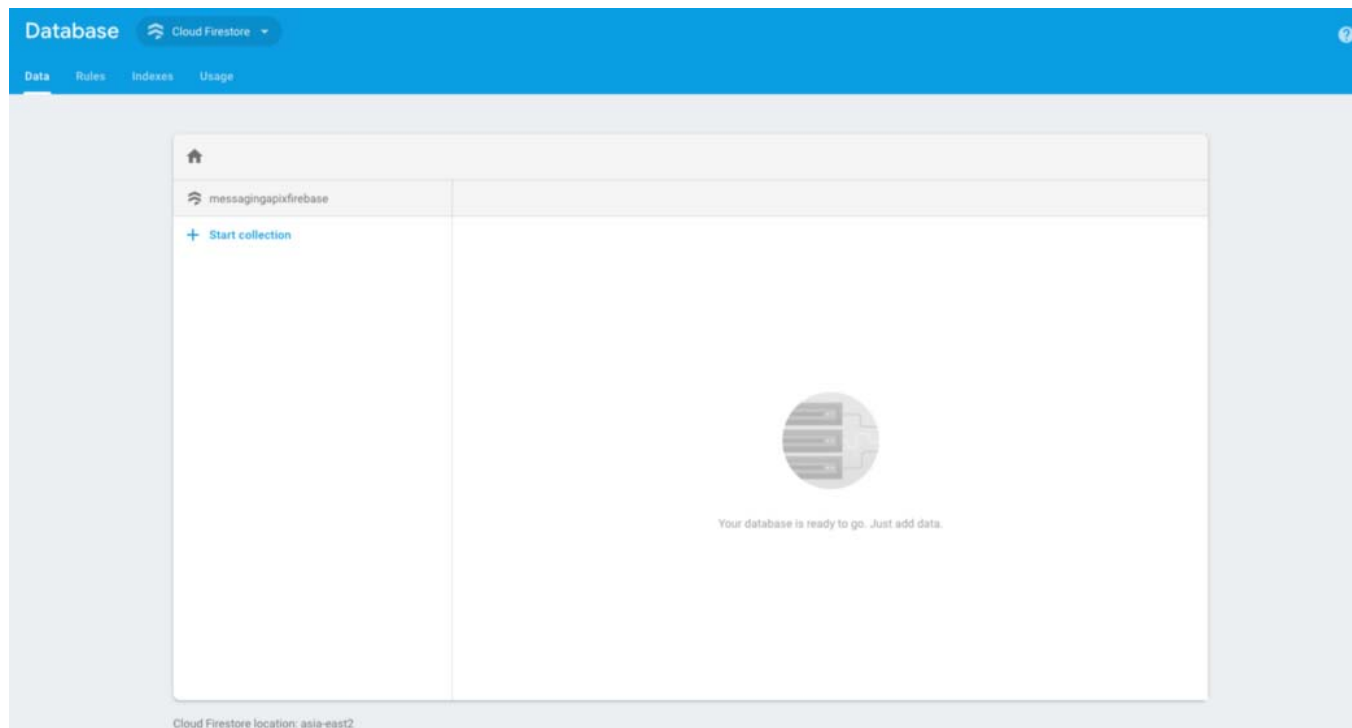


Next, the Firestore database location. Select the location near your service area for decreasing the latency. (Mine is asia-east2)





Once it is created you'll see this page



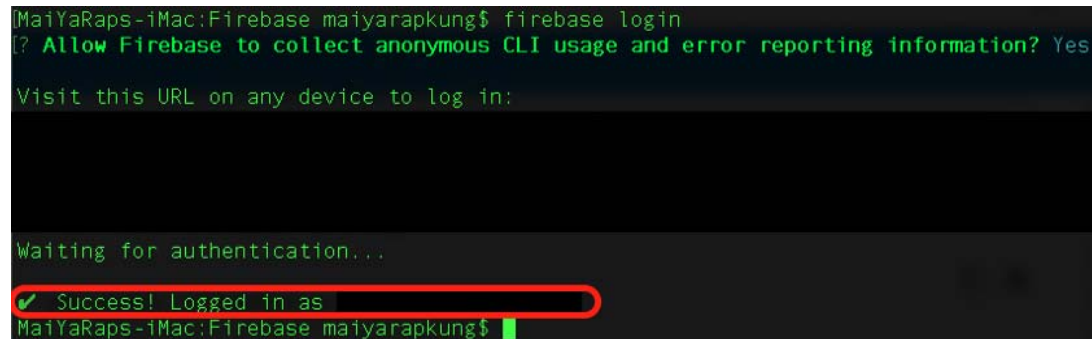
• • •

{Start developing the Cloud Function}

First, we gonna log in with Google Account in Firebase tools by typing this command and follow the login steps

```
$ firebase login
```

Once it done, you'll see this message

A terminal window with a dark background and green text. The prompt is 'MaiYaRaps-iMac:Firebase maiyarakung\$'. The command 'firebase login' has been entered. The output shows a confirmation prompt: '[?] Allow Firebase to collect anonymous CLI usage and error reporting information? Yes'. Below this, it says 'Visit this URL on any device to log in:'. Then, it says 'Waiting for authentication...'. Finally, a green checkmark icon is followed by the text 'Success! Logged in as', which is highlighted by a red oval. The prompt returns to 'MaiYaRaps-iMac:Firebase maiyarakung\$' with a green cursor.

```
MaiYaRaps-iMac:Firebase maiyarakung$ firebase login
[?] Allow Firebase to collect anonymous CLI usage and error reporting information? Yes
Visit this URL on any device to log in:

Waiting for authentication...
✓ Success! Logged in as
MaiYaRaps-iMac:Firebase maiyarakung$
```

