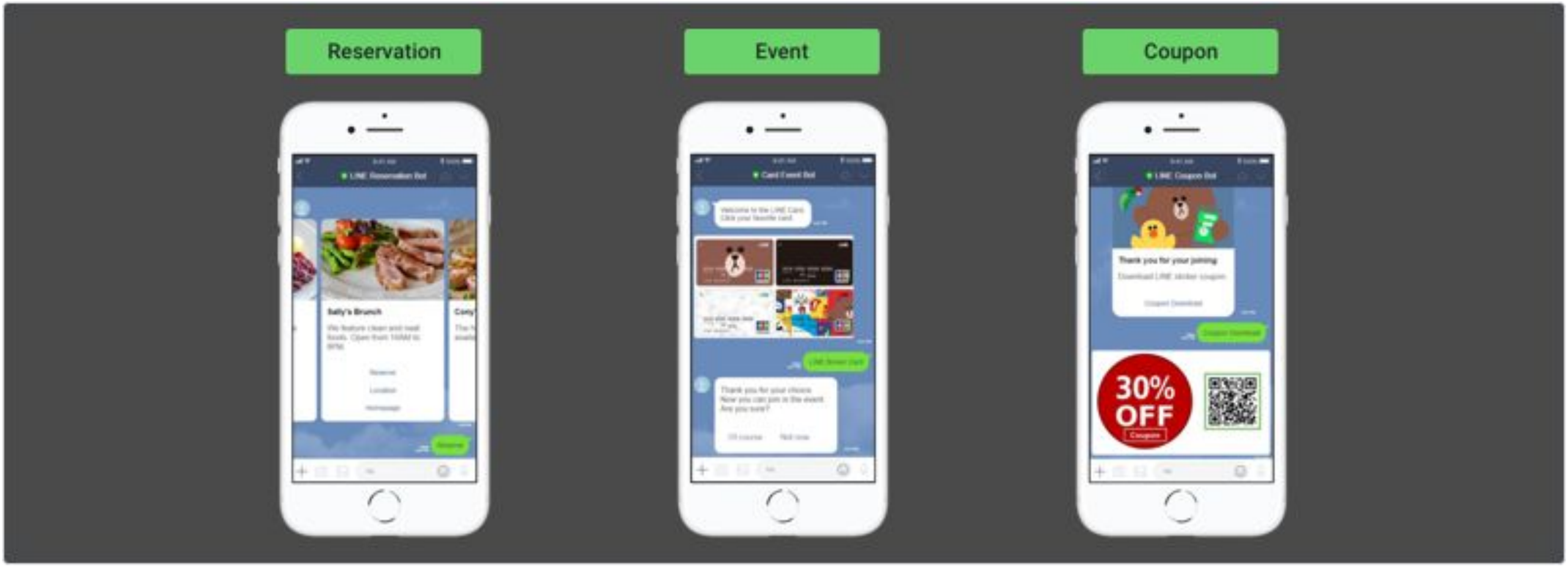
計算理論 111-1 期末專題

FSM Line Bot, TOC 111-1

TA: 陳紘霈、蔡易彤

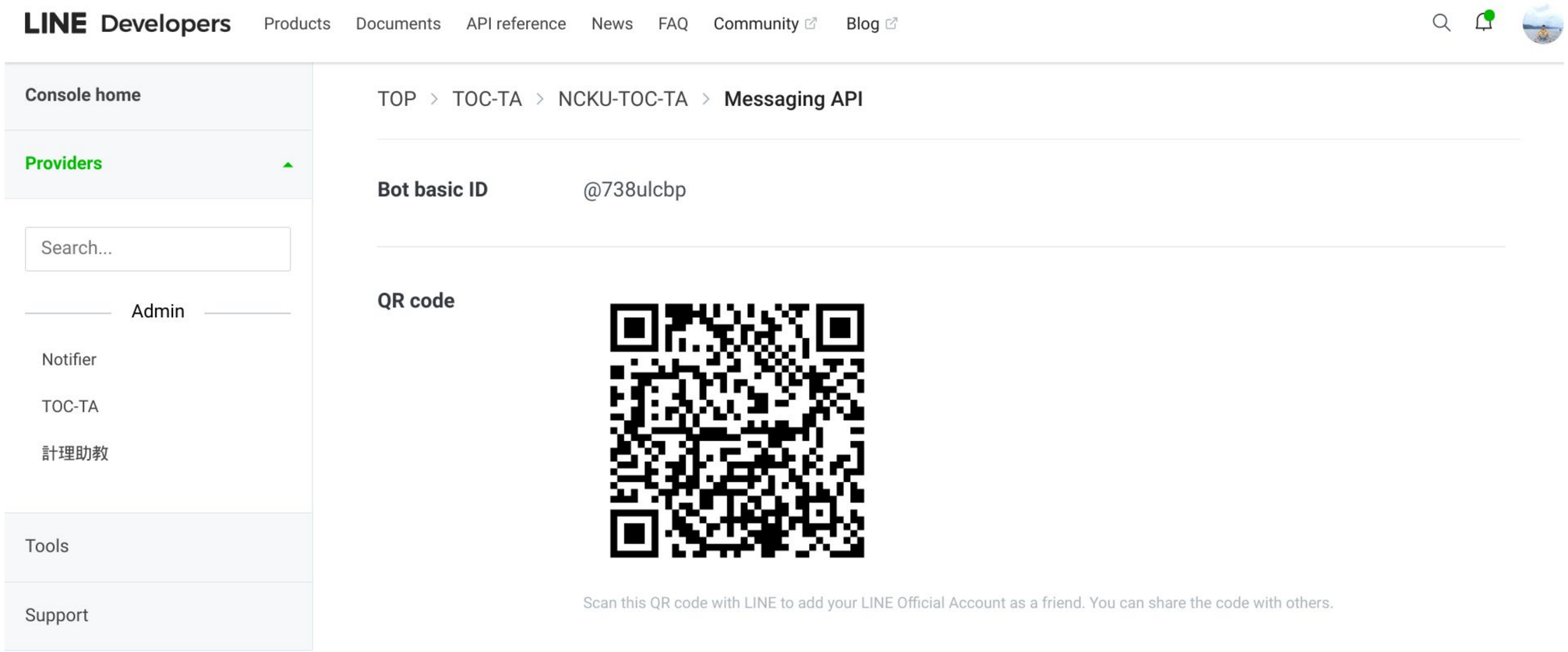
Goal

Build a Line Bot with FSM



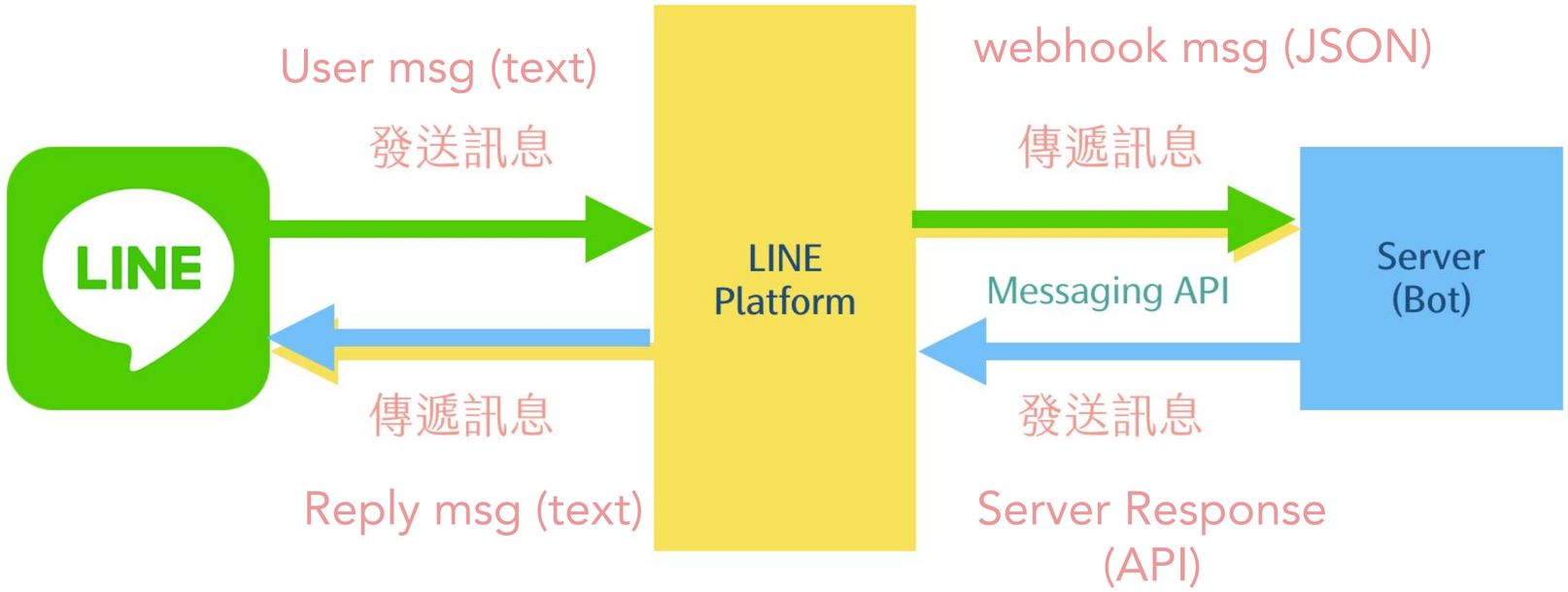
