Big Data Analytics: Streaming Data Collection

Streaming Data

- Music
- Image
- Log of Online Shopping (login、add_to_cart 、browsing 、checkout…etc)

- What other types of data do we generate every day?

Outline

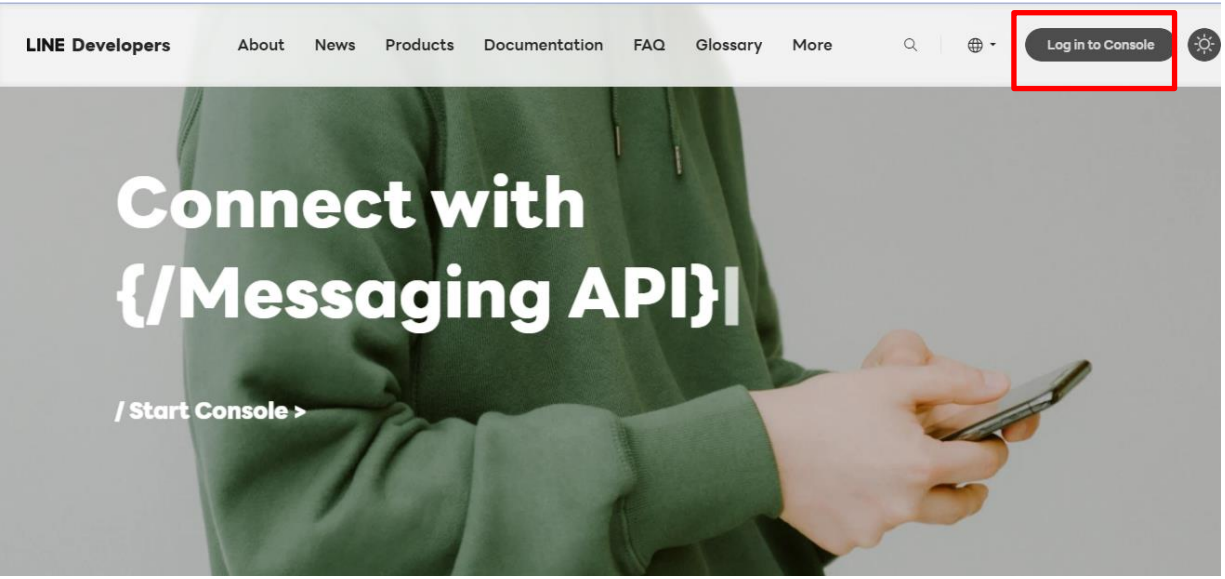
- Create LINE Bot
- 「Parrot」LINE bot
- Graphic LINE Bot
  - Create Rich Menu
- LINE Bot basic interaction
  - Return Text
  - Return Image
  - Return Sticker
  - Mutiple Messages
  - Return Location
  - Quick Replies

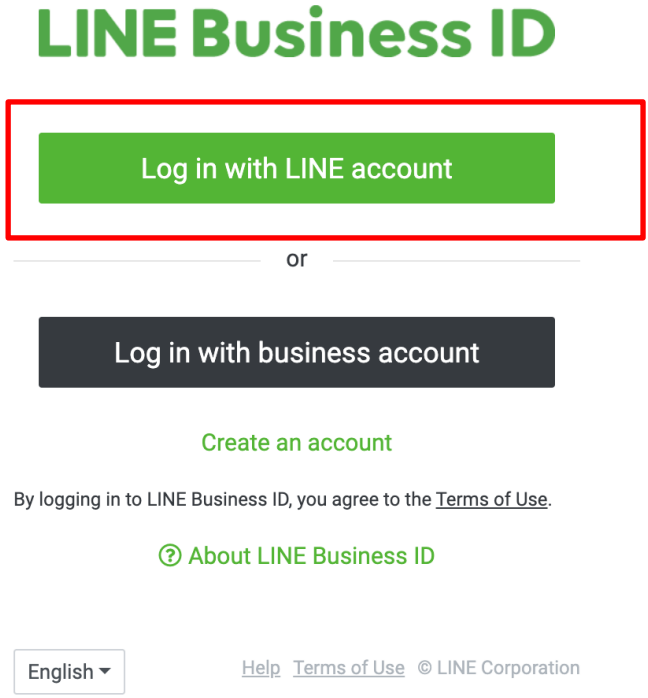
Line Bot

- Since April 2016, Line has been offering a free trial account for the "Line Bot API," allowing anyone to develop diverse applications of chatbots on the LINE platform.
- Up to now, there have been 100,000 Line Bots developed and in use.
- In April 2019, Line introduced Line Official Account 2.0, significantly enhancing various features of Line Bots. This means that we can also build Line Bots to collect streaming data.

To apply for a LINE developer account

- https://developers.line.biz/en/
- Click ‘Log in to Console’, then click the 'Log in with LINE Account' button, enter your LINE account and password, and click the 'Log In' button."





Enter some relevant personal information.

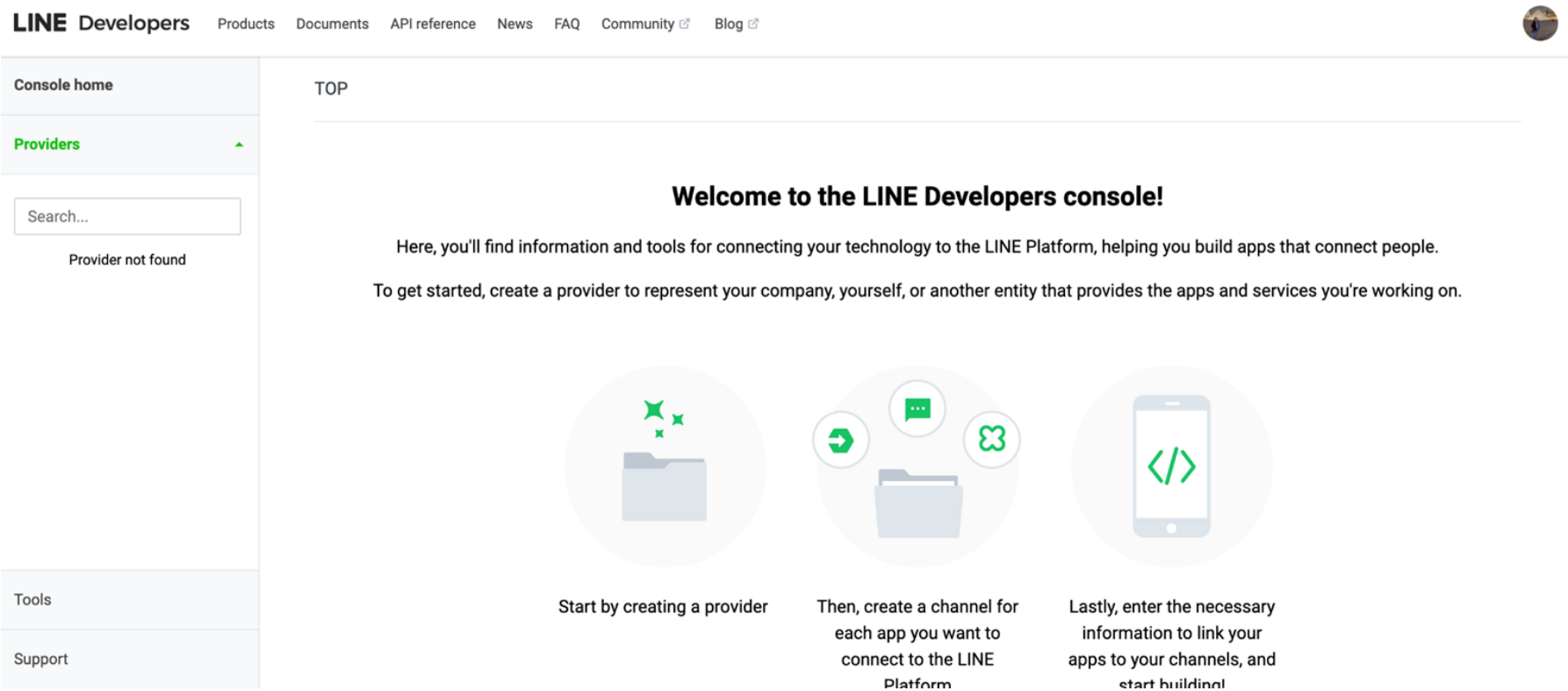
- Enter some relevant personal information.
- After entering the information, click 'Create’.

Hi, 林家瑜（Sally)! Welcome to the LINE Developers console.

Enter your information and select Create my account.

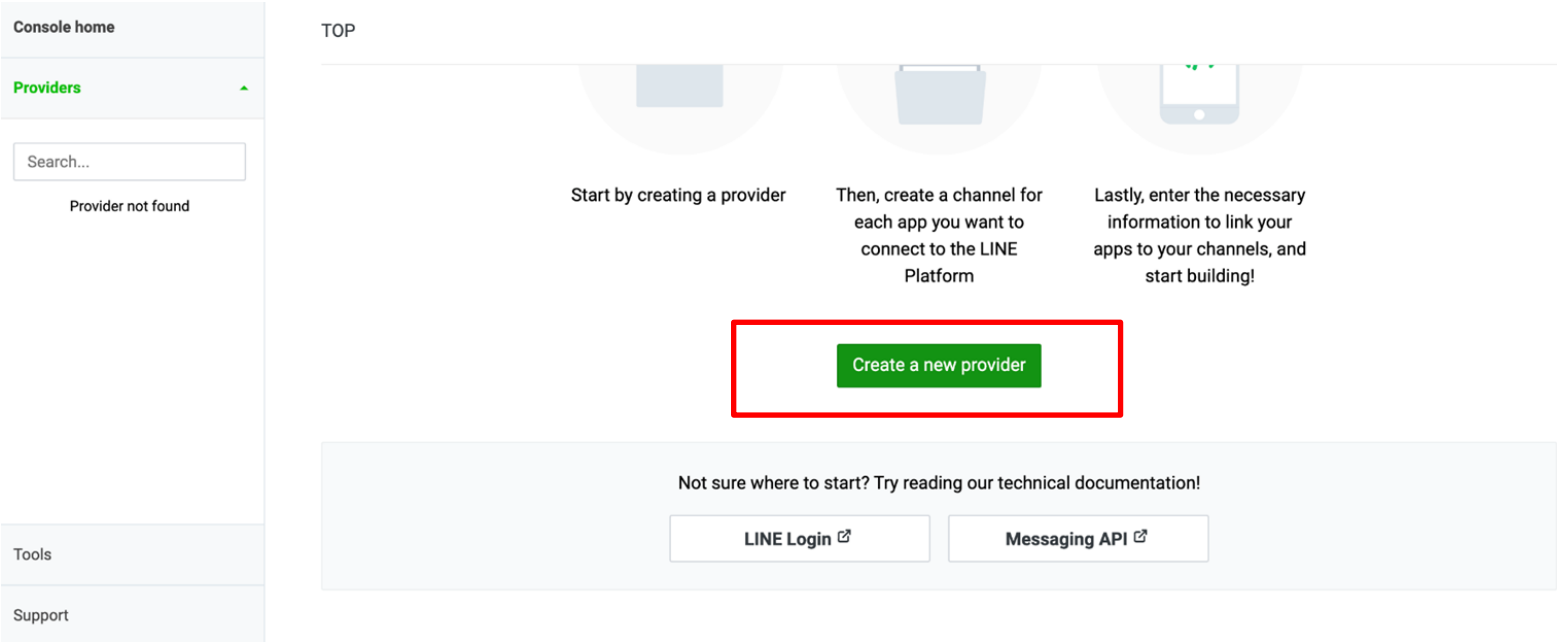
You can still change your developer information later.