Next, we need to create a folder that will contain our cloud function and helper-files and open your terminal then locate into that folder directory. Then we'll initialize our firebase project by typing this command

```
$ firebase init
```

*Next, You'll see these options which have Database, Firestore, Functions, etc.
We gonna select Functions option (Use SpaceBar to select) Then press enter*

```
MaiYaRaps-iMac:Firebase maiyarakung$ firebase init

#####
##      ##      ##      ##      ##      ##      ##      ##
#####
##      ##      ##      ##      ##      ##      ##      ##
#####
##      ##      ##      ##      ##      ##      ##      ##
#####
##      ##      ##      ##      ##      ##      ##      ##
#####

You're about to initialize a Firebase project in this directory:

  /Users/maiyarakung/Local Document/Programming Work/Medium/LineMessagingAPIxFirebase/Firebase

? Which Firebase CLI features do you want to set up for this folder? Press Space to select features, then
Enter to confirm your choices.
  ☐ Database: Deploy Firebase Realtime Database Rules
  ☐ Firestore: Deploy rules and create indexes for Firestore
  ☒ Functions: Configure and deploy Cloud Functions
  ☐ Hosting: Configure and deploy Firebase Hosting sites
  ☐ Storage: Deploy Cloud Storage security rules
```


Then, Choose your project and press enter.

```
=== Project Setup

First, let's associate this project directory with a Firebase project.
You can create multiple project aliases by running firebase use --add,
but for now we'll just set up a default project.

? Select a default Firebase project for this directory:
  [don't setup a default project]

> messagingapixfirebase (MessagingAPIxFirebase)
```



(Move up and down to reveal more choices)

Next, We'll write this function in JavaScript so choose JavaScript Then press enter.

=== Functions Setup

A **functions** directory will be created in your project with a Node.js package pre-configured. Functions can be deployed with **firebase deploy**.

? **What language would you like to use to write Cloud Functions?** (Use arrow keys)

> JavaScript

TypeScript

Next, answer all question like this

Do you want to use ESLint to catch probable bugs and enforce style? — Yes

Do you want to install dependencies with npm now? — absolutely Yes

Note: This will start to install all of the dependencies that you gonna need

Done!. Now let check what we got in our project folder.

```
| -firebase.json  
| -functions
```

```
|- node_modules
|- index.js
|- package-lock.json
|- package.json
```

We'll get something like this.

* Coding part for CloudFunction

*Next, we'll start creating the Cloud Function. open up the file named **index.js** in functions folder with your favorite IDE (Mine is Visual Studio Code). You'll see the initial code like this.*

```
1  const functions = require('firebase-functions');
2  // // Create and Deploy Your First Cloud Functions
3
4
5  // // https://firebase.google.com/docs/functions/write-firebase-functions
6  //
7  // exports.helloWorld = functions.https.onRequest((request, response) => {
8  //   response.send("Hello from Firebase!");
9  // });
```

LineMessagingAPIxFirebase-DefaultCloudFunction.js hosted with ❤ by GitHub

[view raw](#)

Now we'll create our Cloud Function called "LineMessAPI" which will receive the POST Request from LINE Messaging API also send our message back to it.

```
1 exports.LineMessAPI = functions.https.onRequest((request, response) => {  
2     response.send("Hello from CloudFunction!");  
3 });
```

LineMessagingAPIxFirebase-TestCloudFunction.js hosted with ❤ by GitHub

[view raw](#)

Pro Tips!

You can specify where your cloud function is going to be deployed near your services area (Mine is *asia-east2 = HongKong*), This will decrease latency between your bot and cloud function (Make it reply faster). even more, you can specify how much ram using in this function is good for your project (Mine is 1GB).

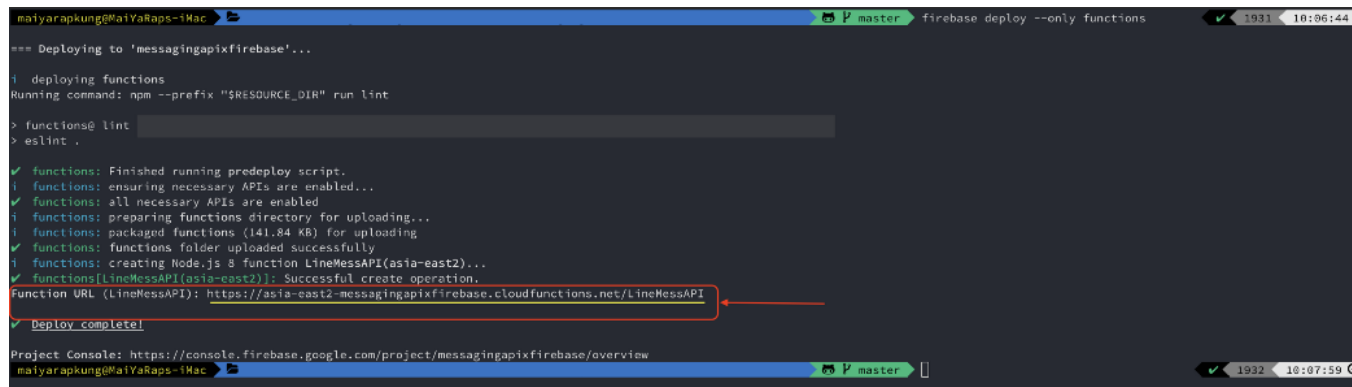
Like this:

```
1 const functions = require('firebase-functions');  
2 const region = 'asia-east2';  
3 const spec = {  
4     memory: "1GB"  
5 };  
6 exports.LineMessAPI = functions.region(region).runWith(spec).https.onRequest((request, response) => {  
7  
8 });
```

We'll use the cloud function URL as the webhook so let deploy it for the first time and grab to URL and config it at LINE Developer console.