Let’s chat

Test your app (channel)



Line bot overview

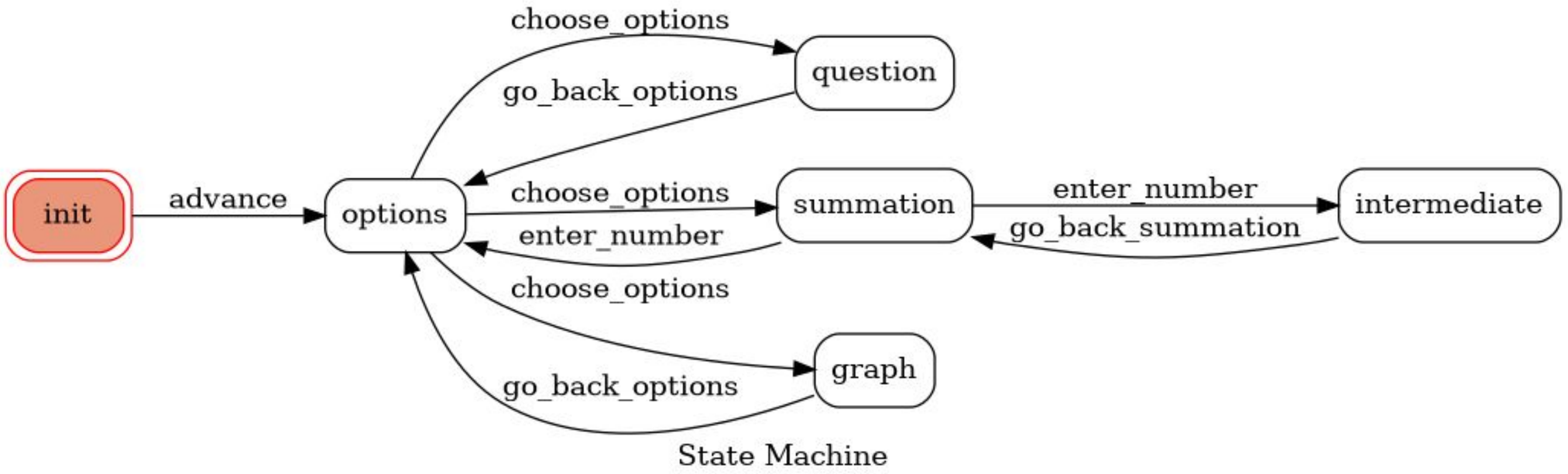
Below shows API call underneath the system. Main work of this project is to build the Server (bot).