Entering LINE Development Console



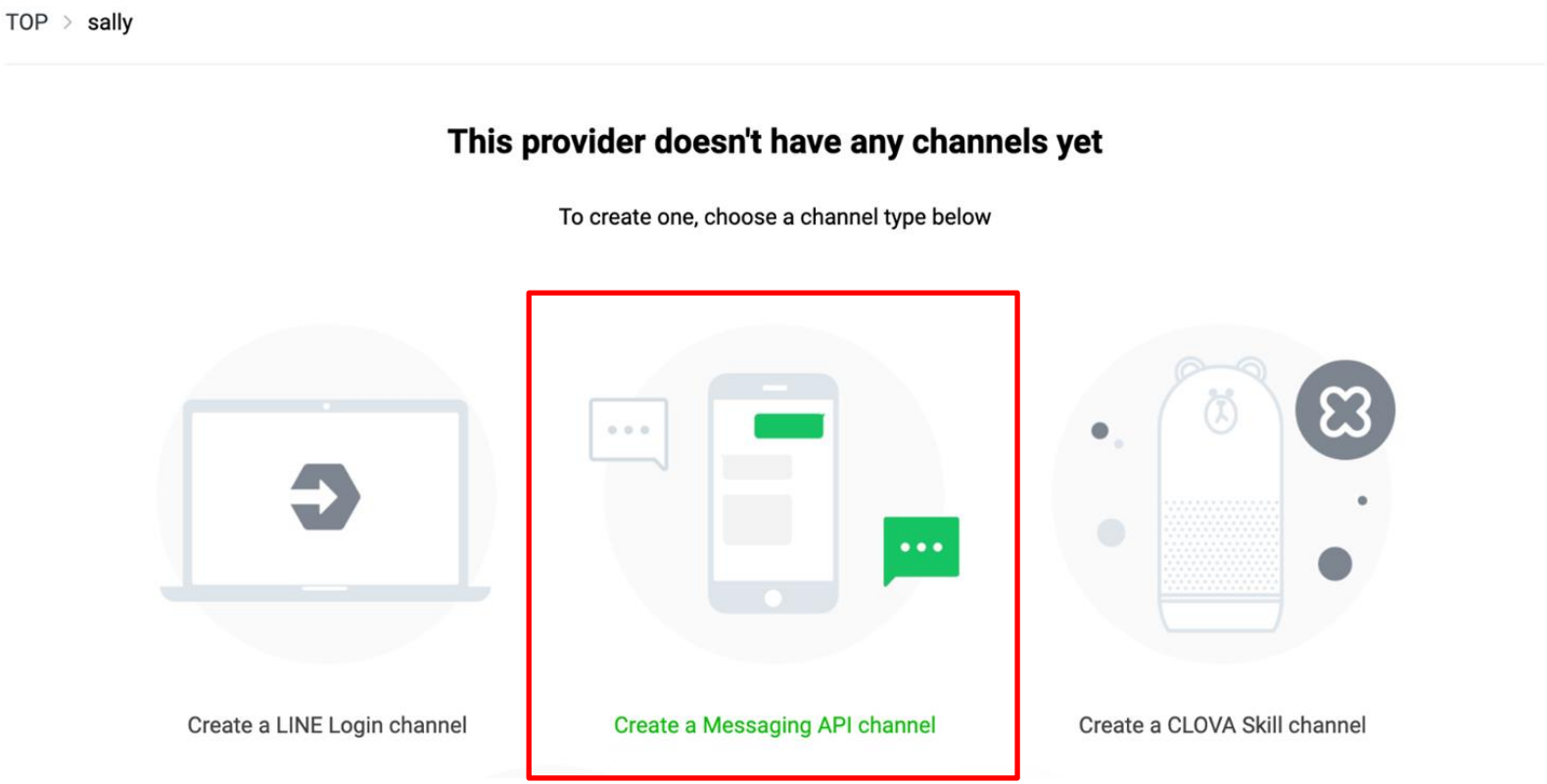
Create Provider

- Create a new provider
- Enter your name in the 'Provider' field



Create Message API Channel

- After create a new Provider
- Create a Message API Channel



Upload LINE Bot Icon

- Upload a LINE Bot icon in the 'Register' section (you can choose your preferred image).

Enter the channel name and description

- Enter your channel name and description
  - You cannot change Line Bot name in 7 days

Channel name

春月日式食堂

Note: The channel name can't be changed for seven days.

- ✓ Don't leave this empty
- ✓ Don't use special characters (4-byte Unicode)
- ✓ Enter no more than 20 characters

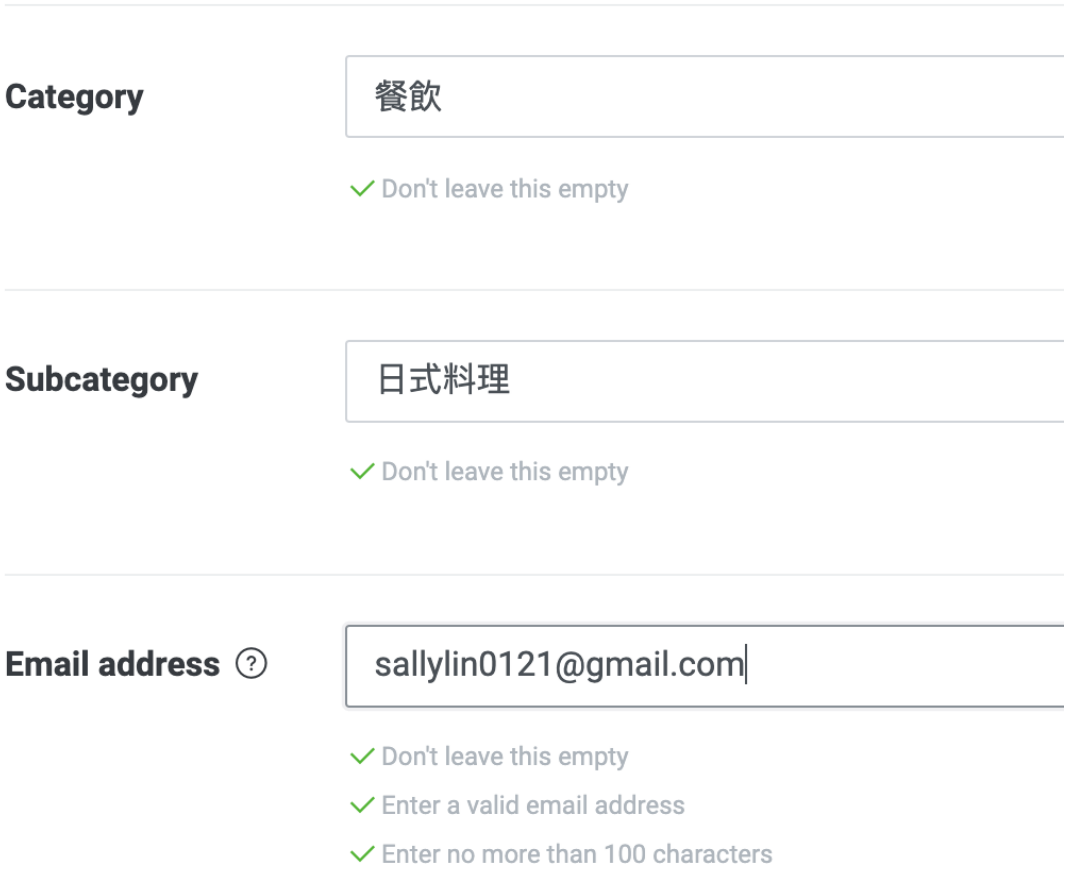
Channel description

有溫度的日式食堂

- ✓ Don't leave this empty
- ✓ Don't use special characters (4-byte Unicode)
- ✓ Enter no more than 500 characters

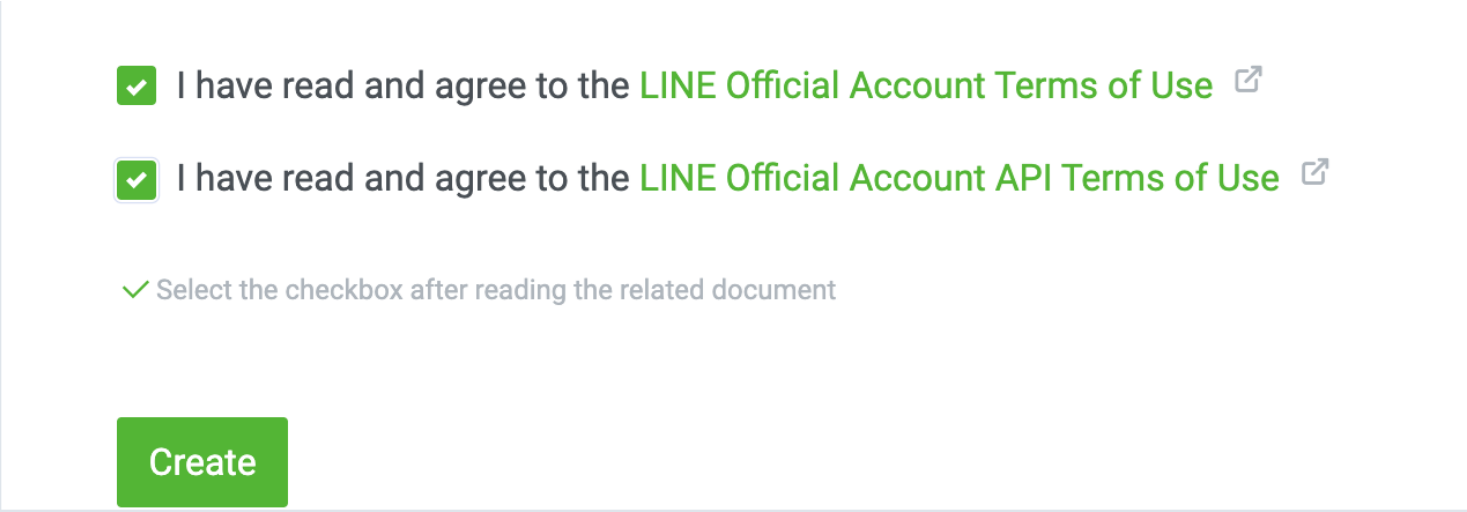
Choose Category

- Choose Category and Subcategory
- Enter your Email address



Check the account permissions

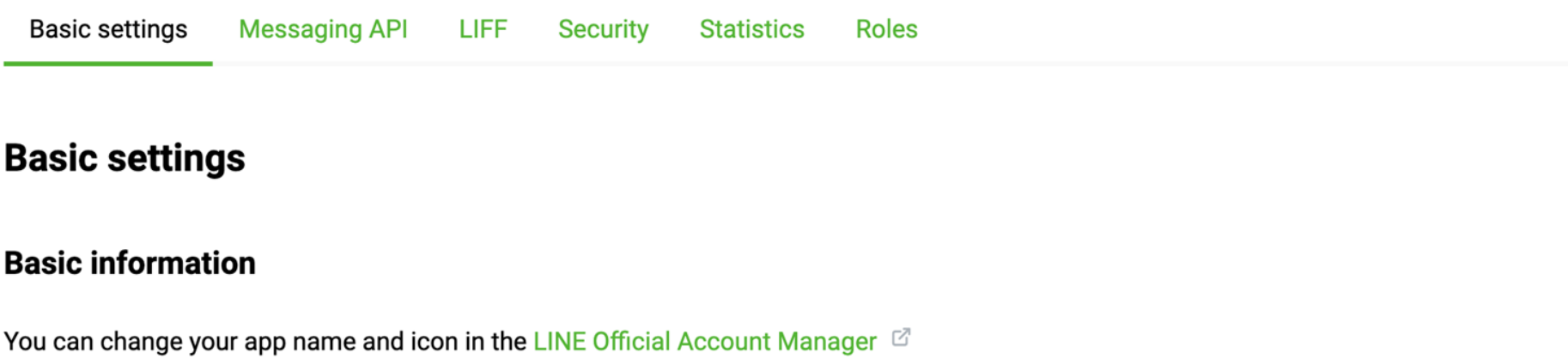
- Check the account permissions.
- Create!!



LINE Bot Creation Success

LINE Bot Name

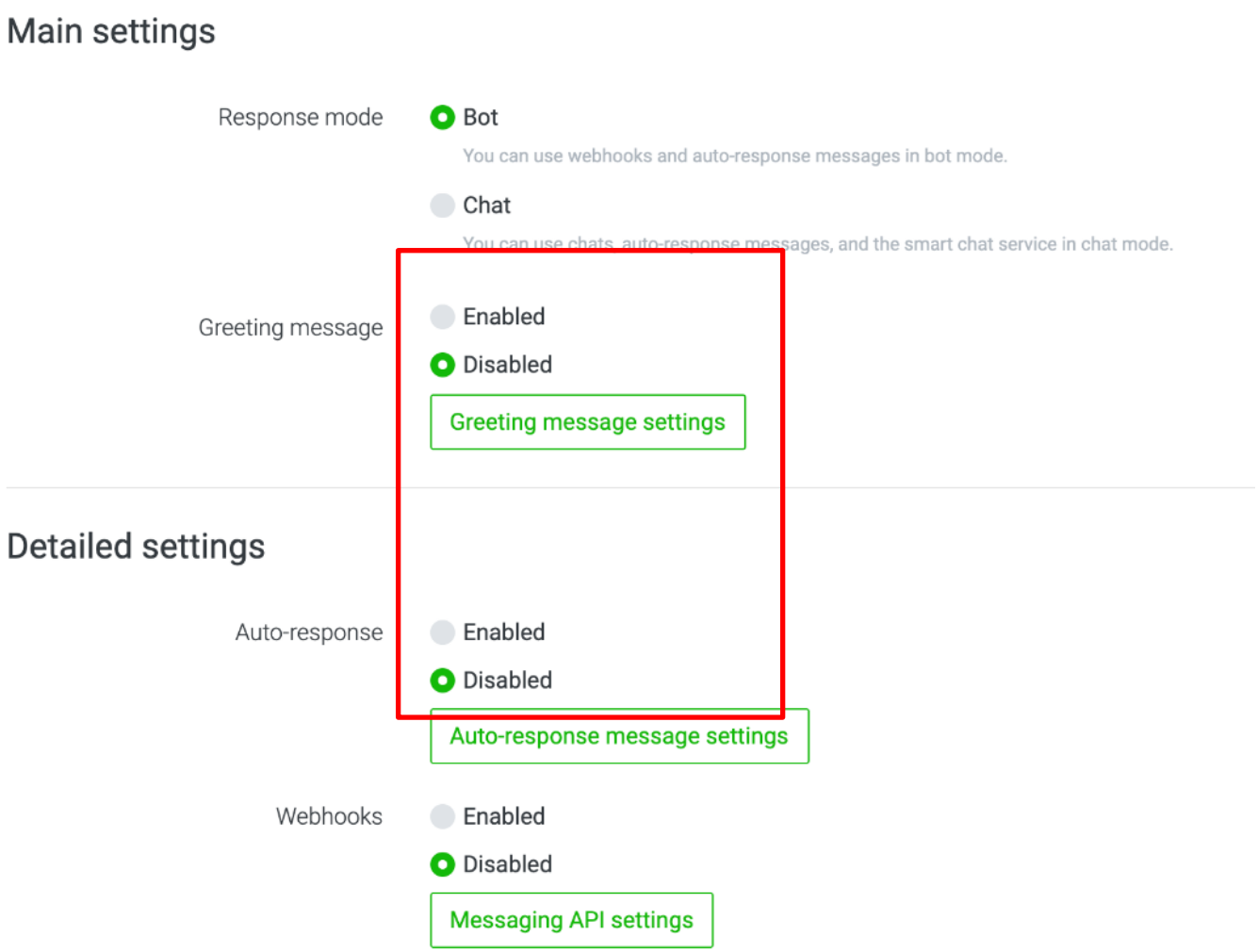
LINE Bot Image



Message API

- Change to Message API page
- The default value to the fields of Auto-reply messages and Greeting messages are Enabled， this means Line Bot will automatically send welcome and reply messages.
- Since these messages are usually designed based on specific needs, click the 'Edit' button to the right of the 'Auto-reply messages' and 'Greeting messages' sections and set both fields to 'Disable'.