You can deploy your cloud function by using this command

```
$ firebase deploy --only functions
```



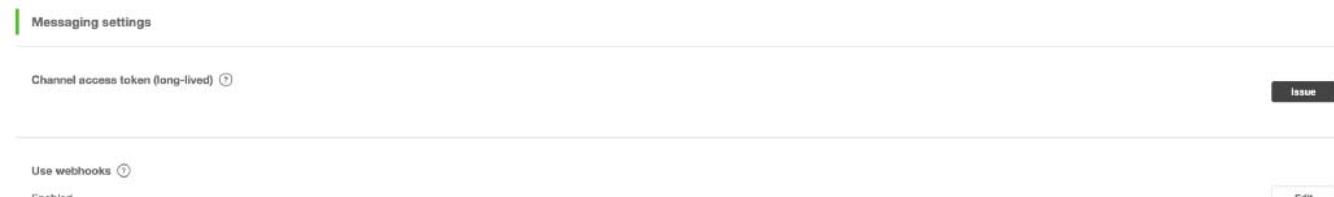
```
maiyarapkung@M1YaRaps-iMac ➜ firebase deploy --only functions
== Deploying to 'messagingapixfirebase'...

i deploying functions
Running command: npm --prefix "$RESOURCE_DIR" run lint
> functions@ lint
> eslint .

✓ functions: Finished running predeploy script.
✓ functions: ensuring necessary APIs are enabled...
✓ functions: all necessary APIs are enabled
✓ functions: preparing functions directory for uploading...
✓ functions: packaged functions (141.84 KB) for uploading
✓ functions: functions folder uploaded successfully
✓ functions: creating Node.js 8 function LineMessAPI(asia-east2)...
✓ functions[LineMessAPI(asia-east2)]: Successful create operation.
Function URL (LineMessAPI): https://asia-east2-messagingapixfirebase.cloudfunctions.net/LineMessAPI
✓ Deploy complete!

Project Console: https://console.firebase.google.com/project/messagingapixfirebase/overview
maiyarapkung@M1YaRaps-iMac ➜
```

You'll get the functions URL(This will show up for the first time deploy only, also You may find it in the firebase website in function page)





Webhook URL Requires SSL ⓘ

https://

Max: 500 characters

Please enter a URL that supports HTTPS.

Update Cancel

Note: The cloud function URL will be generated when you deploy the cloud function for the first time, you can get it from the CLI or the firebase website in function page.

*Back to our cloud function, When we want to do some reply message back to our user in LINE so we need to do the POST request from our cloud function back to LINE Messaging API so that we need to call **the dependency named request (Version ^2.88.0)**, So let open up the package.json file (Not package-lock.json) then add this dependency to your cloud function.*

```
{
  "name": "functions",
  "description": "Cloud Functions for Firebase",
  "scripts": {
    "lint": "eslint .",
    "serve": "firebase serve --only functions",
    "shell": "firebase functions:shell",
    "start": "npm run shell",
    "deploy": "firebase deploy --only functions",
    "logs": "firebase functions:log"
  },
  "engines": {
    "node": "8"
  },
  "dependencies": {
    "firebase-admin": "^8.0.0",
    "firebase-functions": "^3.1.0",
    "request": "^2.88.0"
  },
  "devDependencies": {
    "eslint": "^5.12.0",
    "eslint-plugin-promise": "^4.0.1",
    "firebase-functions-test": "^0.1.6"
  },
  "private": true
}
```


Don't forget to install your dependency by using this command

```
$ npm install
```

Next, let call this dependency in our code like this but let also initialize the firebase-admin for connecting to the Firestore Database

```
1 // Call request dependency
2 const request = require('request');
3 //------(Firebase Admin)-----//
4 // Call firebase-admin dependency
5 const admin = require("firebase-admin");
6 // Initialize firebase-admin
7 admin.initializeApp(functions.config().firebase)
8 // Create variable for connecting to the Firestore Database
9 const db = admin.firestore();
```

LineMessagingAPIxFirebase-request.js hosted with ❤ by GitHub

[view raw](#)

Now let take a look into the example request which our webhook will receive

```

{
  "destination": "Ua0343aa97924b261da9bd5362039****",
  "events": {
    "0": {
      "replyToken": "29e4fe9d921a430680635a681fa53da0",
      "timestamp": "156414582****",
      "type": "message",
      "message": {
        "id": "10281586537***",
        "type": "text",
        "text": "Hello"
      },
      "source": {
        "type": "user",
        "userId": "U1efbc797c7174dd636c047f5ca8e****"
      }
    }
  }
}