FSM

Use finite-state-machine logics to construct the bot.

- In different state, user should receive different reply.
- Line Bot should be stateful, not stateless for this project.



Previous Works

- Line Fitness - 黃政瑋
- Daily-NBA - 周鼎軒
- Currency Related - 葉家彣

Project Outline

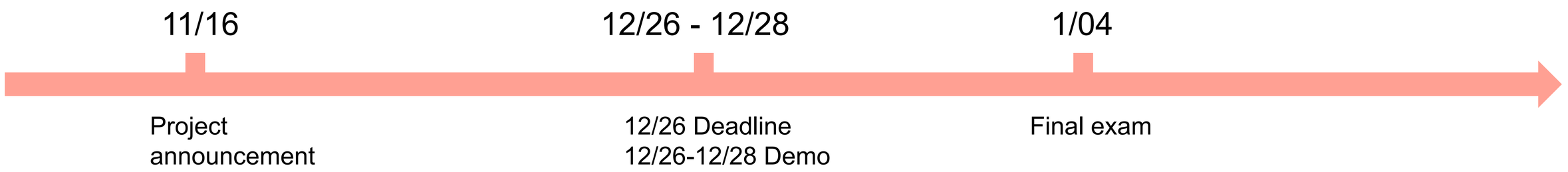
- Project timeline
- Project roadmap
- Sample code
- Grading
- Documentation and Tutorial
- Reference
- Precautions

Project Timeline

1. Project announcement (2022-11-16)
2. Fill in Github repo for grading (Google Forms)

   Deadline (2022-12-26)

   ○ Grading is based on the latest commit before deadline
4. Demo reservation (Google Forms)
5. On site demo



Project Roadmap

1. Have Python installed
2. Become a LINE developer
   - Create LINE Bot channel
   - Obtain access token and key
3. Build a simple server (Bot server)
4. Binding LINE bot
5. Test and develop
   - Read read read the docs!
6. Deploy
7. Show your work

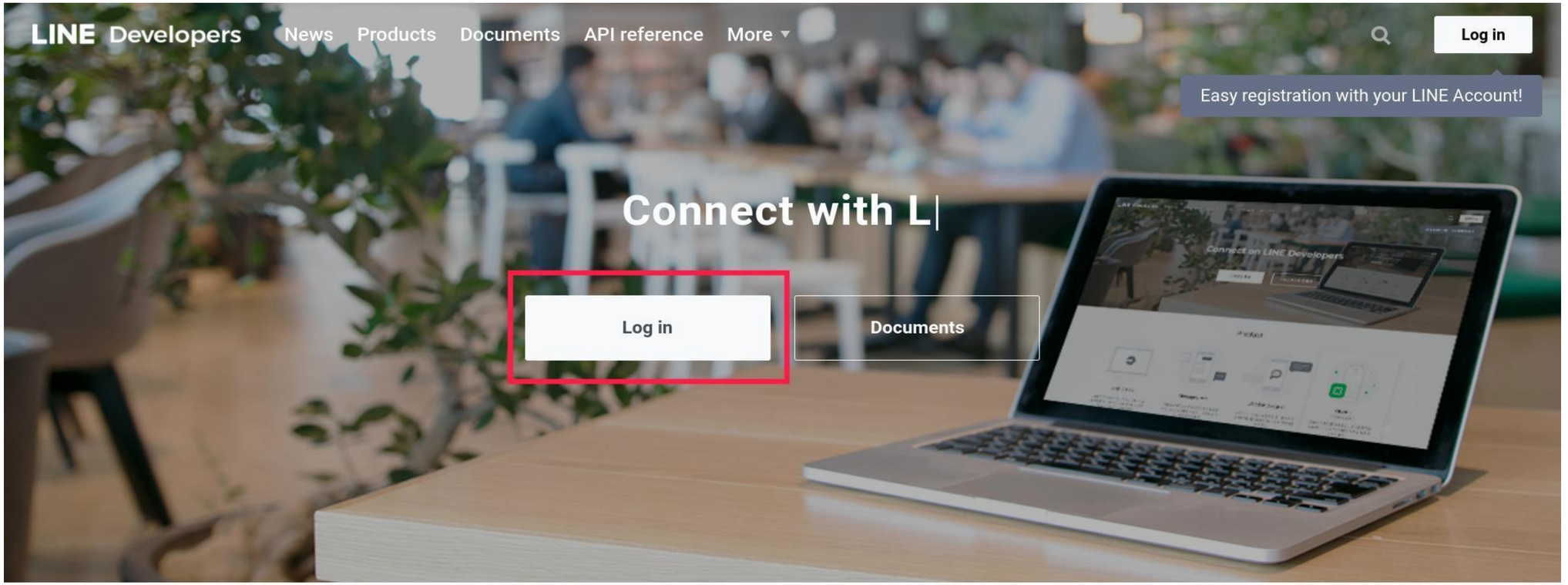
Python installation guide

Suggestion:

- Use pipenv or virtualenv to create your python environment

LINE Developer (Account)

Register and Login!