Message API


Main settings
Response mode
Bot
You can use webhooks and auto-response messages in bot mode.
Chat
You can use chats, auto-response messages, and the smart chat service in chat mode.
Greeting message
Enabled
Disabled
Greeting message settings
Detailed settings
Auto-response
Enabled
Disabled
Auto-response message settings
Webhooks
Enabled
Disabled
Messaging API settings

Add the LINE Bot as a friend

- After creating the LINE Bot on the LINE Developer page, users can add the LINE Bot as a friend in the LINE app.
- Use the QR code in the Message API section.

QR code

Send message to LINE Bot

- Since the auto-reply function has been turned off earlier, LINE Bot will not automatically respond to messages. 。
- But you can see the 'Read' indicator, indicating that LINE Bot has successfully read the messages we sent 。

Outline

- Create LINE Bot
- 「Parrot」LINE bot
- Graphic LINE Bot
  - Create Rich Menu
- LINE Bot basic interaction
  - Return Text
  - Return Image
  - Return Sticker
  - Mutiple Messages
  - Return Location
  - Quick Replies

「Parrot」LINE bot

- The simplest example of interaction between a LINE Bot and a user is sending a message to the LINE Bot, and the LINE Bot responding with the same message, just like a parrot mimicking speech. This is often referred to as a 'parrot' LINE Bot."

To obtain the necessary information for LINE Bot API program

- Channel secret and Channel access token is all you need, in order for the API to function properly.
- Open LINE Bot (default: Basic Setting), and record the value in the Channel Secret field for future use.
- If this value happens to be accidentally known by others, you can generate a new Channel Secret value by clicking the 'Issue' button on the right.

Channel Access Token

- When creating a LINE Bot, it won't be generated automatically by default. Please click the 'Issue' button on the right

- Record the generated Channel Access Token value.

Install LINE Bot SDK

- To enable interaction between LINE Bot and users using the LINE Bot API, it is necessary to install the LINE Bot SDK.
- To install the LINE Bot API, run it in the command line window:
  - *pip install line-bot-sdk*

```
(base) linjiayude-MacBook-Pro:~ sallylin$ pip install line-bot-sdk==1.8.0
Collecting line-bot-sdk==1.8.0
  Downloading line_bot_sdk-1.8.0-py2.py3-none-any.whl (44 kB)
     |████████████████████████████████| 44 kB 436 kB/s
Requirement already satisfied: future in /opt/anaconda3/lib/python3.8/site-packa
ges (from line-bot-sdk==1.8.0) (0.18.2)
Requirement already satisfied: requests>=2.0 in /opt/anaconda3/lib/python3.8/sit
e-packages (from line-bot-sdk==1.8.0) (2.24.0)
Requirement already satisfied: idna<3,>=2.5 in /opt/anaconda3/lib/python3.8/site
-packages (from requests>=2.0->line-bot-sdk==1.8.0) (2.10)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /opt/a
naconda3/lib/python3.8/site-packages (from requests>=2.0->line-bot-sdk==1.8.0) (
1.25.9)
Requirement already satisfied: certifi>=2017.4.17 in /opt/anaconda3/lib/python3.
8/site-packages (from requests>=2.0->line-bot-sdk==1.8.0) (2020.6.20)
Requirement already satisfied: chardet<4,>=3.0.2 in /opt/anaconda3/lib/python3.8
/site-packages (from requests>=2.0->line-bot-sdk==1.8.0) (3.0.4)
Installing collected packages: line-bot-sdk
Successfully installed line-bot-sdk-1.8.0
```

To create a website using Flask

- To use LINE Bot, you need to set up a web server, and in this case, we are using the Flask framework.
- When installing Anaconda, the Flask module is already included by default, so there is no need to install it separately.

The basic structure of a Flask application

```
from flask import Flask
app = Flask(__name__)

路由一
路由二
.......

if __name__ == '__main__':
    app.run
```

Import Flask module
Create Flask object

Run this Flask program

Creating routes

- "Routes" are the backbone of a Flask application, and the syntax for creating routes is as follows:

```
@app.route('網頁路徑')
def 函式名稱():
    處理程式
```

- The "@" symbol, known as a decorator, is used to associate a specific URL route with the subsequent function in Flask. In other words, when you enter the URL associated with the decorator in your web browser's address bar, it will trigger the execution of the function defined in the next line.
- The "web page path" is the portion of the website's URL that comes after the domain or host address. For example, if the host address is "http://127.0.0.1:5000" and the web page path is "/append," then the complete web page address or URL would be "http://127.0.0.1:5000/append"
- The "function name" can be arbitrarily chosen when defining view functions in Flask. However, it is a common convention to use function names that are related to the web page path they correspond to.

Hello!Flask!

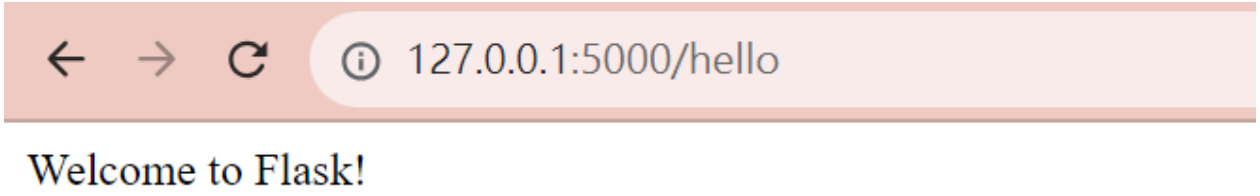
```python
from flask import Flask
app = Flask(__name__)

@app.route('/hello')
def index():
    return 'Welcome to Flask!'

if __name__ == '__main__':
    app.run()
```

Result :

```
* Serving Flask app "__main__" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off

* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Access a website in a browser

- http://127.0.0.1:5000/hello



- If the server is currently running, you can terminate its execution by pressing "Ctrl+C."

Multiple URLs to map to the same function

- Sometimes websites need different URLs to display the same content. For example, typically entering the server address "http://127.0.0.1:5000/hello" or adding "/index" will both display the homepage.
- The routing syntax for mapping multiple URLs to the same function is as follows：

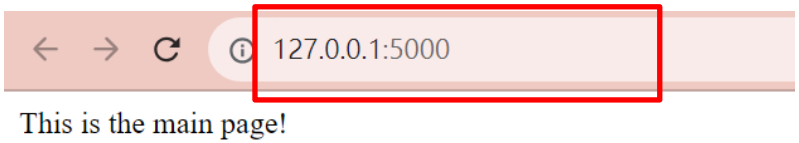
```
@app.route('網頁路徑一')
@app.route('網頁路徑二')
...
def 函式名稱():
    處理程式
```

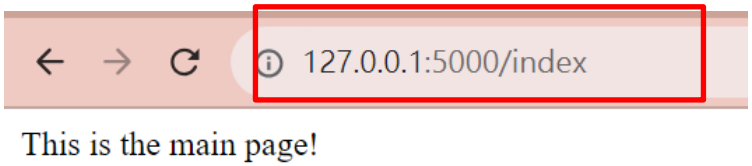
Multiple URLs to map to the same function

```python
from flask import Flask
app = Flask(__name__)
@app.route('/')
@app.route('/index')
def index():
    return 'This is the main page!'

if __name__ == '__main__':
    app.run()
```

web page path '/' or '/index' will both execute function index

- When you run the program, entering the server address "http://127.0.0.1:5000/" or "http://127.0.0.1:5000/index" in your browser will display the homepage.





Passing Path Parameters (1/2)

- Most web pages are not static; their content may need to change dynamically. In such cases, parameters can be sent to the web page through routing.
- The syntax for passing parameters through routing is as follows:

```
@app.route('網頁路徑/<資料型態一:參數一>/<資料型態二:參數二>/.....')
```

- The parameters are enclosed in "<" and ">".

Passing Path Parameters(2/2)

- The data types provided by Flask are as follows:

Data types	Description
string	default value
int	integers
float	floating-point numbers.
path	path names that include "/" characters.

- The data type of a parameter can be omitted, and the default value is "string."

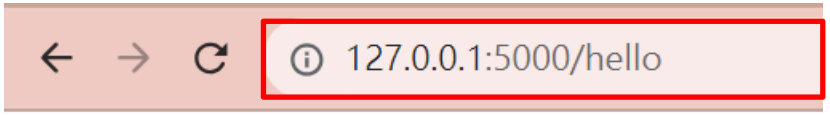
```
#傳遞字串型態參數「name」到hello網址：
@app.route('/hello/<string:name>')
def hello(name):
    處理程式
```

Passing Path Parameters Example

```python
#Passing Path Parameters Example
from flask import Flask
app = Flask(__name__)

@app.route('/hello/<name>')
def index(name):
    return '{}, Welcome to the main page!'.format(name)

if __name__ == '__main__':
    app.run()
```

- When running the program, entering the URL "http://127.0.0.1:5000/hello/Daniel" in your browser will display the parameter "Daniel" on the web page.
- Note: If parameters are set in the route, the URL must have matching parameter values.

Daniel, Welcome to the main page!



Not Found

linebot-parrot.py

```python
app = Flask(__name__)

configuration = Configuration(access_token='User Channel access token')
handler = WebhookHandler('User Channel secret')


@app.route("/callback", methods=['POST'])
def callback():
    # get X-Line-Signature header value
    signature = request.headers['X-Line-Signature']

    # get request body as text
    body = request.get_data(as_text=True)
    app.logger.info("Request body: " + body)

    # handle webhook body
    try:
        handler.handle(body, signature)
    except InvalidSignatureError:
        app.logger.info("Invalid signature. Please check your channel access token/channel secret.")
        abort(400)

    return 'OK'