```

Here is the data containing our user information, message, and the replyToken(This will be used for sending the reply back to the user). As you can see this is the JSON formatted data. In this project, we need the text, userId, and timestamp from this data. In nodejs, we can decode it easily like this.

```

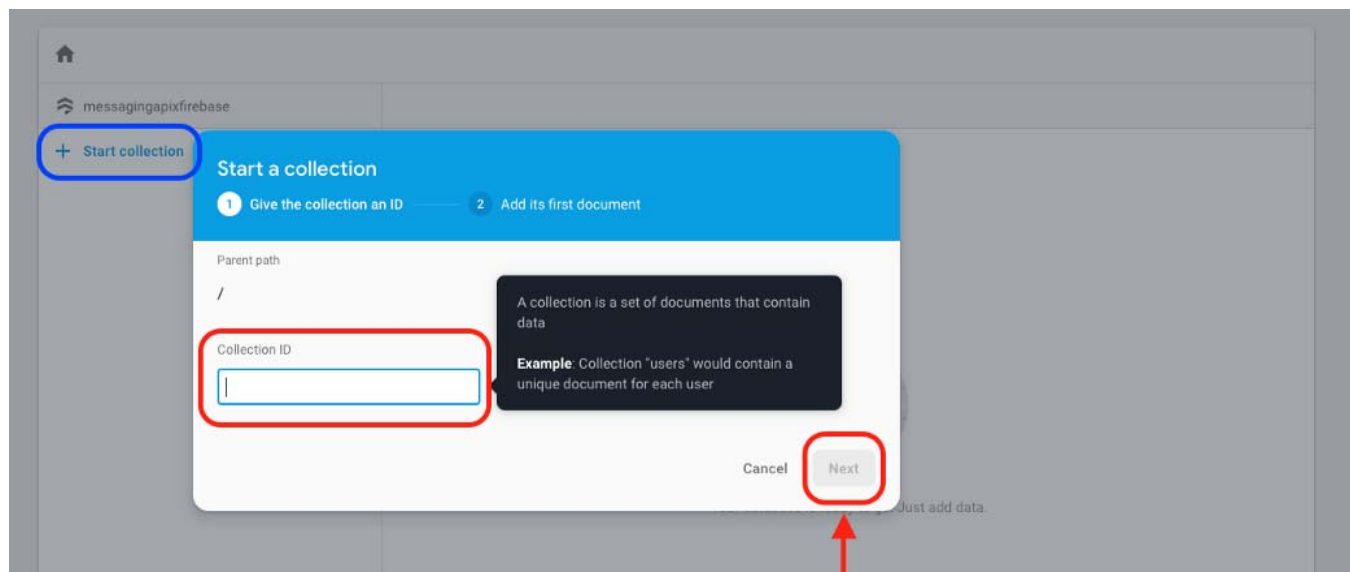
1  exports.LineMessAPI = functions.region(region).runWith(spec).https.onRequest((request, response)
2    // We create this variable for getting the request body from LINE Messaging API and keep it
3    var event = request.body.events[0]
4    // This is location where our userId is contained (Full path is "events[0].source.userId")
5    var userId = event.source.userId
6    // This is location where our timestamp is contained (Full path is "events[0].timestamp")

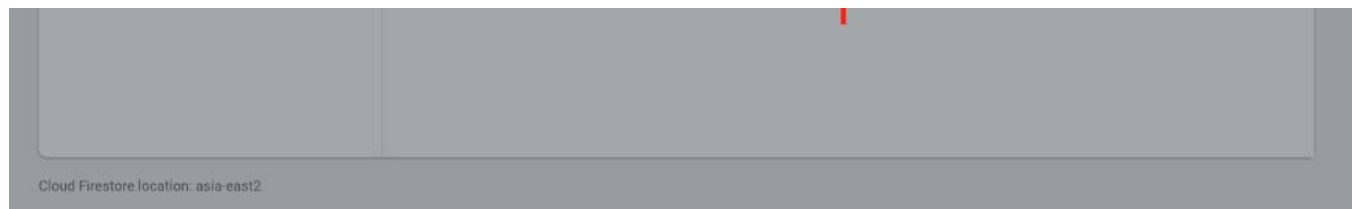
```

```
7   var timestamp = event.timestamp
8
9   var userText = ""
10  // We just want to make sure that the message is actually a text not sticker, image, or some
11  if (event.type === "message" && event.message.type === "text"){
12    // This is location where our text is contained (Full path is "events[0].message.text")
13    userText = event.message.text
14  } else {
15    // If it doesn't the text
16    userText = "(Message type is not text)";
17  }
18  });
```

LineMessagingAPIxFirebase-JsonDecode.js hosted with ❤ by GitHub [view raw](#)

*Now, let keep `userId`, `text`, and `timestamp` into the Firestore Database but first let create the collection called **chat-history***





Next, choose auto-id for document id and just click save.

A screenshot of the "Start a collection" dialog box in the Google Cloud console. The dialog has a blue header with the title "Start a collection" and two steps: "1 Give the collection an ID" (completed) and "2 Add its first document" (active). The "Document parent path" is set to "/chat-history". The "Document ID" field is empty, and the "Auto-ID" button next to it is highlighted with a red circle. A tooltip points to the "Auto-ID" button, stating: "A collection must contain at least one document, Cloud Firestore's unit of storage. Use an auto-generated ID or enter a custom ID if needed. Documents store your data as fields." Below the "Document ID" field is a table for adding fields to the document. The table has three columns: "Field", "Type", and "Value". The first row shows an empty "Field" input, followed by an equals sign, a "string" type selected from a dropdown, and an empty "Value" input. There is a minus button to the right of the "Value" input. At the bottom left of the table is a plus button to add more fields. At the bottom right of the dialog are "Cancel" and "Save" buttons.