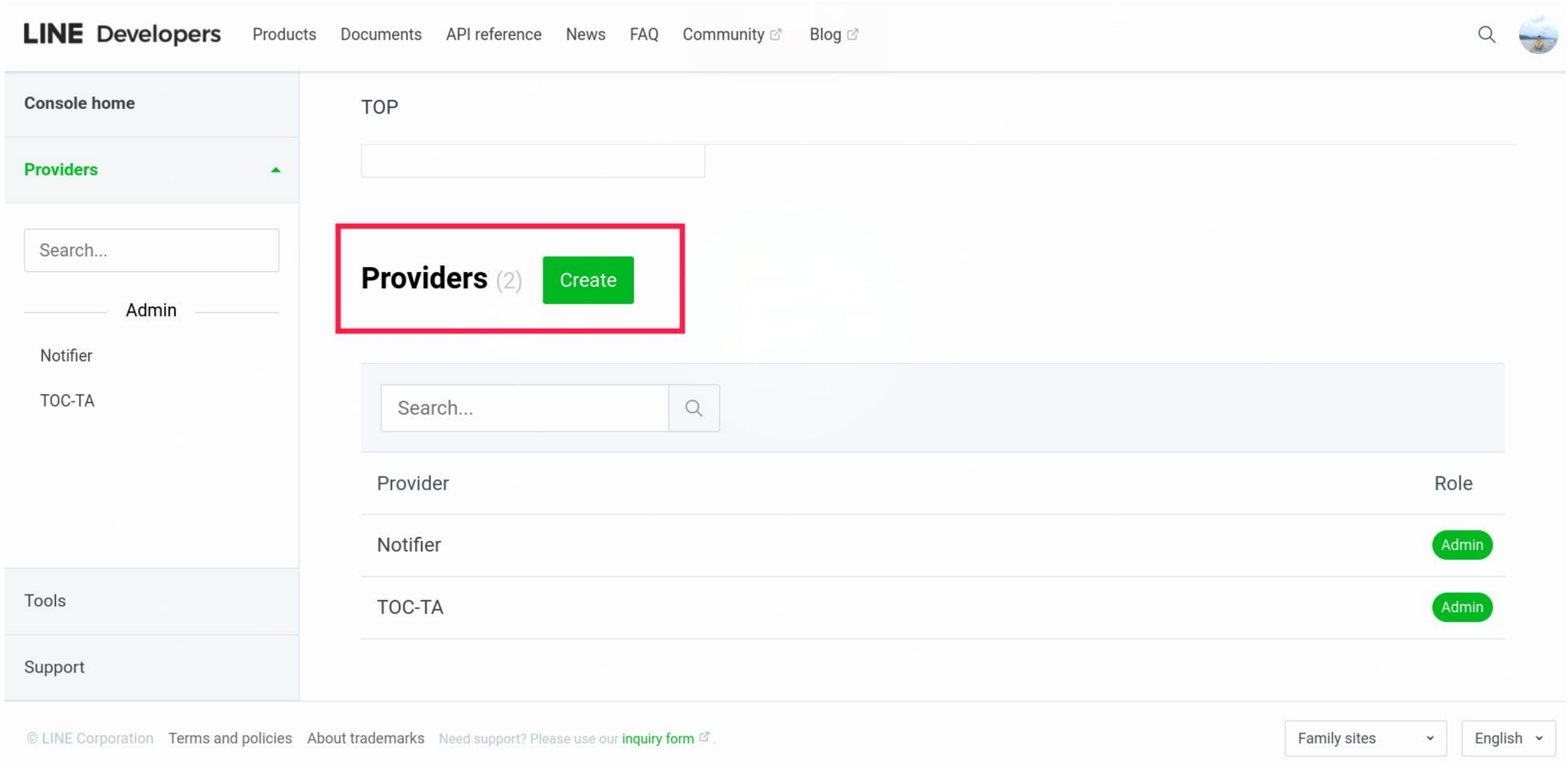


Products

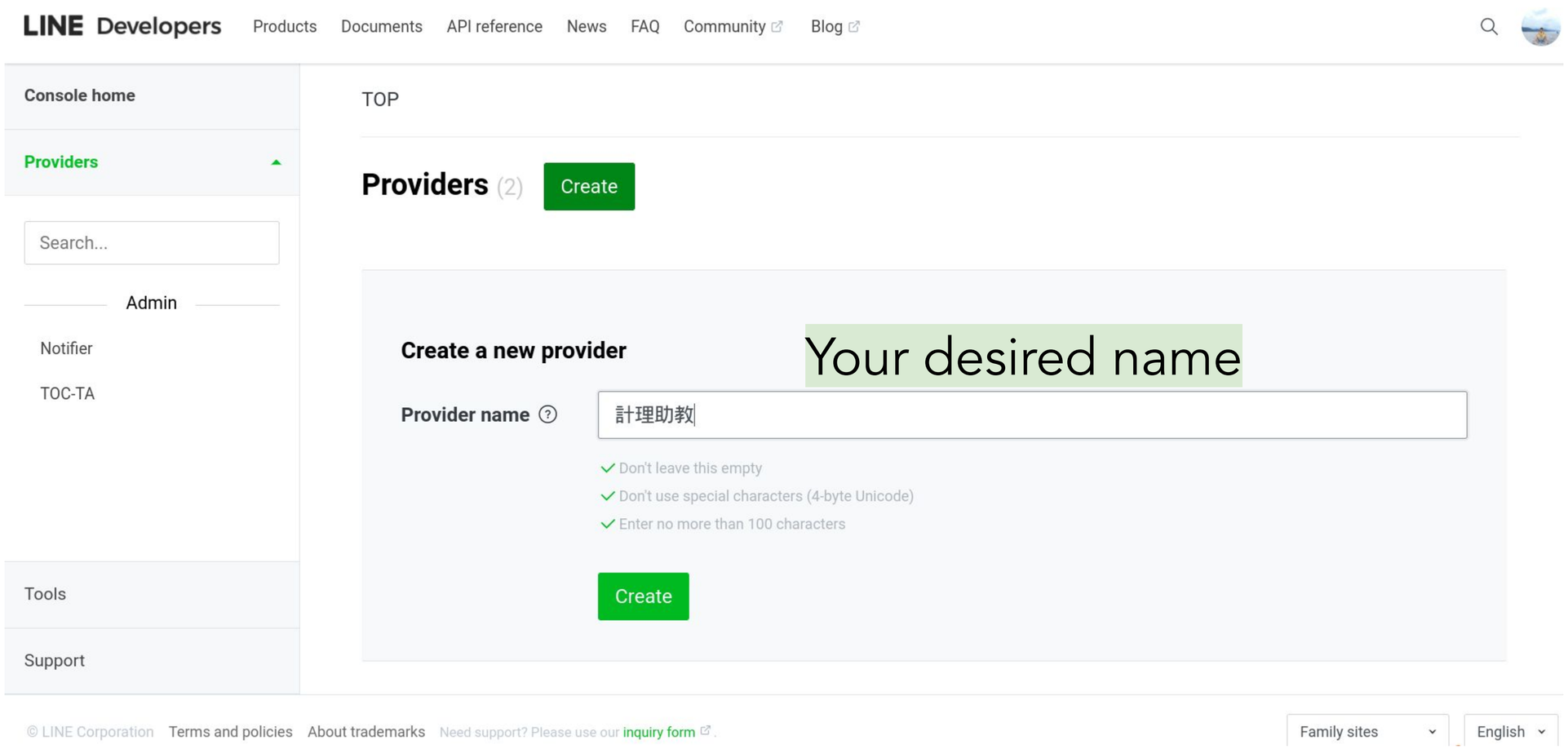
LINE Developer (Con'd)