@handler.add(MessageEvent, message=TextMessageContent)
def handle_message(event):
    with ApiClient(configuration) as api_client:
        line_bot_api = MessagingApi(api_client)
        line_bot_api.reply_message_with_http_info(
            ReplyMessageRequest(
                reply_token=event.reply_token,
                messages=[TextMessage(text=event.message.text)]
            )
        )

if __name__ == "__main__":
    app.run()
```

Setup Channel secret and Channel access token

Create a callback route to check if the LINE Bot data is correct

If a user sends a message, send back the received text message.

Webhook

- Use a webhook URL as the server connection for LINE Bot
- What is Webhook? Webhook is a method that allows one website to subscribe to events on another website.
- Subscription: When certain events occur, the notifier informs the subscriber that the event has happened. Since the subscriber has subscribed, it implies that upon receiving the event, the subscriber is expected to take certain actions. 。

https://itherp.ithome.com.tw/articles/10193212

Line Messaging API Webhook

- Subscription phase: We input communication address in the LINE Developer backend and then submit.
- Notification phase: When someone sends a message to the group where KAMI Dog is on LINE, a notification will be received. LINE will send such messages to the communication address we have set up.
- Action phase: When we responds to a group message, KAMI Dog will send the message to LINE, and then LINE will help us send it to the group.

https://ithelp.ithome.com.tw/articles/10193212

The communication protocol for Webhooks(1/2)

- There are many methods of communication between websites, and Webhook uses the HTTP protocol.
- HTTP: Sending an HTTP Request is like sending a postcard. The postcard travels through the network, and every post office and mail carrier that handles it can see the content you've written. The recipient must acknowledge receipt of each postcard as it arrives and immediately send a reply, which is the HTTP Response. In the end, the sender receives the reply, completing one round of communication.

https://ithelp.ithome.com.tw/articles/10193212

The communication protocol for Webhooks (2/2)

https://ithelp.ithome.com.tw/articles/10193212

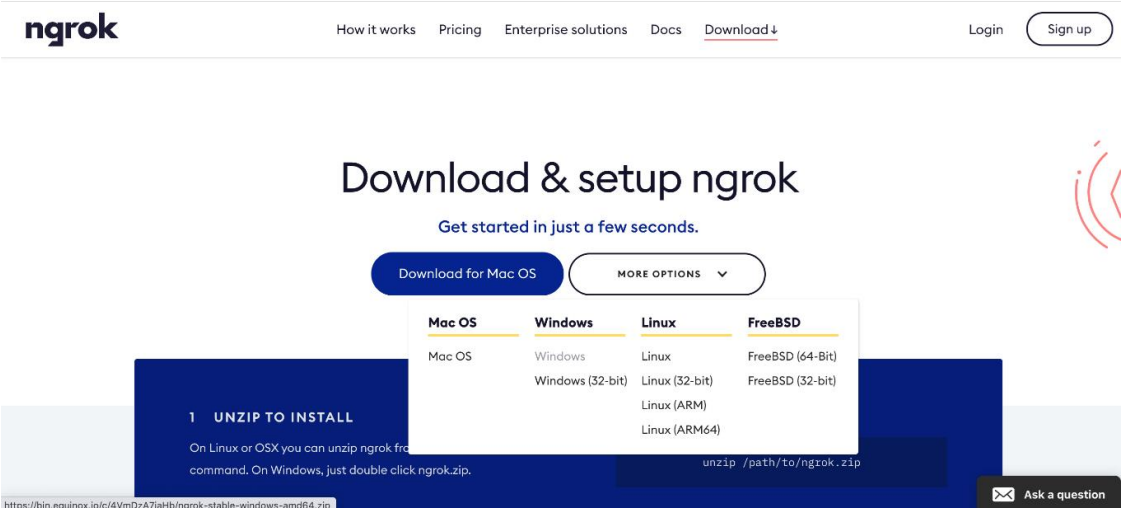
- **The HTTP protocol has security issues.**
- This type of communication is insecure. If any of the intermediary post offices or mail carriers are malicious, they can forge the content of the letters.
- Similar fraudulent practices exist on the internet. So, how can we ensure security? By using a secure HTTP protocol.
- **HTTPS Protocol:** The HTTPS protocol is a secure version of the HTTP protocol. It’s almost the same as HTTP, the difference is that it transmits encrypted letters instead of postcards, making it difficult for intermediary post offices or mail carriers to understand the content.
- The sender's signature is authenticated, ensuring the sender's identity, so there is no need to worry about receiving forged messages.

Setting up an HTTPS server using ngrok

- Using a webhook URL as the server connection for the LINE Bot. The webhook URL has two requirements:
  - It must be a URL (cannot be an IP address)
  - The communication protocol is "https"
    - Based on data communication security considerations,all data transmission for developing application services on the LINE platform must be done through encrypted channels. Therefore, when developers set up a webhook server for the LINE Messaging API，it is imperative to use the HTTPS communication protocol.
- ngrok is a proxy server that can create a secure external channel for a local web server, allowing communication between the internal server and the outside world.
  - It can not only establish an HTTP server but also an HTTPS server, fully meeting the requirements of a LINE Bot server

Download the ngrok user system compression file.

- https://ngrok.com/download Download the user system's compressed file.
  - Choose your corresponding operating system to download



- After extracting, the ngrok executable file will be generated. Copy this executable file to the folder whe <linebotTest1.py> program is located.

ngrok

ngrok-stable-darwin-amd64.zip

Execute linebotTest1.py

```python
app = Flask(__name__)

configuration = Configuration(access_token='User Channel access token')
handler = WebhookHandler('User Channel secret')


@app.route("/callback", methods=['POST'])
def callback():
    # get X-Line-Signature header value
    signature = request.headers['X-Line-Signature']

    # get request body as text
    body = request.get_data(as_text=True)
    app.logger.info("Request body: " + body)

    # handle webhook body
    try:
        handler.handle(body, signature)
    except InvalidSignatureError:
        app.logger.info("Invalid signature. Please check your channel access token/channel secret.")
        abort(400)

    return 'OK'


@handler.add(MessageEvent, message=TextMessageContent)
def handle_message(event):
    with ApiClient(configuration) as api_client:
        line_bot_api = MessagingApi(api_client)
        line_bot_api.reply_message_with_http_info(
            ReplyMessageRequest(
                reply_token=event.reply_token,
                messages=[TextMessage(text=event.message.text)]
            )
        )

if __name__ == "__main__":
    app.run()
```

Note: When MacOS is updated to version 12 (Monterey), the Airplay function will occupy port 5000. However, Flask defaults to port 5000. This can cause a 403 Forbidden error with ngrok. To avoid this, specify a different port (e.g., 5001) when using ngrok.

assign port number->app.run(port=xxxx)

```
* Serving Flask app "__main__" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
```

The default port for Flask is 5000.

```
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

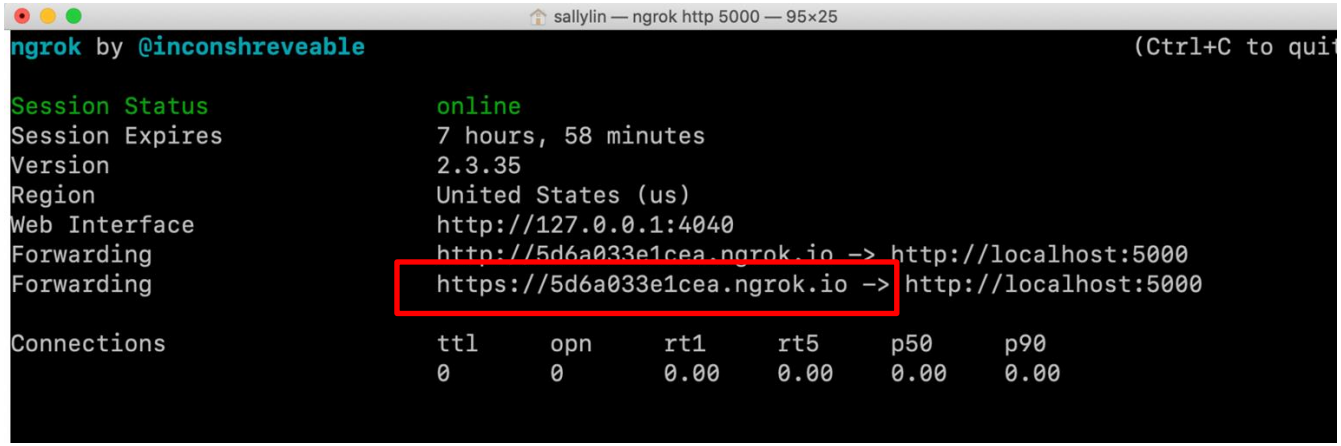
Starting the ngrok server (1/2)

- The syntax to start the ngrok server is:

```
#syntax to start the ngrok server
ngrok http port number
```

Starting the ngrok server (2/2)

- The default port for Flask is 5000.
- For macOS versions starting from Monterey, specify a port other than 5000.
- Open the command line and execute "ngrok http 5000."
  - *mac: ./ngrok http 5000*
- Starting the ngrok server



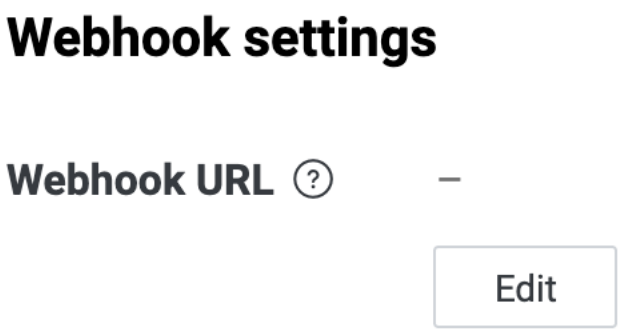
- Take note of the URL. (Note: The URL will change when the ngrok server is restarted.)

Setting up the LINE Bot's Webhook URL

- After setting up the ngrok server, you need to set the LINE Bot's Webhook URL to the ngrok server's HTTPS server URL. This way, the LINE Bot will be able to respond to user messages.

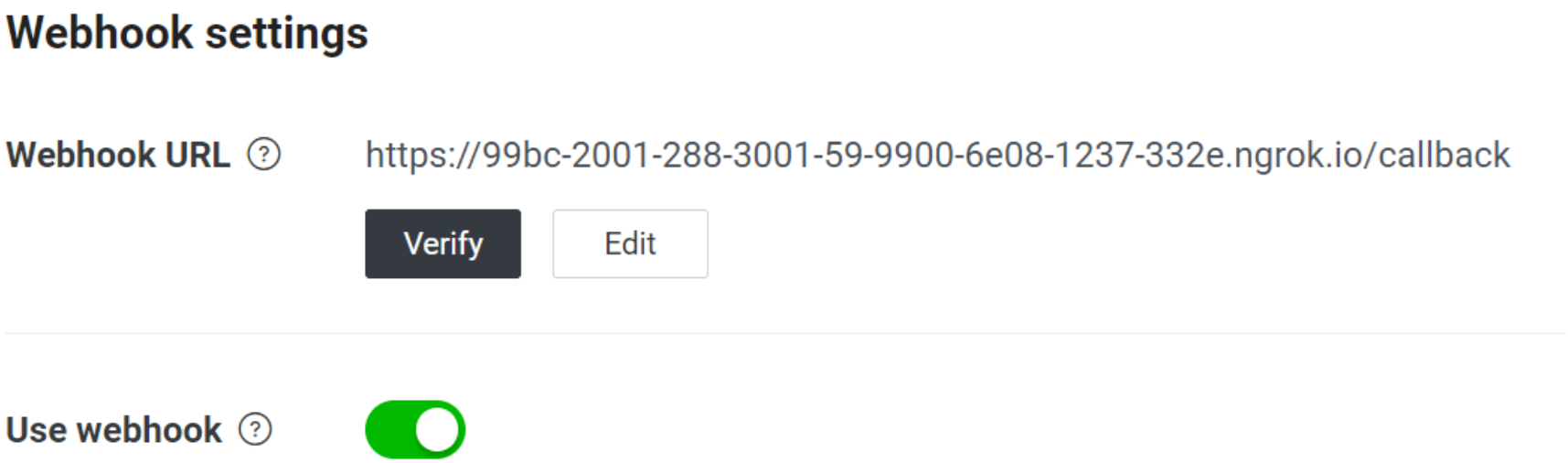
Configure it in the Message API(1/2)

- When you open the Message API tab on the LINE Bot settings page, the Webhook URL for the LINE Bot is not set by default.
- Click the "Edit" button to modify the settings.



Configure it in the Message API (2/2)

- Enter the ngrok server's HTTPS server URL followed by "/callback".
- Change the webhook to "Enabled."



Finish Parrot LINE Bot

Outline

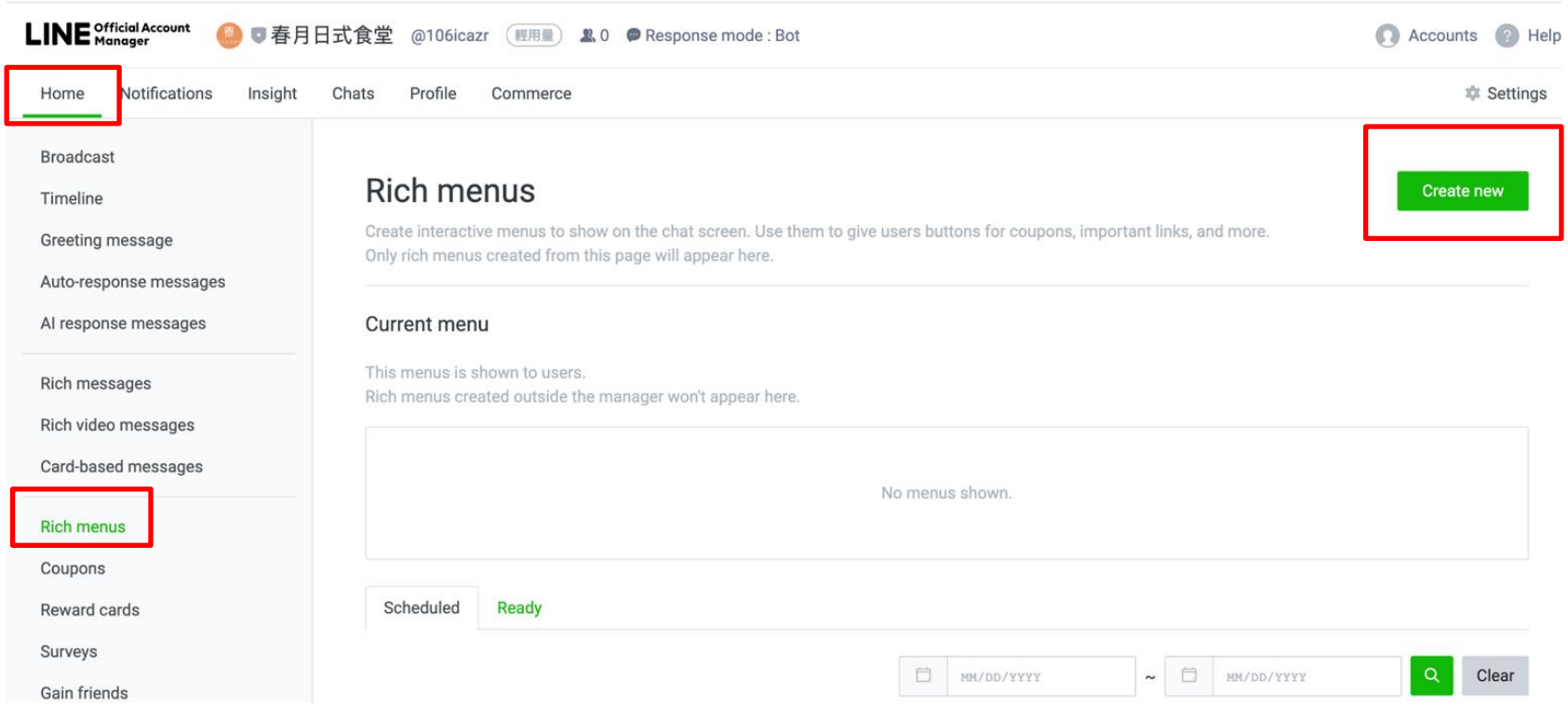
- Create LINE Bot
- 「Parrot」LINE bot
- Graphic LINE Bot
  - Create Rich Menu
- LINE Bot basic interaction
  - Return Text
  - Return Image
  - Return Sticker
  - Mutiple Messages
  - Return Location
  - Quick Replies

To create a rich menu

- Rich menu allows the LINE Bot to have menu functionality similar to a mobile app.
- When the user clicks on a menu item icon, it will execute the specified function.