Start a collection

1 Give the collection an ID — 2 Add its first document

Document parent path

/chat-history

Document ID

Auto-ID

A collection must contain at least one document, Cloud Firestore's unit of storage. Use an auto-generated ID or enter a custom ID if needed. Documents store your data as fields.

Field	Type	Value
	= string	

+

Cancel Save

Back to coding, We will insert our data into this collection by using this command

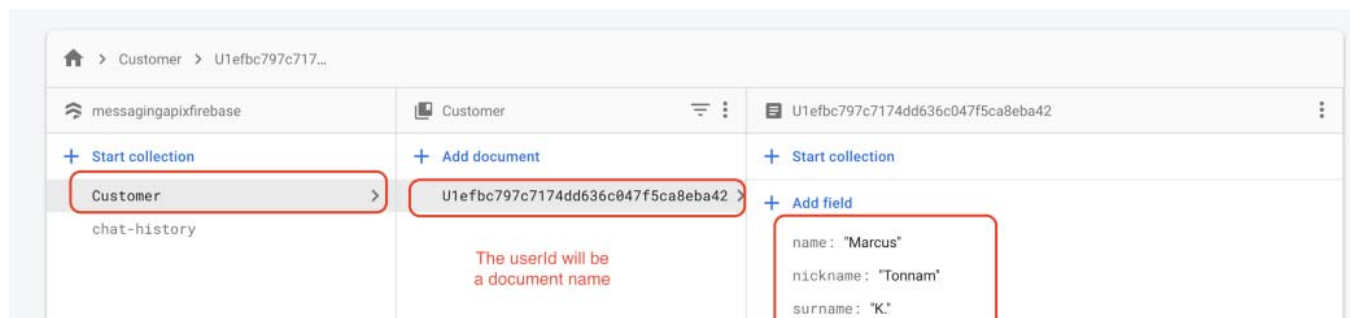
```
1 // Create one variable then call the db variable
2 // for adding data to the Firestore database we use this command
3 // db.collection([Collection Name]).doc([DocumentName]).set({}),
4 // The document name must be a string. so we call .toString() function to convert the timestamp
5 const addChatHistory = db.collection("chat-history").doc(timestamp.toString()).set({
6     //Then set our data into these index
7     "userId": userId,
8     "Message": userText,
9     "timestamp": timestamp
10 })
```

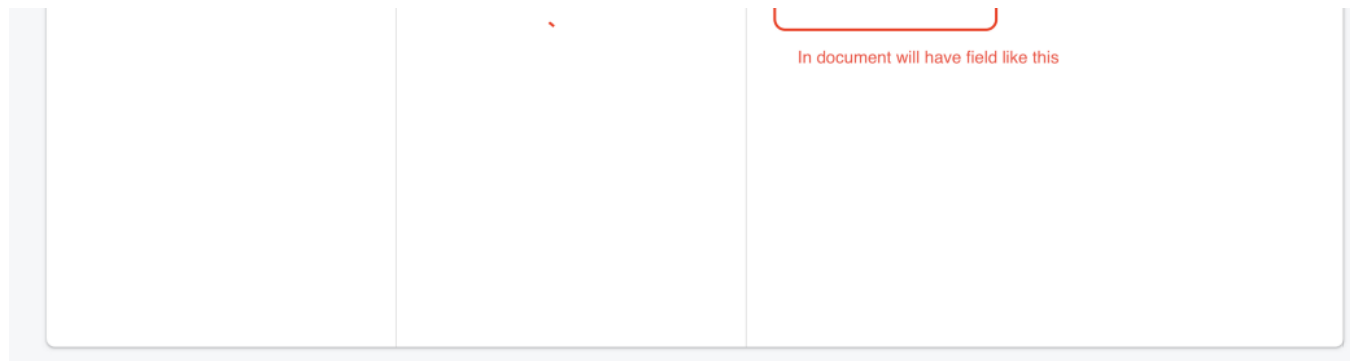
LineMessagingAPIxFirestore-addingDatatoFirestore.js hosted with ❤ by GitHub

[view raw](#)

Now we complete two of our goals, let continue our project

Next, let create another collection called Customer for keeping our customer information (Do the same step as you created the chat-history collection).





Example data.

You might ask, How can I get my userId? — *You'll see.*

Then we will check if our user is in the Customer collection by using this command

```
1      const getUserData = db.collection("Customer").doc(userId).get().then( returnData =>{
2          if (returnData.exists){
3              // This will return the data from the customer collection
4              var name = returnData.data().name
5              var surname = returnData.data().surname
6              var nickname = returnData.data().nickname
7              // Call the reply function
8          } else {
9              // Reply function
10         }
11         return null
12     }).catch(err => {
13         console.log(err)
14     })
```

*Next, We'll create the function for doing the reply message back to the user also specify the header of the request. (We'll create this function **outside of the exports.LineMessAPI**)*

```
1 // Don't forget to enter your channel access token.
2 const LINE_HEADER = {
3   "Content-Type": "application/json",
4   "Authorization": "Bearer {'Enter your channel access token'}"
5 }
6
7 function reply_message(replytoken,textfrom){
8   return request.post({
9     uri: `https://api.line.me/v2/bot/message/reply`,
10    headers: LINE_HEADER,
11    body: JSON.stringify({
12      replyToken: replytoken,
13      messages: [
14        {
15          type: "text",
16          text: textfrom
17        }
18      ]
19    })
20  });
21 }
```

The function `reply_message` can be used by passing the 2 data into it. First is `replyToken` and second is the text you want to reply. Mean that this function can only reply just the text if you want it to reply more message type you need to edit its body.