LINE account

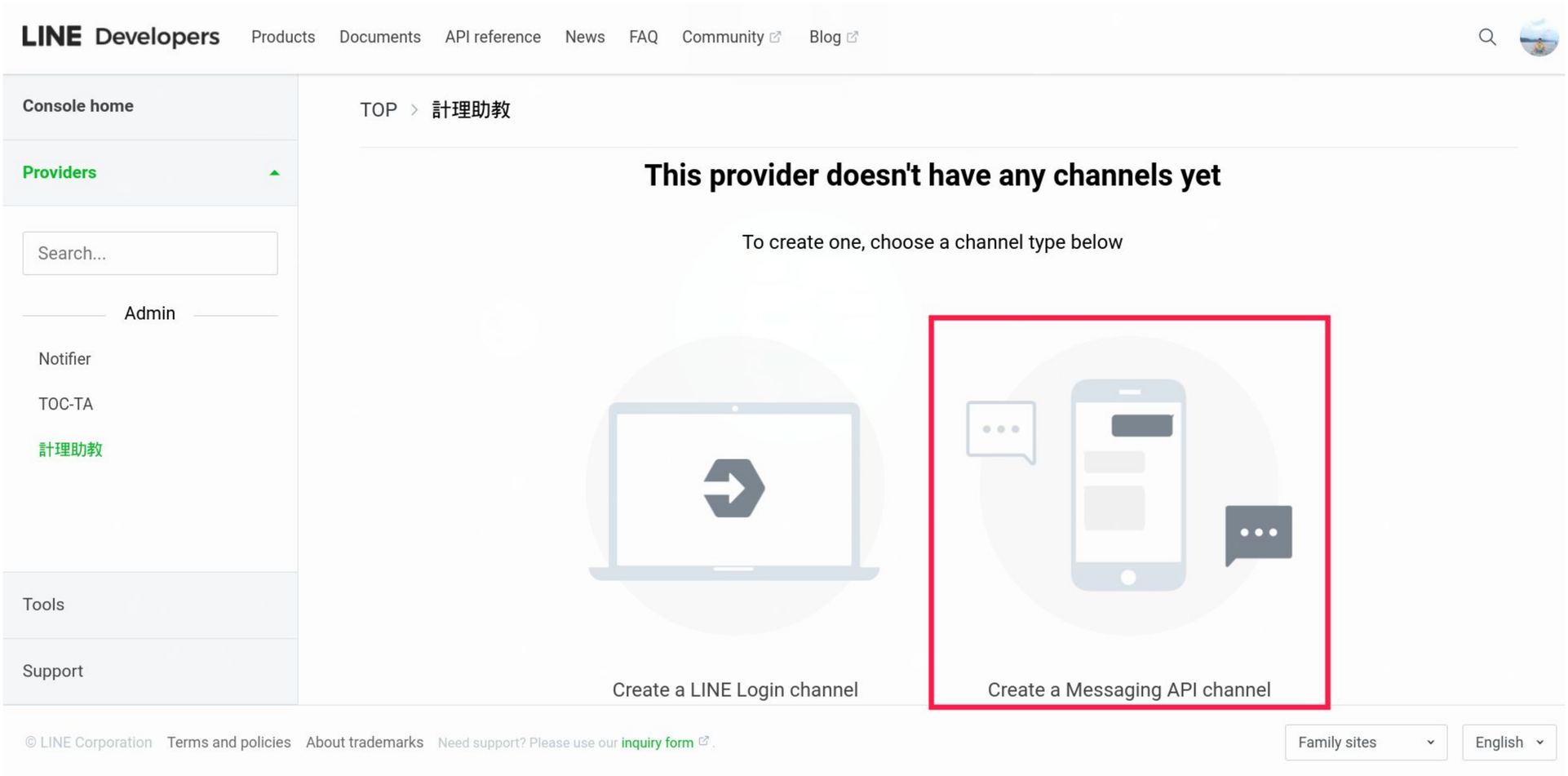
LINE Developer (Con'd)