Homepage

- On the homepage, click "Rich Menu."
  (Rich menu ， Not Rich Message)
- Click the "Create" button in the upper right corner.



Display settings (1/2)

- Title：The name displayed when clicking the rich menu item on the LINE Bot management page.
- Usage Period：Set the effective period for the rich menu, indicating when the rich menu will be active.
- Menu Bar Display Text：The menu name text displayed when the LINE Bot is running on a mobile device, with the default value being "Menu."
- Default Display Method：Whether to display the rich menu when the user opens the LINE Bot on a mobile device.

Display settings (2/2)

Rich menu

Create interactive menus to show on the chat screen. Use them to give users buttons for coupons, important links, and more.

Save draft

Save

Display settings

Title: 測試圖文選單 6/30

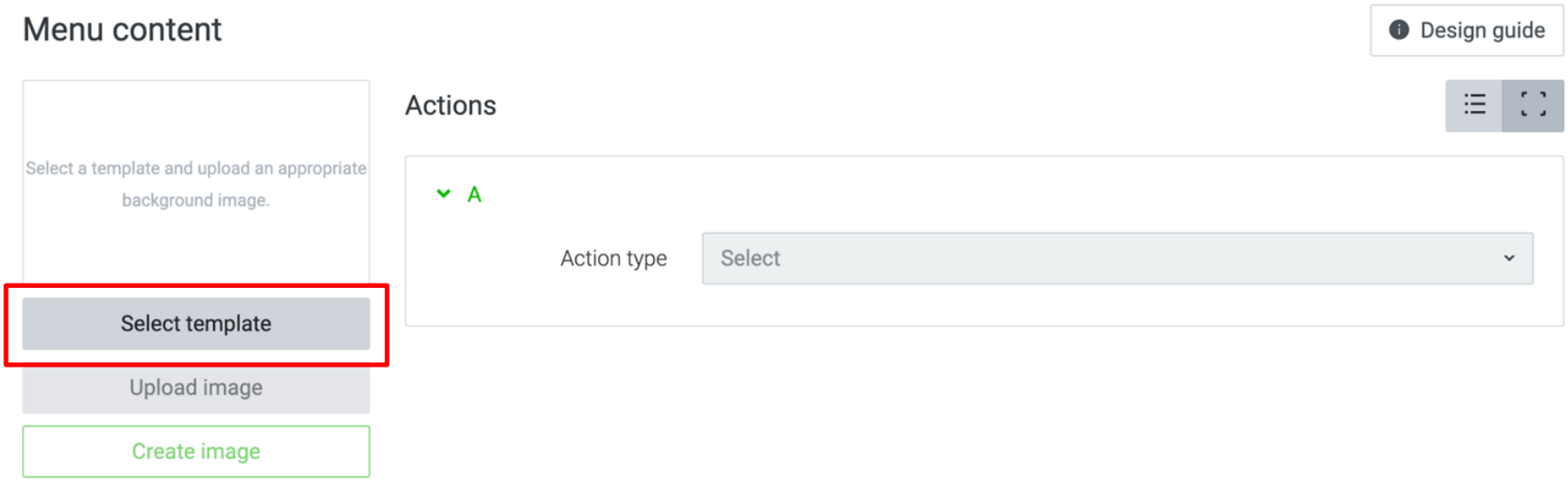
Display period: 08/17/2020 00:00 ~ 08/23/2020 23:59 Clear

Menu bar label ⓘ
- Menu
- Custom label: 圖文選單 4/14

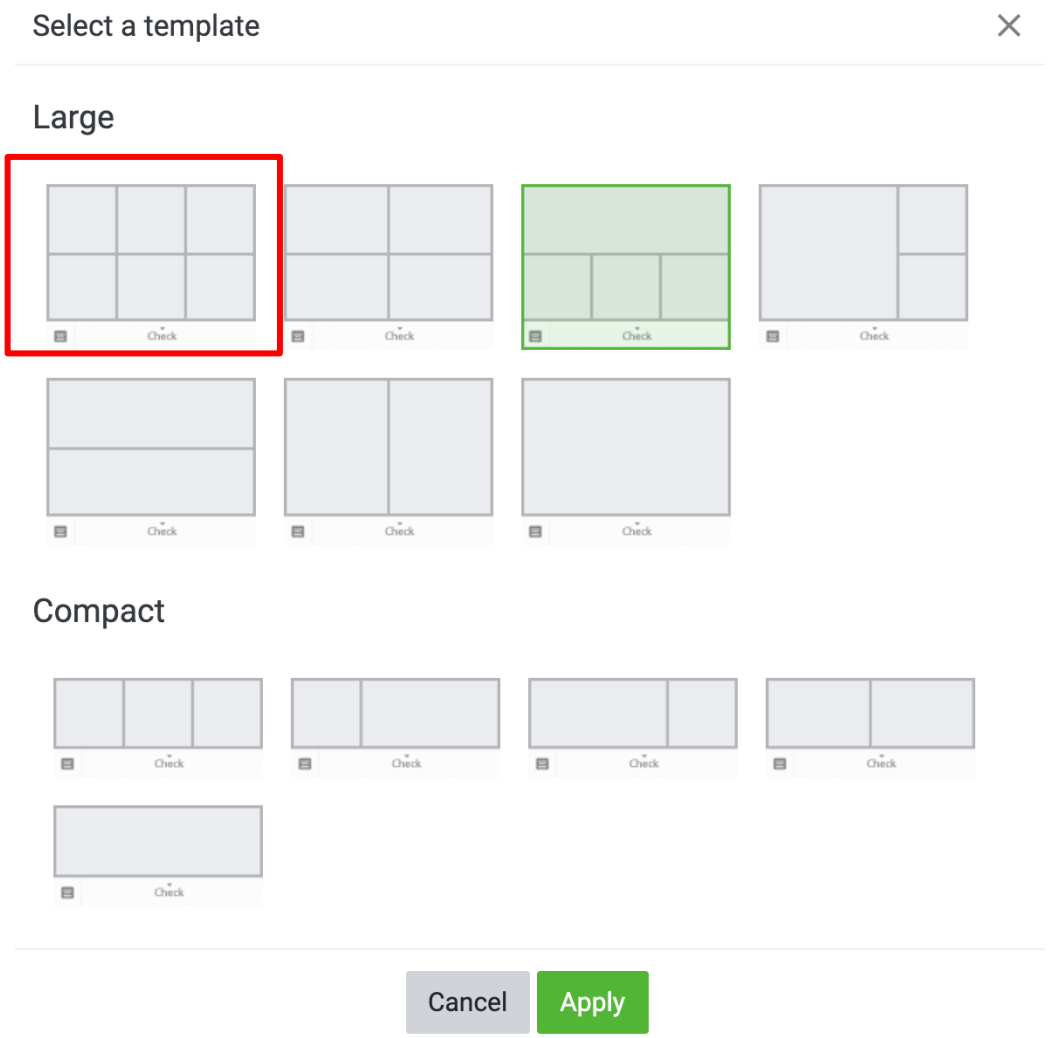
Default behavior ⓘ
- Shown
- Collapsed

Content Settings (1/2)

- Click on "Select Template." The templates are divided into "Large" and "Small."
- Choose the appropriate template and then apply it. ◦

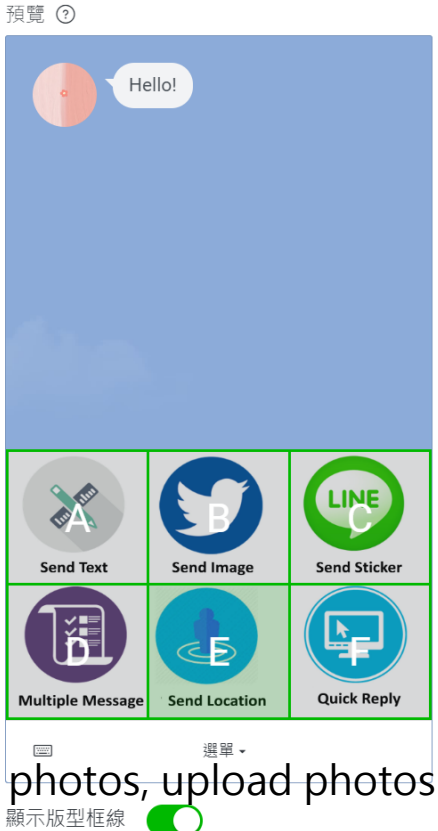


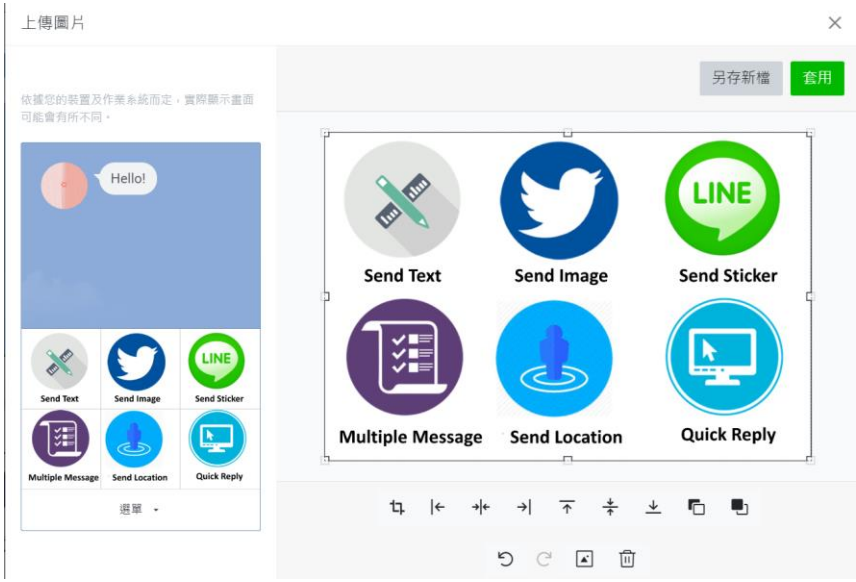
Content Settings (2/2)



Upload the background image

- Upload the images for the rich menu.
- The uploaded images must adhere to the specified dimensions. If there are no specified dimensions for the images, you can choose "Create Image" to generate the images. 。