Next, let fill the code with this function.

```
1      const getUserData = db.collection("Customer").doc(userId).get().then( returnData =>{
2          if (returnData.exists){
3              let name = returnData.data().name
4              let surname = returnData.data().surname
5              let nickname = returnData.data().nickname
6              //This is the reply_message function and I'm going to send the user data back to our
7              reply_message(replyToken, `Hello ${nickname}(${name} ${surname})`)
8          } else {
9              // This will be called if the user is not exist in the database
10             reply_message(replyToken, "You are not the customer, Register?")
11         }
12         return null
13     }).catch(err => {
14         console.log(err)
15     })
```

LineMessagingAPIxFirebase-ReplyFunctionfullfill.js hosted with ❤ by GitHub

[view raw](#)

Lastly, Don't forget to return 200 http status code back to LINE Messaging API by using this command


```
1 return respond.status(200).send(request.method);
```

LineMessagingAPIxFirebase-Lastly.js hosted with ❤ by GitHub

[view raw](#)

Here is the complete code.

```
1  const functions = require('firebase-functions');
2  const request = require('request');
3  const region = 'asia-east2';
4  const spec = {
5    memory: "1GB"
6  };
7
8  const admin = require("firebase-admin");
9  admin.initializeApp(functions.config().firebase);
10 const db = admin.firestore();
11
12 exports.LineMessAPI = functions.region(region).runWith(spec).https.onRequest((request, respond)
13   var event = request.body.events[0]
14   var userId = event.source.userId;
15   var timestamp = event.timestamp;
16   var replyToken = event.replyToken;
17   var userText = ""
18   if (event.type === "message" && event.message.type === "text"){
19     userText = event.message.text
20   } else {
21     userText = "(Message type is not text)";
22   }
23   const addChatHistory = db.collection("chat-history").doc(timestamp.toString()).set({
24     "userId": userId,
25     "Message": userText,
26     "timestamp": timestamp
```

```

27     })
28
29     const getUserData = db.collection("Customer").doc(userId).get().then( returnData =>{
30         if (returnData.exists){
31             var name = returnData.data().name
32             var surname = returnData.data().surname
33             var nickname = returnData.data().nickname
34             reply_message(replyToken, `Hello ${nickname}(${name} ${surname})`)
35         } else {
36             reply_message(replyToken, "You are not the customer, Register?")
37         }
38         return null
39     }).catch(err => {
40         console.log(err)
41     })
42
43     return respond.status(200).send(request.method);
44 });
45
46 const LINE_HEADER = {
47     "Content-Type": "application/json",
48     "Authorization": "Bearer {'Your channel access token'}"
49 }
50
51 function reply_message(replytoken, textfrom){
52     return request.post({
53         uri: `https://api.line.me/v2/bot/message/reply`,
54         headers: LINE_HEADER,
55         body: JSON.stringify({
56             replyToken: replytoken,
57             messages: [
58                 {
59                     type: "text",
60                     text: textfrom
61                 },
62             ]
63         })
59     });
60 }

```

```
}  
]  
})  
});  
}
```

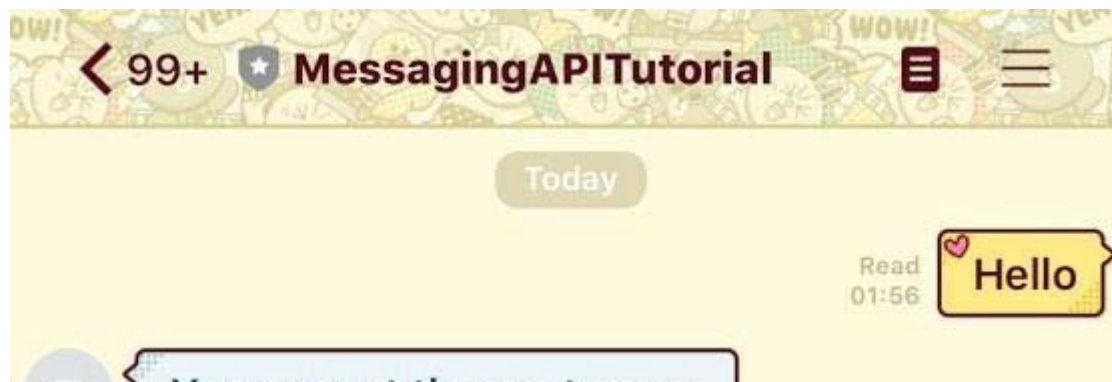
LineMessagingAPIxFirebase-FullCode.js hosted with ❤ by GitHub [view raw](#)

Next, Deploy it by using this command and see the result

```
$ firebase deploy --only functions
```

{Done! Let test it out!}

First, Let leave every collection in the database default and send the message to our chat-bot. This is the result.



You are not the customer,
Register?