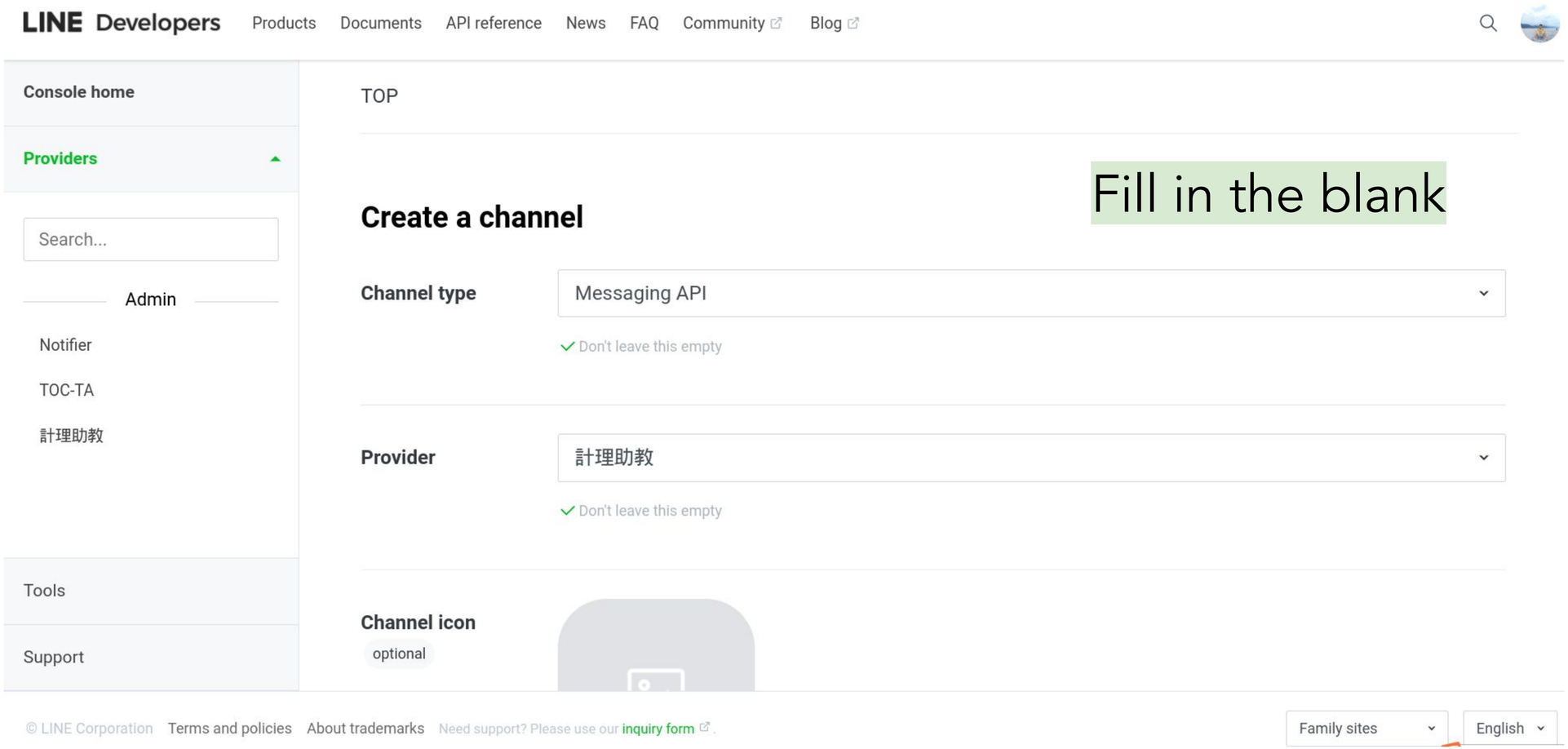
LINE Developer (Con'd)



LINE Developer (Con'd)

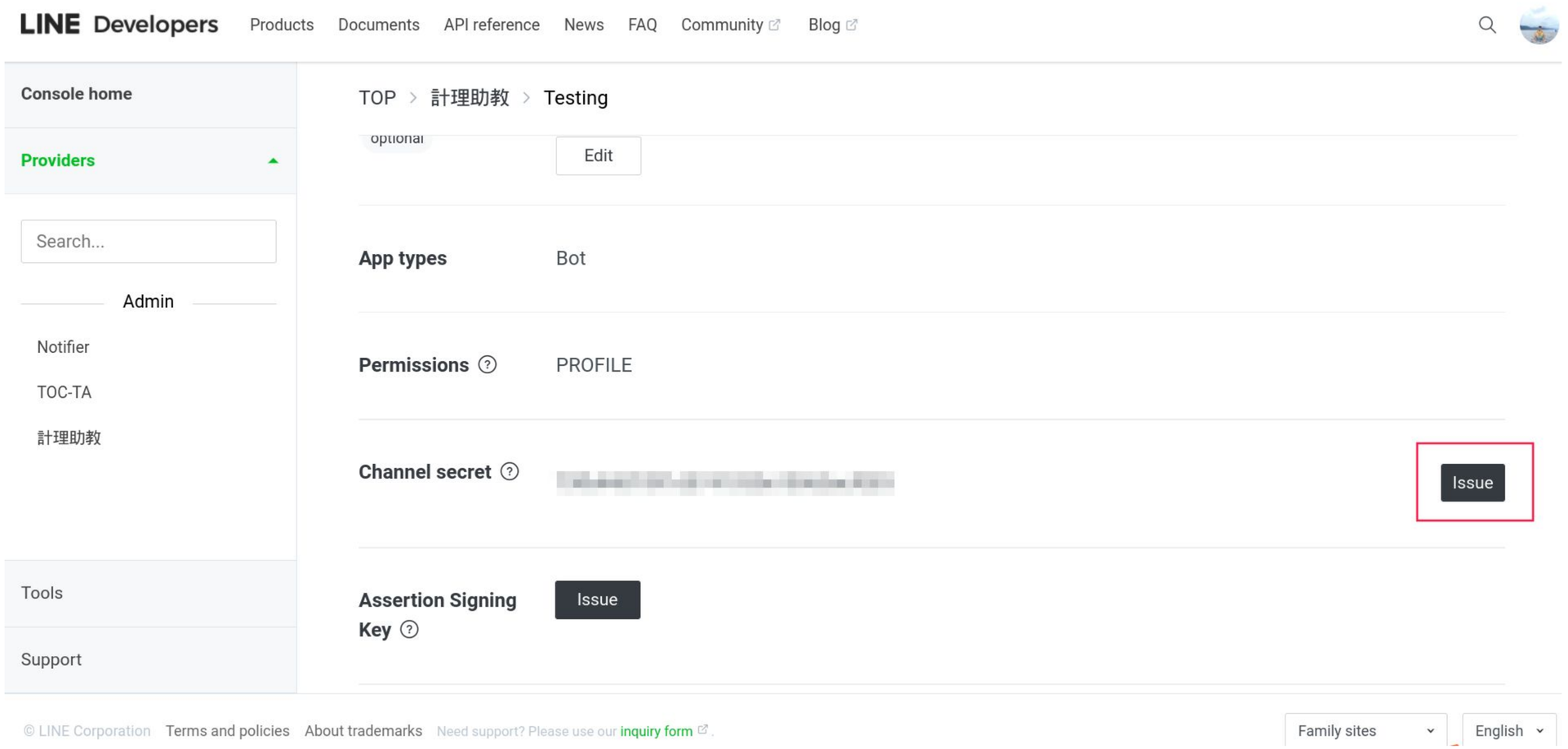


LINE Developer (Con'd)