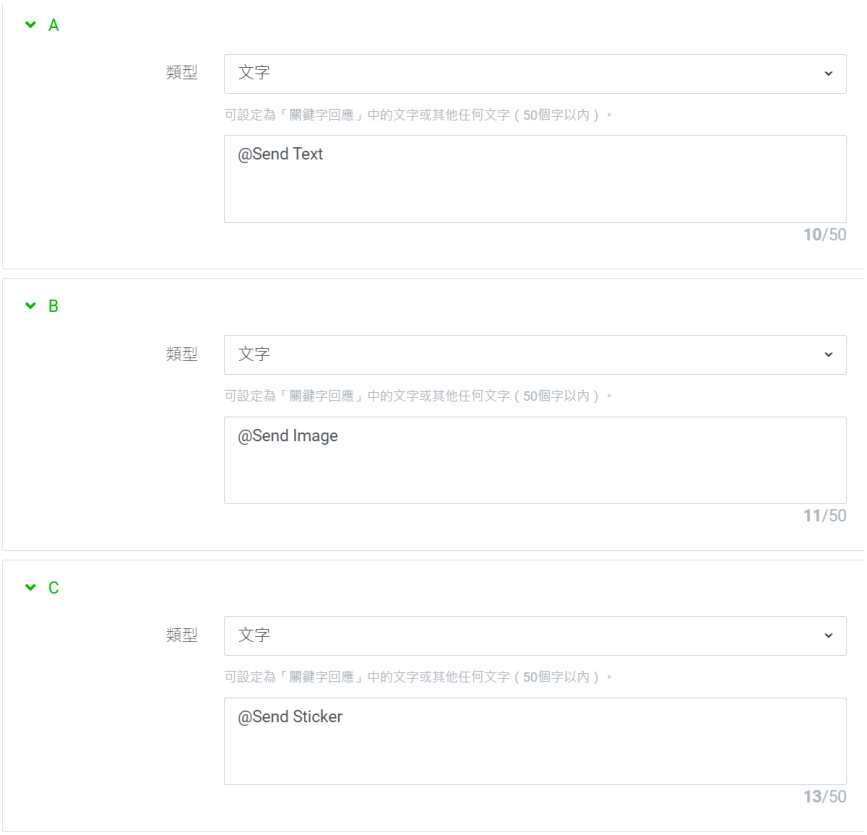


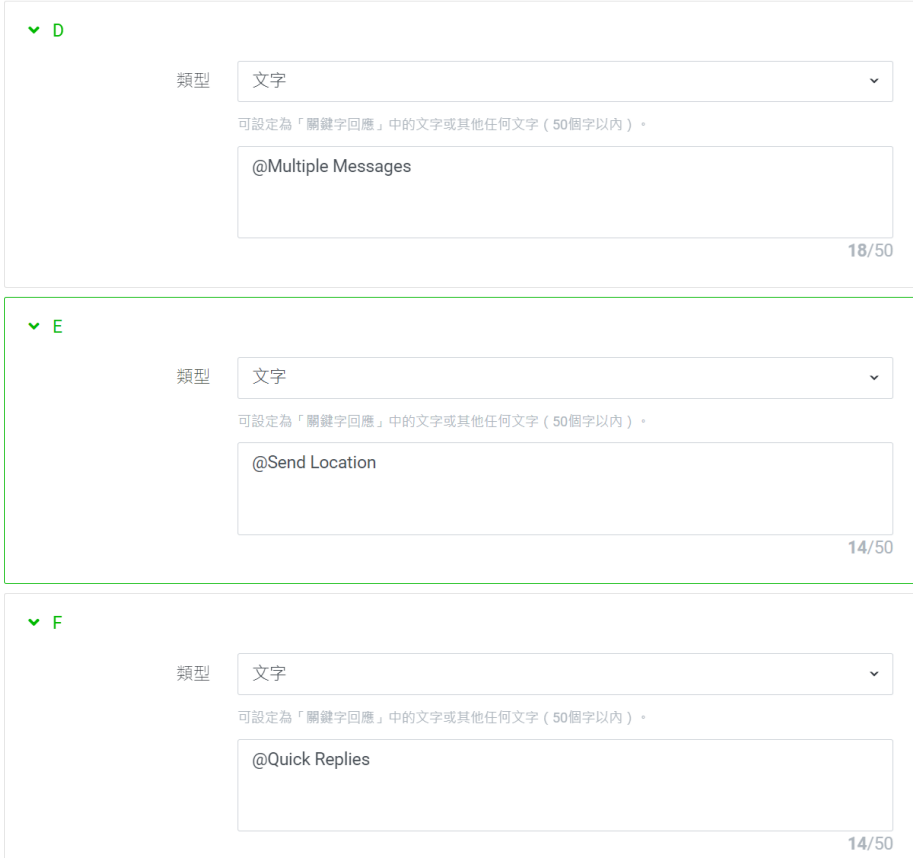


Upload the photos, upload photos for all three positions at once, and click "Apply."

Set the actions for the rich menu

- The "Action" field on the right allows you to set the actions to be executed within the rich menu.





The completed rich menu after configuration

- After saving, you will see the rich menu you just configured.

Rich menus

Create new

Create interactive menus to show on the chat screen. Use them to give users buttons for coupons, important links, and more.
Only rich menus created from this page will appear here.

Current menu

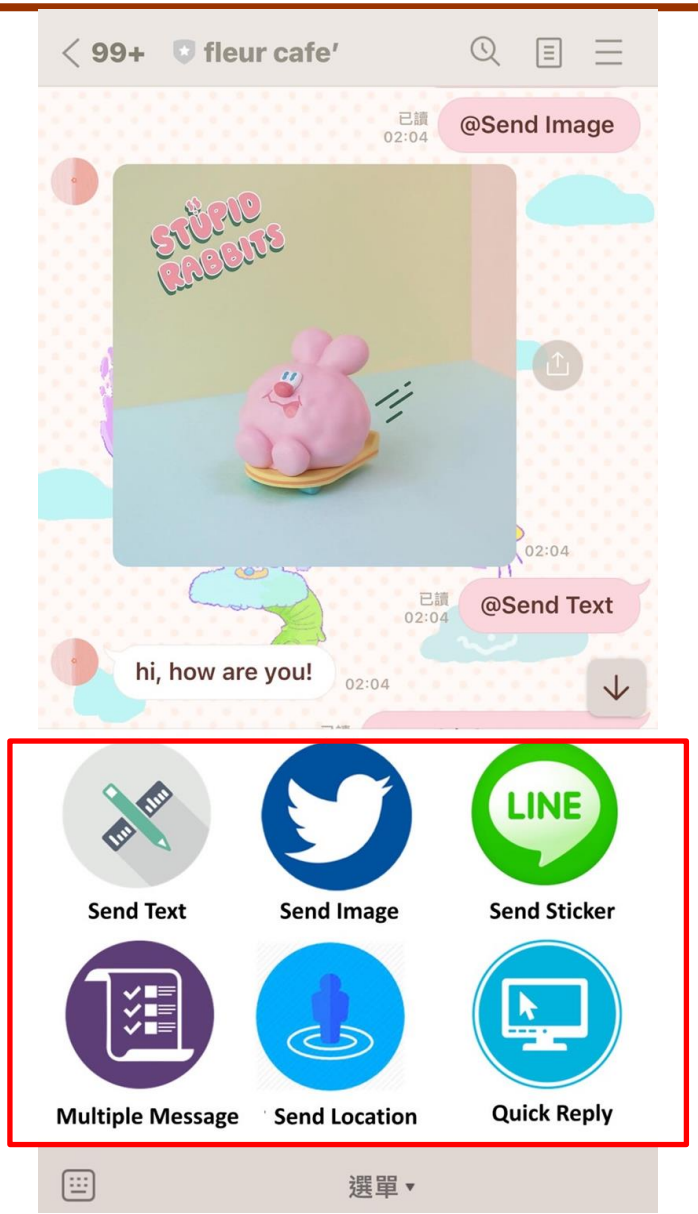
This menus is shown to users.
Rich menus created outside the manager won't appear here.

Title	測試圖文選單
Display period	08/17/2020 00:00 - 08/23/2020 23:59
Action	• Link - https://japanese-restaurant-6365.business.site/?utm_source=gmb&utm_... • Coupon - 測試用優惠券 • Text - @由圖文選單輸入文字

Scheduled Ready

MM/DD/YYYY ~ MM/DD/YYYY Clear

The rich menu is completed!



Outline

- Create LINE Bot
- 「Parrot」LINE bot
- Graphic LINE Bot
  - Create Rich Menu
- LINE Bot basic interaction
  - Return Text
  - Return Image
  - Return Sticker
  - Multiple Messages
  - Return Location
  - Quick Replies

LINE Bot basic interaction

- The LINE Bot SDK provides various APIs that allow developers to interact with users through code. One of the most commonly used functionalities is providing appropriate responses upon receiving user messages.

Creating routes (1/2)

- When a user sends a message to the LINE Bot, it triggers a MessageEvent. Here, we'll handle only received text messages.
- The syntax for creating a route：

```
#create route
@handler.add(MessageEvent, message = TextMessage)
```

- "message = TextMessage" indicates that the route will handle incoming text messages.

Creating routes (2/2)

- Let's create a function to handle the route:

```
#create route
def FunctionName(event):
```

- The event parameter contains various pieces of information, including the messages received.

  For example, if you have a function named handle_message:

```
#handle_message
def handle_message(event):
```

To retrieve the text sent by the user

- The first step in a text processing program is to obtain the text sent by the user. The syntax is:

```
text = event.message.text
```

- For example, to retrieve the text sent by the user and store it in the variable "mtext":

```
mtext = event.message.text
```

Basic syntax for LINE Bot interactive features

- Based on the steps outlined above, the following is the basic syntax for LINE Bot interactive features

```
#LINE Bot interactive features
@handler.add(MessageEvent, message=TextMessageContent)
def handle_message(event):
    mtext = event.message.text
        if mtext == 'Text1':
            'deal with program 1'
        if mtext == 'Text2':
            'deal with program 2'
        ......
```

Return message

- The types of messages for message replies include Text, Image, Location, Sticker, Audio, Video, and Template.

Return message

- The syntax for returning a message is:

```
#The syntax for returning a message
line_bot_api.reply_message(event.reply_token, 'message type')
```

- The "message type" in the above syntax is composed of a message command and its parameters. The syntax is:

```
#Syntax of message type
MessageCommand(parameter1=value1, parameter2=value2, ...)
```

Return a text message

- The syntax for sending a simple text message using TextSendMessage is:

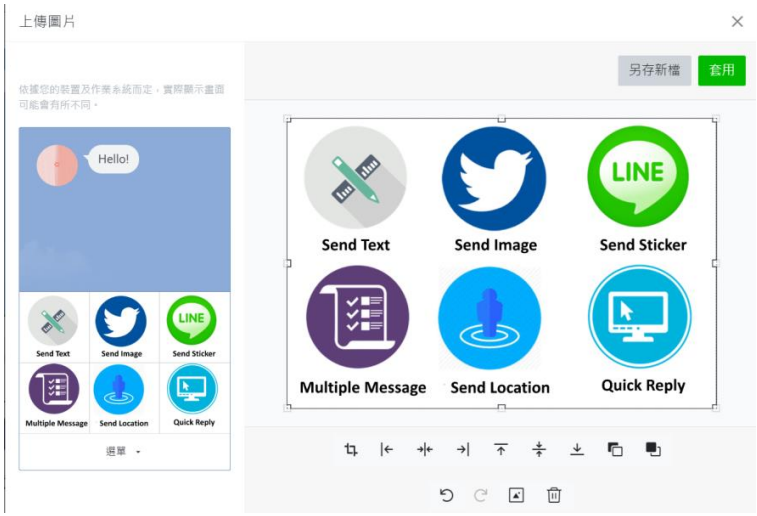
```
#return a text message
line_bot_api.reply_message_with_http_info(
    ReplyMessageRequest(reply_token=event.reply_token,messages=[TextMessage(text= "Content of Text Message")]))
```

Create Messaging API Channel (1/2)

- Following the steps outlined earlier, create another LINE Bot by selecting the "Messaging API Channel"
- Execute the response functions using a rich menu.

Create Messaging API Channel (2/2)

- Create a six-grid rich menu in the previously established LINE Bot.
- All six items are set to "Text," and the respective texts to be sent are as follows:
  - @Send Text
  - @Send Image
  - @Send Sticker
  - @Multiple Messages
  - @Send Location
  - @Quick Replies
  - Be awaqre with full-width and half-width "@" symbols



- Here are the texts for the six items with the "@" symbol at the beginning to distinguish them from user-inputted text

Create a Flask program (1/3)

```python
from flask import Flask, request, abort

from linebot.v3 import (
    WebhookHandler
)
from linebot.v3.exceptions import (
    InvalidSignatureError
)
from linebot.v3.messaging import (
    Configuration,
    ApiClient,
    MessagingApi,
    ReplyMessageRequest,
    TextMessage,
    ImageMessage,
    StickerMessage,
    LocationMessage,
    QuickReply,
    QuickReplyItem,
    MessageAction,
)
from linebot.v3.webhooks import (
    MessageEvent,
    TextMessageContent
)
```

import library:

```
from linebot.v3.messaging import (
    Configuration,
    ApiClient,
    MessagingApi,
    ReplyMessageRequest,
    TextMessage,
    ImageMessage,
    StickerMessage,
    LocationMessage,
    QuickReply,
    QuickReplyItem,
    MessageAction,
)
```

Create a Flask program (2/3)

- The previously taught callback functionality for the Parrot LINE Bot.

```
@app.route("/callback", methods=['POST'])
def callback():
    # get X-Line-Signature header value
    signature = request.headers['X-Line-Signature']

    # get request body as text
    body = request.get_data(as_text=True)
    app.logger.info("Request body: " + body)

    # handle webhook body
    try:
        handler.handle(body, signature)
    except InvalidSignatureError:
        app.logger.info("Invalid signature. Please check your channel access token/channel secret.")
        abort(400)

    return 'OK'
```

Create a Flask program (3/3)

- To make the LINE Bot respond to the various functions selected by the user in the rich menu.
- Provide appropriate responses based on the content of the message. Here is the code for "@Send Text"

```
@handler.add(MessageEvent, message=TextMessageContent)
def handle_message(event):
    mtext = event.message.text
    with ApiClient(configuration) as api_client:
        line_bot_api = MessagingApi(api_client)
        if mtext == "@Send Text":
            try:
                line_bot_api.reply_message_with_http_info(
                ReplyMessageRequest(
                reply_token=event.reply_token,
                messages=[TextMessage(text= "hi, how are you!")]
                    )
                )
            except:
                line_bot_api.reply_message_with_http_info(
                ReplyMessageRequest(
                reply_token=event.reply_token,
                messages=[TextMessage(text= "error")]
                    )
                )
```

To handle input text messages

Check if it is "@Send Text"

LINE Bot return text message

To handle errors that occur in your LINE Bot , respond with an "error" message.