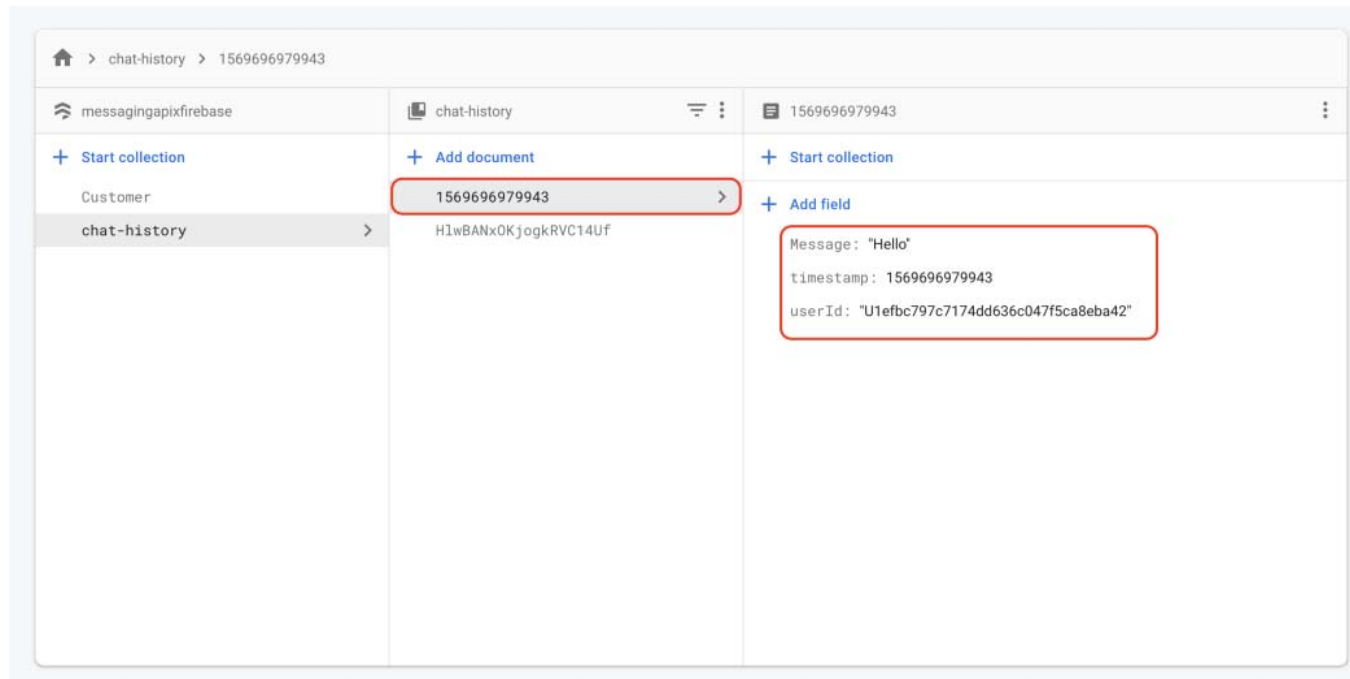
01:56



Aa

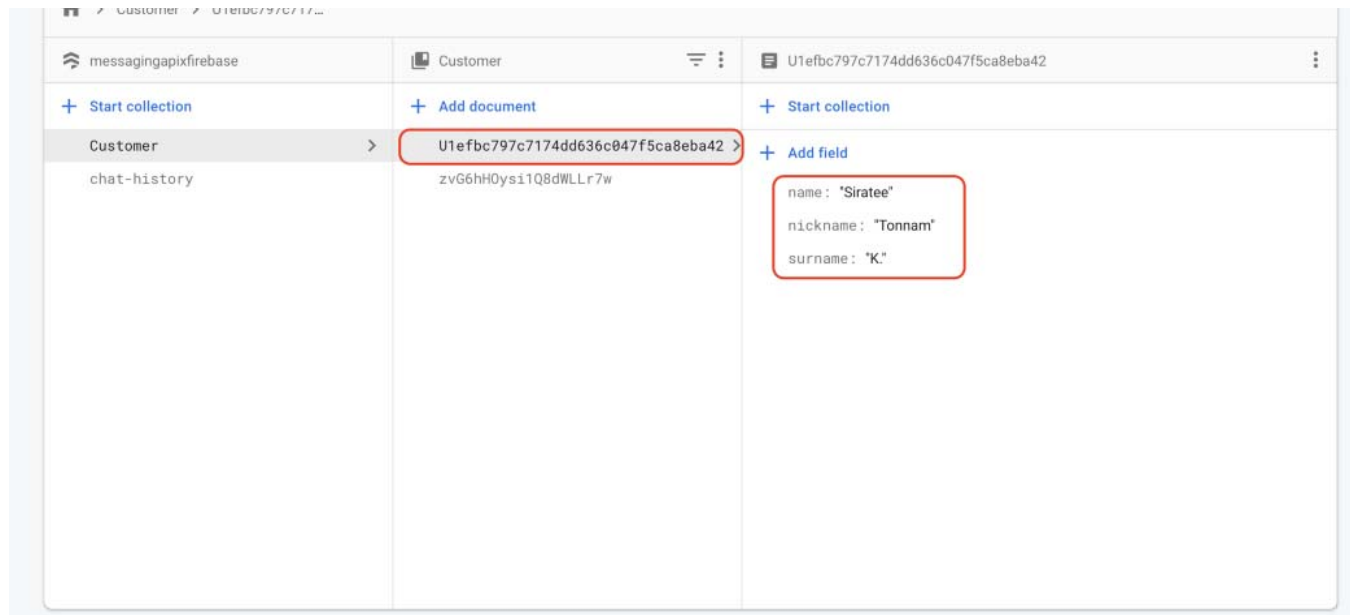


and let see in the Firestore Database, In chat-history collection



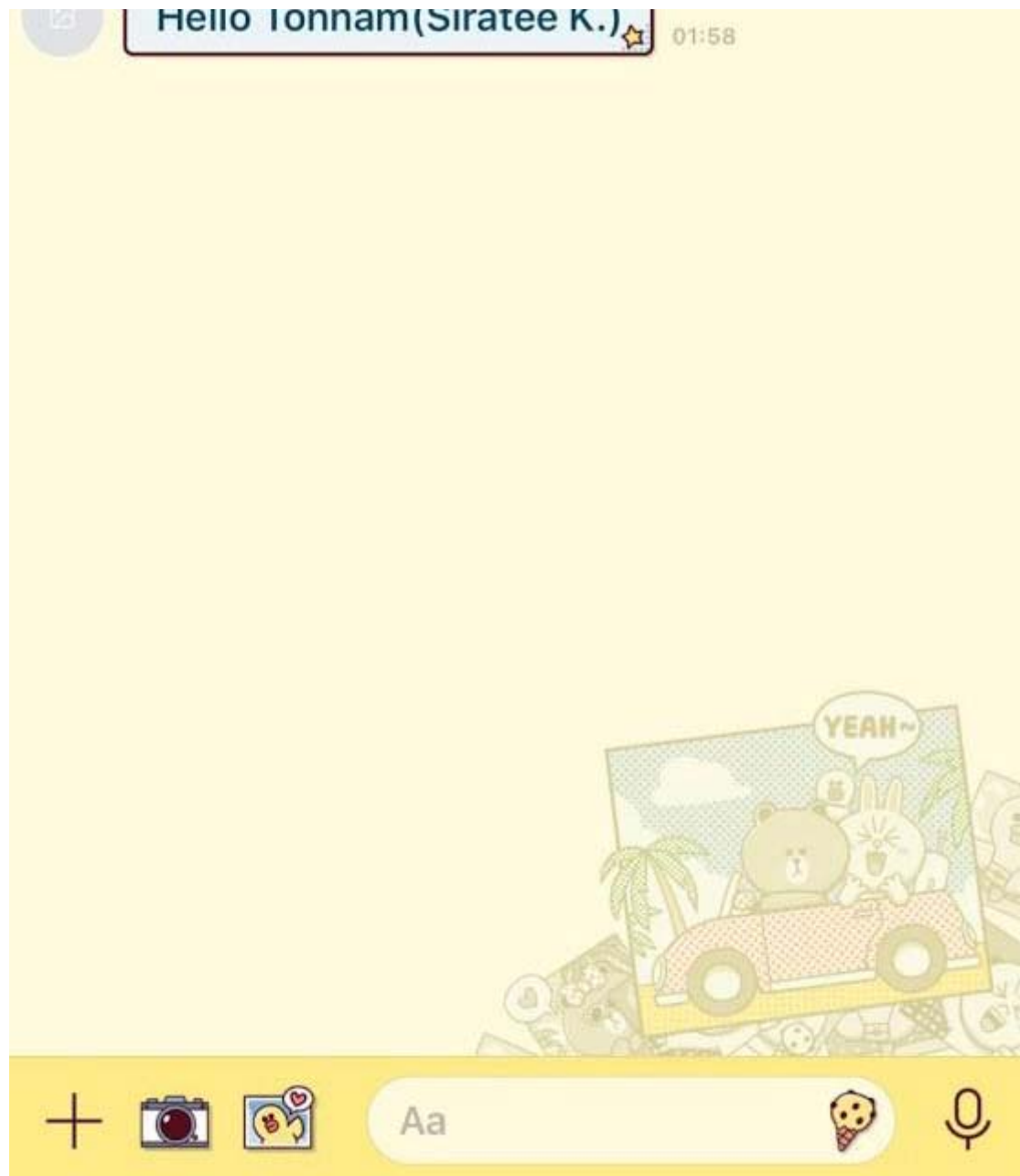
You'll see the new document contains the chat detail from the user.

Now let grab the userId from this document and then go to Customer collection and then create the new document by using userId as the document name. also create the fields like this (name, surname, nickname)



And then let test sending the message to our chat-bot again





That is. Now you can build your chat-bot using cloud function as a webhook and connect your cloud function for storing or querying the data in the

Firestore database...

Here is the full source code on my GitHub:

sirateek/MessagingAPIxFirebase

The full source code in this tutorial is on my github.com

. . .

So that's all for this topic, Thank you for your reading. This tutorial is taking a bit long time to read because it has so many details to explain. The actual purpose is to make you understand it not to make you just copy and paste the code. so that why this takes a bit long time to read.

If you have the question, Feel free to write it down in the comment, I'll reply ASAP and also If you like this tutorial please click the handclaps button that will be my encouragement for posing the next tutorial ^^

and as always if I make the mistake about the language please apologize for this. because English is not my primary language but I'll try my best for sharing my knowledge with other developers in worldwide

Thank you for reading and see you in the next post ^.^)~

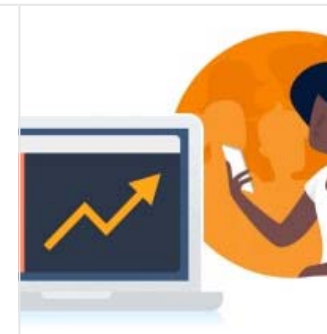
Tonnam,
Software Developer.

{Referent}

Documentation | Firebase

Firebase SDK reference, integration guides, sample code, and libraries

firebase.google.com



LINE Developers

Edit description

developers.line.biz



JavaScript

Line Messging Api

Firestore

Firebase

Line Api



[About](#) [Write](#) [Help](#) [Legal](#)