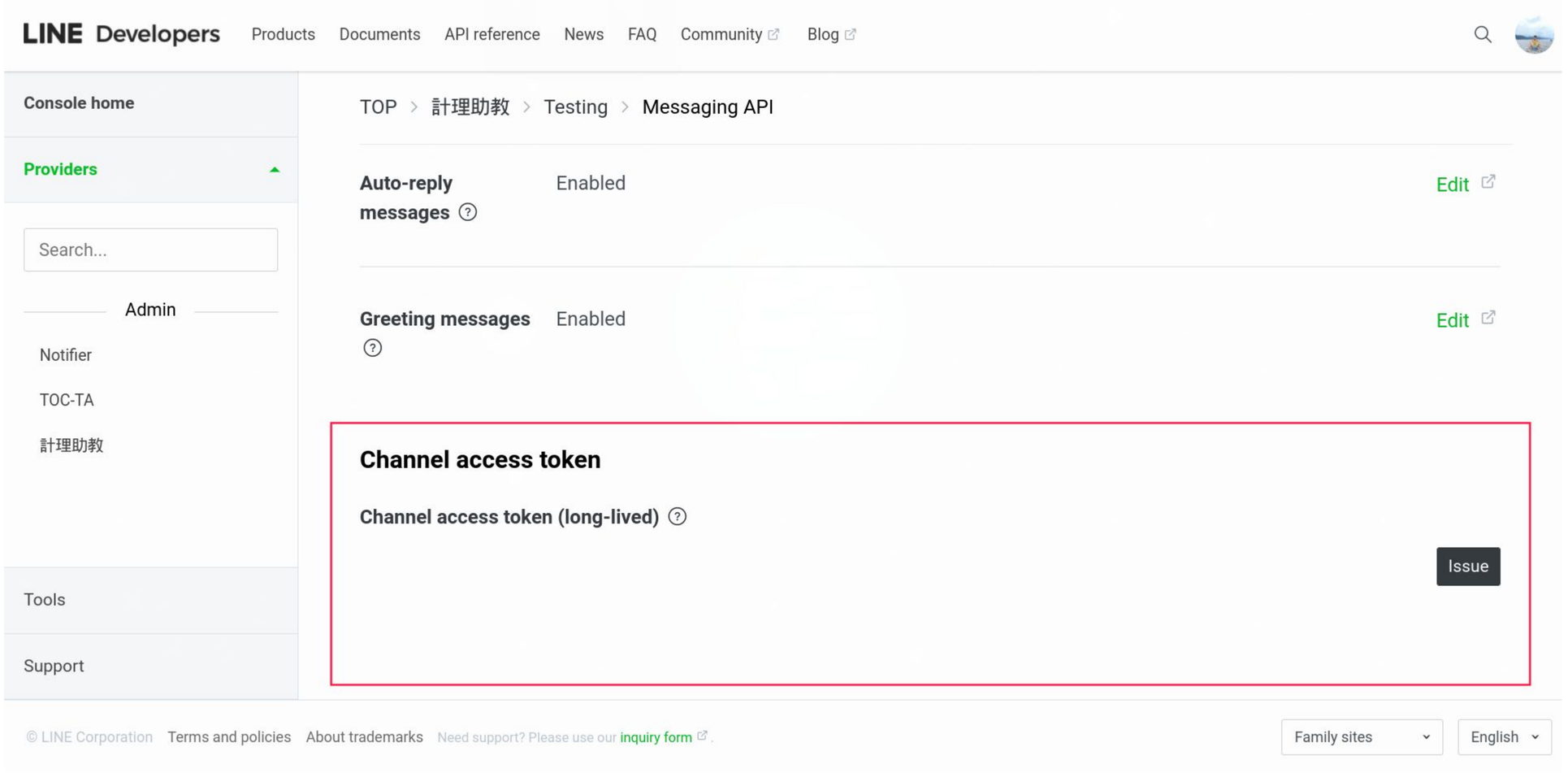
LINE Developer (Con'd)

After channel creation, on Basic Setting, issue channel secret



LINE Developer (Con'd)

On messaging API settings, at the button, gather channel access token



Building the Bot Server

Building a HTTP Server

- GET
- POST
- PUT
- DELETE

Choose a framework (Python)

- Flask
- Django
- Fastapi

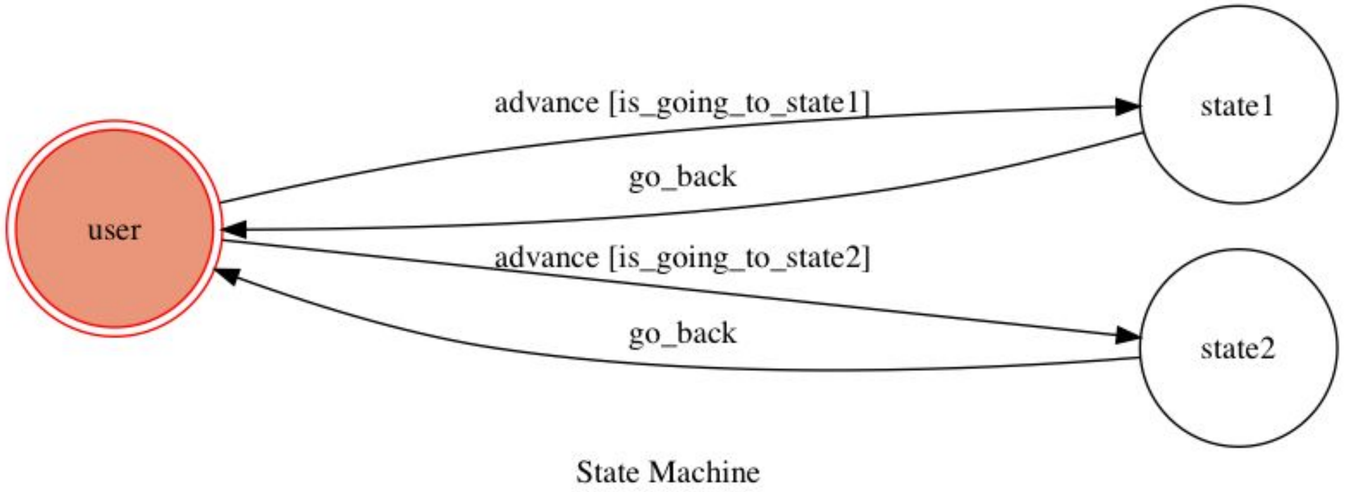
Use FSM tools (pip install transitions) to build response logic