Run the Flask program (local server)

- Run the Flask program

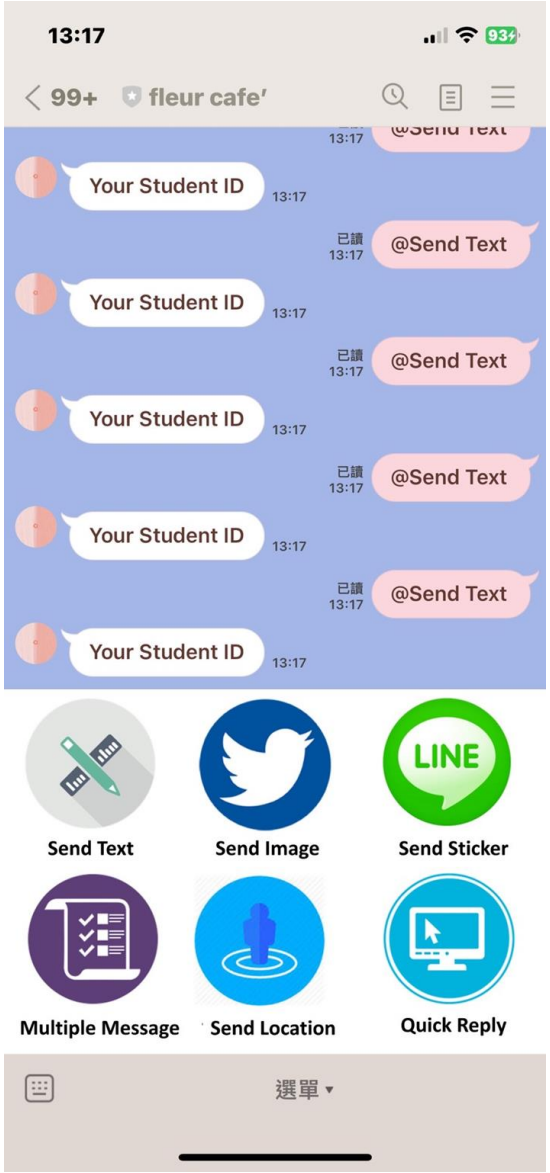
```
* Serving Flask app "__main__" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
```

```
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Execute the ngrok server

- Run the ngrok server.
- Set the LINE Bot's Webhook URL to the ngrok server's HTTPS server URL followed by '/callback'.

Send Text Message - Completed!



Outline

- Create LINE Bot
- 「Parrot」LINE bot
- Graphic LINE Bot
  - Create Rich Menu
- LINE Bot basic interaction
  - Return Text
  - Return Image
  - Return Sticker
  - Multiple Messages
  - Return Location
  - Quick Replies

Return Image Message(1/4)

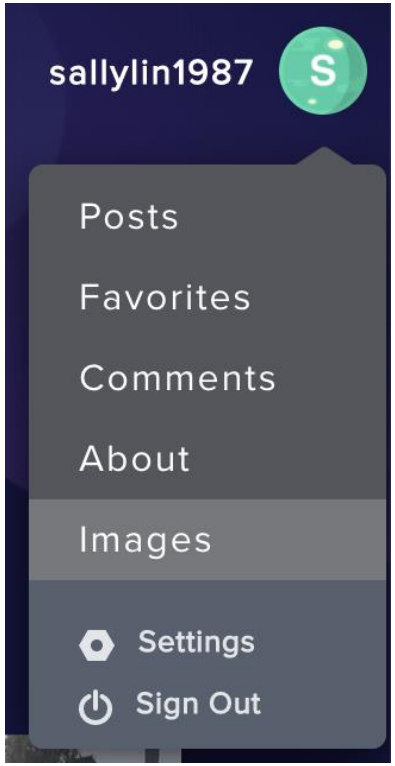
- The syntax for returning an image is:

```
#return a image message
line_bot_api.reply_message_with_http_info(
    ReplyMessageRequest(reply_token=event.reply_token,messages=[
        ImageMessage(
        original_content_url = "Original Image URL",
        preview_image_url = "Preview Image URL")]))
```

- The command to return an image message is "ImageMessage".

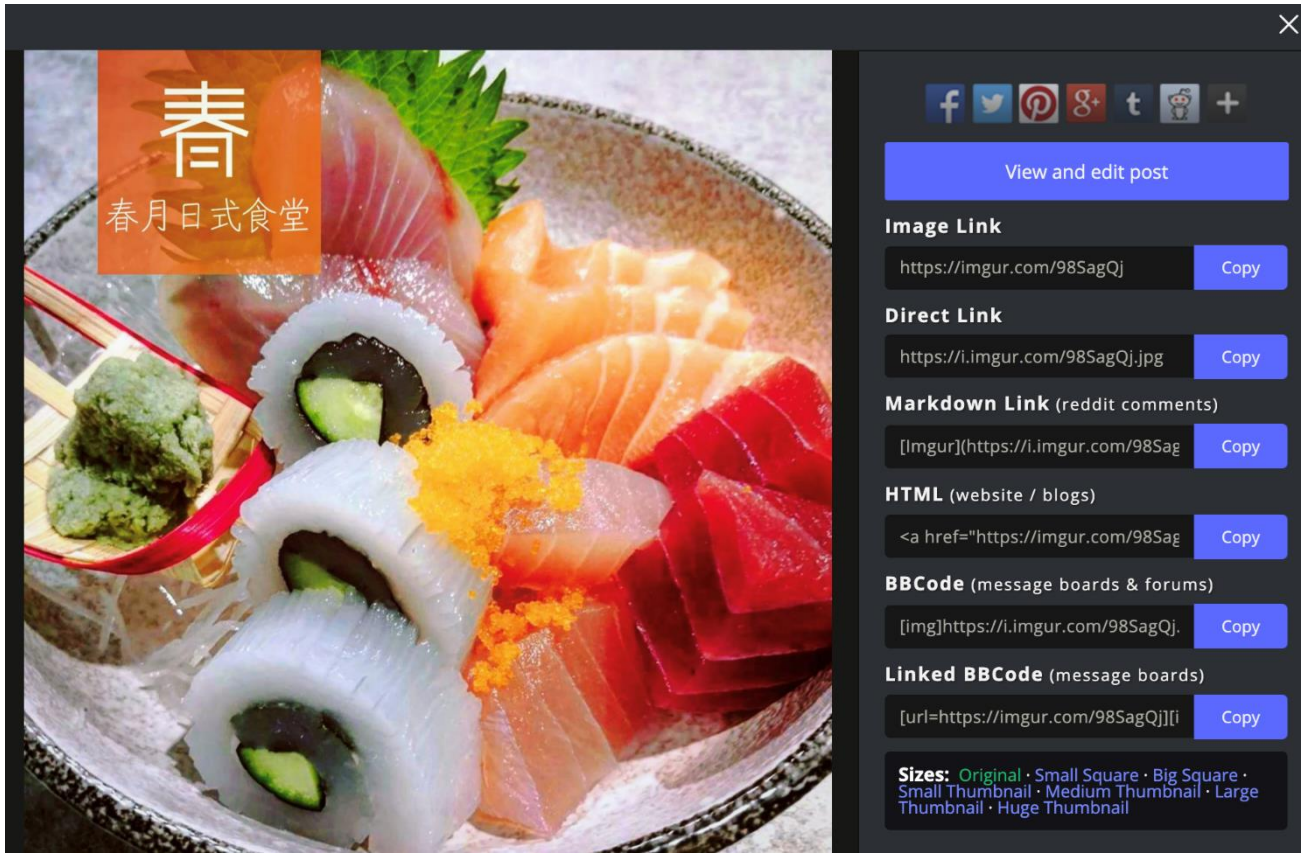
Return Image Message(2/4)

- Images that are typically sent are usually first uploaded to the cloud.
- You can use https://imgur.com/.
- Go to "images" under your account to view the uploaded images.
- Click on "Add Images" in the top right corner.



Return Image Message(3/4)

- After adding the image, click on the image and select "Direct Link."



Return Image Message(4/4)

- Fill the image URLs into the provided syntax.

The command of return the image is "ImageMessage"

```
elif mtext == "@Send Image":
    try:

        line_bot_api.reply_message_with_http_info(
        ReplyMessageRequest(
        reply_token=event.reply_token,
        messages= [ImageMessage(
        original_content_url = "https://i.imgur.com/whv9VZA.jpg",
        preview_image_url = "https://i.imgur.com/whv9VZA.jpg")]))
    except:
        line_bot_api.reply_message_with_http_info(
        ReplyMessageRequest(
        reply_token=event.reply_token,
        messages=[TextMessage(text= "error")]
            )
        )
```

Setting the parameters for returning an image. Typically, the original image and the preview image are the same.

Send Image Message - Completed！

Outline

- Create LINE Bot
- 「Parrot」LINE bot
- Graphic LINE Bot
  - Create Rich Menu
- LINE Bot basic interaction
  - Return Text
  - Return Image
  - Return Sticker
  - Multiple Messages
  - Return Location
  - Quick Replies

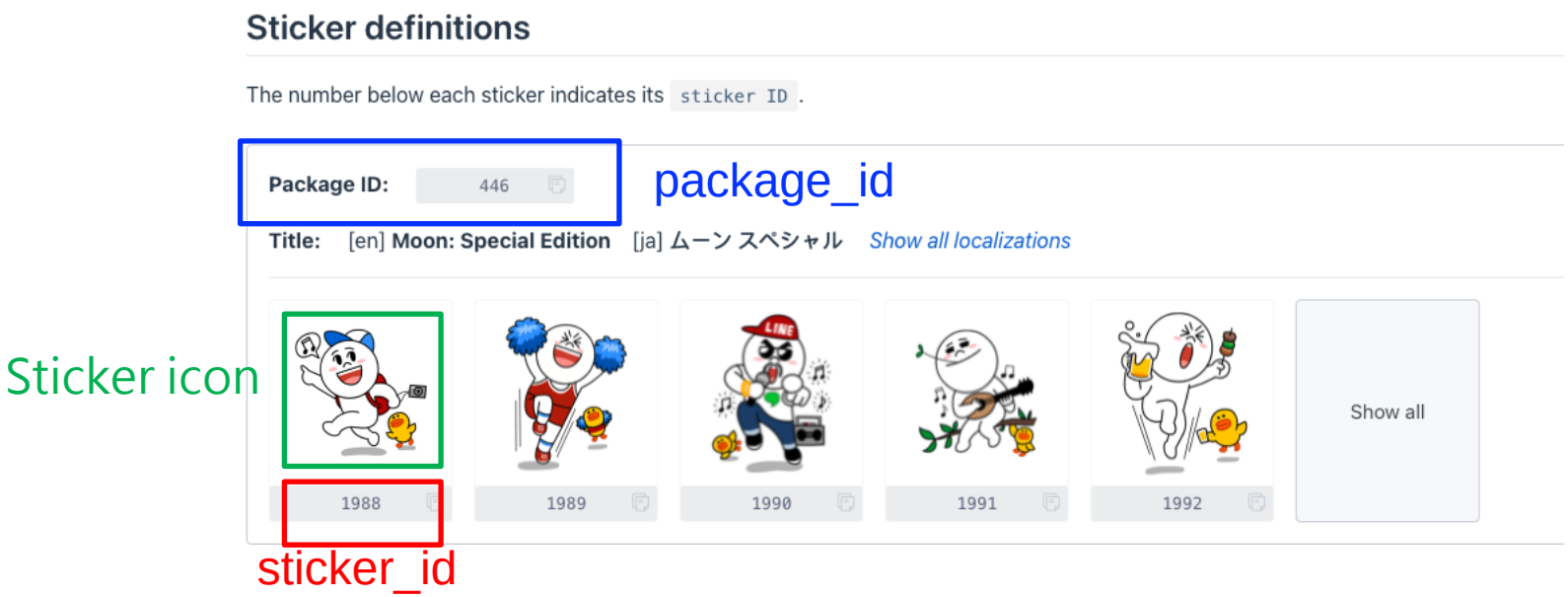
Return Sticker (1/3)

- LINE Bot can also return stickers, and the syntax is：

```
#return a sticker message
line_bot_api.reply_message_with_http_info(
    ReplyMessageRequest(reply_token=event.reply_token,messages=[
        StickerMessage(
                package_id = 'package id',
                sticker_id = 'sticker id'
                )]))
```

Return Sticker (2/3)

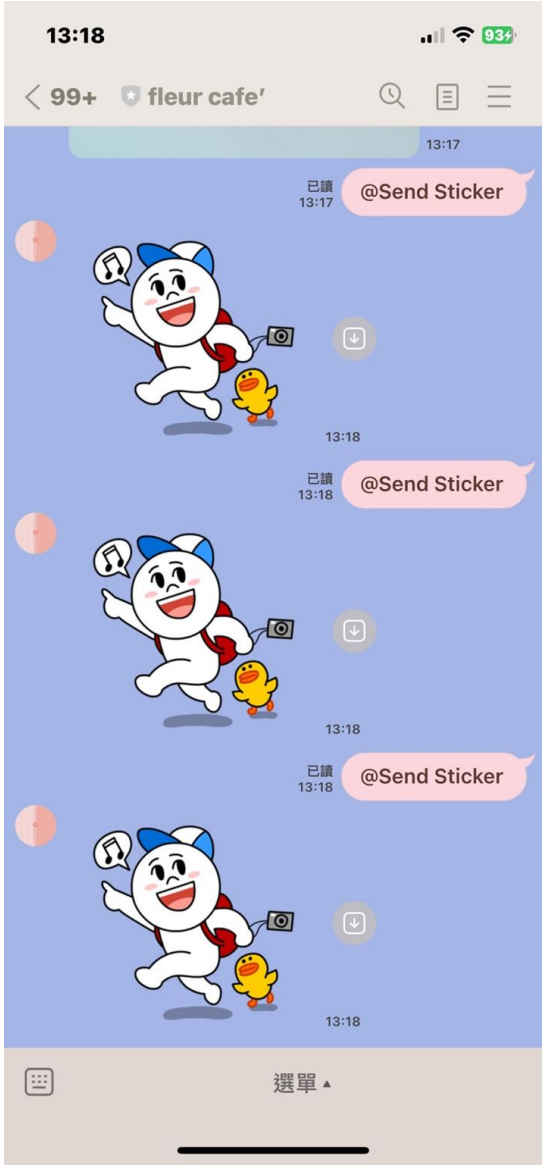
- There are a wide variety of LINE Bot stickers available. You can browse them on the following website: https://developers.line.biz/en/docs/messaging-api/sticker-list/



Return Sticker (3/3)

```
elif mtext == "@Send Sticker":
    try:
        line_bot_api.reply_message_with_http_info(
        ReplyMessageRequest(
        reply_token=event.reply_token,
        messages= [StickerMessage(
        package_id = '446',
        sticker_id = '1988'
        )]))

    except:
        line_bot_api.reply_message_with_http_info(
        ReplyMessageRequest(
        reply_token=event.reply_token,
        messages=[TextMessage(text= "error")]
            )
        )
```

Send Sticker Message- Complete！



Outline

- Create LINE Bot
- 「Parrot」LINE bot
- Graphic LINE Bot
  - Create Rich Menu
- LINE Bot basic interaction
  - Return Text
  - Return Image
  - Return Sticker
  - Multiple Messages
  - Return Location
  - Quick Replies

Multiple Messages (1/2)

- LINE Bot can not only send a single message but also send multiple messages at once.
- The method is to set the message variable as a list, with each response message as an element of the list. The syntax is:

```
#return multiple messages
Message = [
    'first return',
    'second return',
    ...
]
line_bot_api.reply_message_with_http_info(
    ReplyMessageRequest(reply_token=event.reply_token,messages=Message))
```

Multiple Messages (2/2)

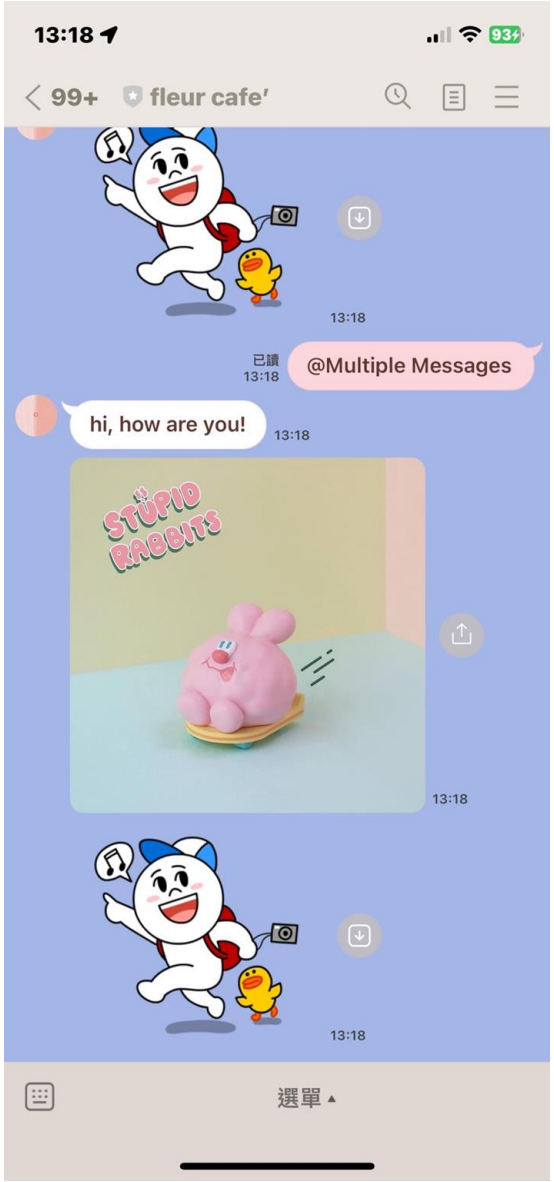
```
elif mtext == "@Multiple Messages":
    try:
        line_bot_api.reply_message_with_http_info(
        ReplyMessageRequest(
        reply_token=event.reply_token,
        messages= [TextMessage(text= "hi, how are you!"),
                   ImageMessage(
        original_content_url = "https://i.imgur.com/whv9VZA.jpg",
        preview_image_url = "https://i.imgur.com/whv9VZA.jpg"),
                   StickerMessage(
        package_id = '446',
        sticker_id = '1988'
        )
                  ]))

    except:
        line_bot_api.reply_message_with_http_info(
        ReplyMessageRequest(
        reply_token=event.reply_token,
        messages=[TextMessage(text= "error")]
            )
        )
```

Text Message

Image Message

Sticker Message

Send Multiple Messages - Complete!



Outline

- Create LINE Bot
- 「Parrot」LINE bot
- Graphic LINE Bot
  - Create Rich Menu
- LINE Bot basic interaction
  - Return Text
  - Return Image
  - Return Sticker
  - Multiple Messages
  - Return Location
  - Quick Replies

Return Location Message (1/2)

- The syntax for returning a location message that displays a specified latitude and longitude on a Google map is:

```
#return location message
line_bot_api.reply_message_with_http_info(
    ReplyMessageRequest(reply_token=event.reply_token,messages=LocationMessage(
                    title = 'title',
                    address = 'address',
                    latitude = latitude,
                    longitude = longtide
                )))
```

- The message command to send a location is "LocationMessage".

Return Location Message (2/2)

The message command to send a location is "LocationMessage".

```
elif mtext == "@Send Location":
    try:
        line_bot_api.reply_message_with_http_info(
        ReplyMessageRequest(
        reply_token=event.reply_token,
        messages= [LocationMessage(
            title = 'National Central University',
            address = '桃園市中壢區中大路300號',
            latitude = 24.968972,
            longitude = 121.1946
        )]))

    except:
        line_bot_api.reply_message_with_http_info(
        ReplyMessageRequest(
        reply_token=event.reply_token,
        messages=[TextMessage(text= "error")]
            )
        )
```

The returned location is National Central University

Send Location Message - Complete!

- Clicking on "Send Location" will open Google Maps, allowing you to move or zoom the map as desired.

Outline

- Create LINE Bot
- 「Parrot」LINE bot
- Graphic LINE Bot
  - Create Rich Menu
- LINE Bot basic interaction
  - Return Text
  - Return Image
  - Return Sticker
  - Multiple Messages
  - Return Location
  - Quick Replies

Quick Reply (1/2)

- Quick replies offer a series of options for users to choose from, which can include text, location, date, and more. The most commonly used option is text, and you can provide up to 13 options.
- Syntax of Quick Reply：

```
#return Quick Replies
line_bot_api.reply_message_with_http_info(
    ReplyMessageRequest(reply_token=event.reply_token,messages=TextMessage(
                        text= "Hint Text",
                        quick_reply = QuickReply(
                        items=[
                            QuickReplyItem(action = MessageAction(label ='value 1', text ='text 1')),
                            QuickReplyItem(action = MessageAction(label ='value 2', text ='text 2')),
                            ...
                        ]
                        )

                    ) ))
```

- label: Text displayed on the quick reply button.
- text: Text returned when the user selects that option.

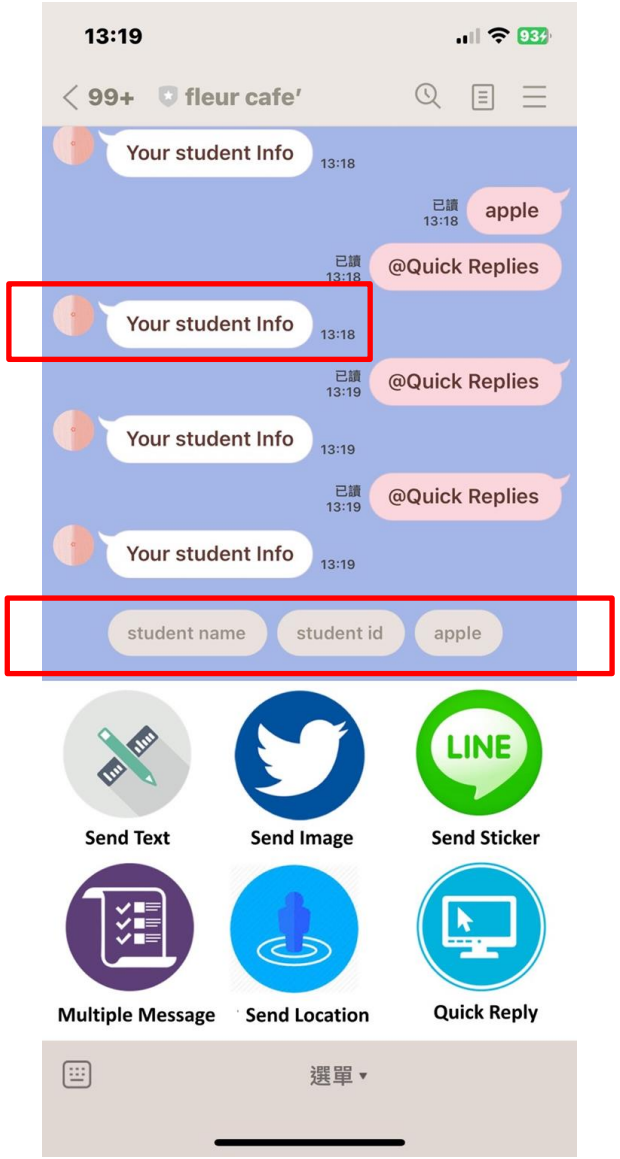
Quick Reply (2/2)

```
elif mtext == "@Quick Replies":
    try:
        line_bot_api.reply_message_with_http_info(
        ReplyMessageRequest(
        reply_token=event.reply_token,
        messages= [TextMessage(
            text= "Please select a number",
            quick_reply = QuickReply(
            items=[
                QuickReplyItem(action = MessageAction(label ='value 1', text ='text 1')),
                QuickReplyItem(action = MessageAction(label ='value 2', text ='text 2')),
            ]
            )

        )]))

    except:
        line_bot_api.reply_message_with_http_info(
        ReplyMessageRequest(
        reply_token=event.reply_token,
        messages=[TextMessage(text= "error")]
            )
        )
```

Hint Text

Set up quick menu options

Send Quick Reply - Complete！

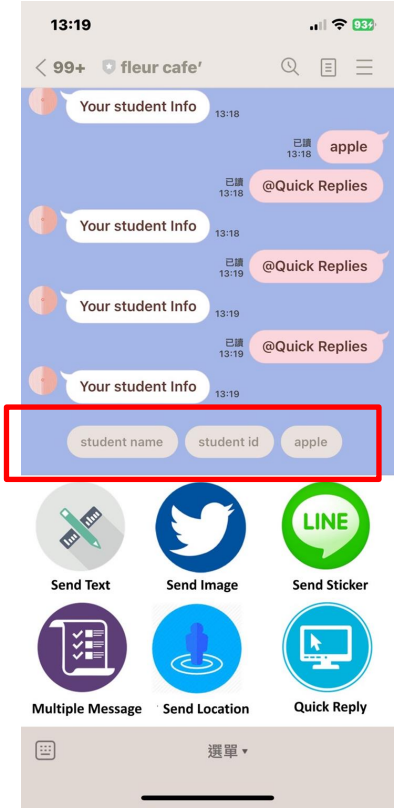


Hint Text

Quick Reply

Exercise

- How to send information again after receiving the user's reply from the quick reply menu?
- For example: Upon receiving "apple," respond with a picture?



Quick Reply