Sample Code

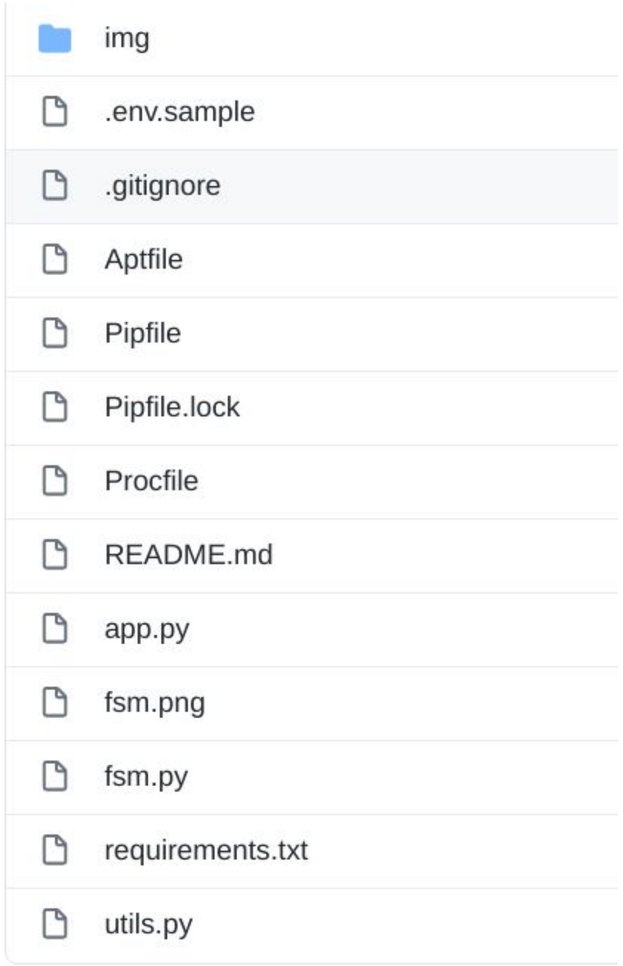
https://github.com/NCKU-CCS/TOC-Project-2020

Building the Bot Server - FSM Graph

FSM graph for sample code.



Writing a HTTP server - Overview



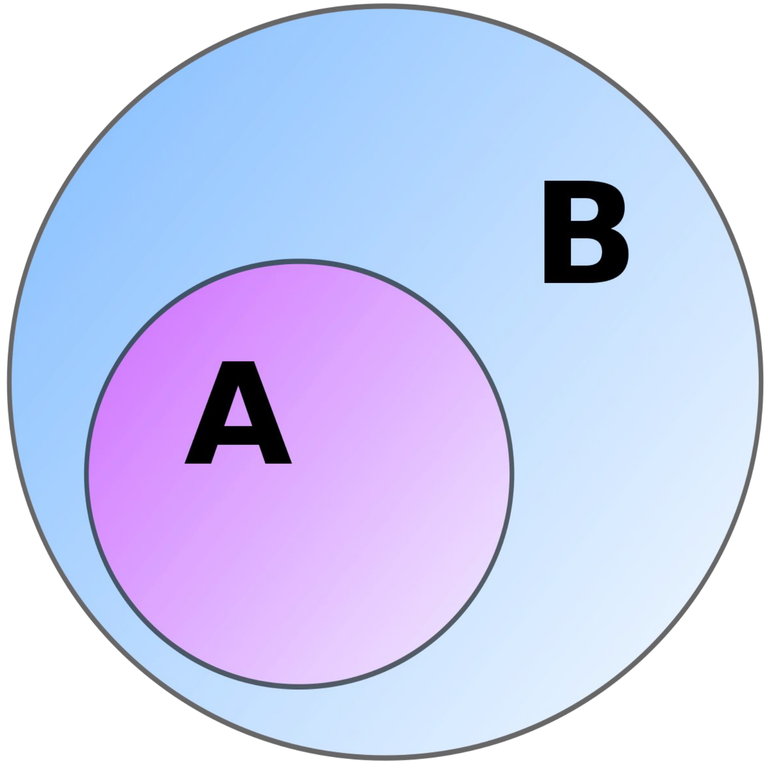
Flask server overview

- app.py - main file
- fsm.py - file handles fsm logic
- .env - file stores environment variable
- utils.py - helper function

Writing a HTTP server - Python Environment

Best practice for writing python project is to develop your code under an enclosed environment.

- virtualenv
- poetry
- pipenv - [tutorial]



B - denoted as system python environment
A - denoted as project-level python environment

Writing a HTTP server - LINE Token

To make Bot Server able to communicate with your channel, you need to set LINE_CHANNEL_SECRET and LINE_CHANNEL_ACCESS_TOKEN to your server environment, so the app have right permission to interact with your channel.

LINE_CHANNEL_SECRET, LINE_CHANNEL_ACCESS_TOKEN are obtained from previous steps (LINE channel creation)

Writing a HTTP server - Utils

utils.py - SDK for line message API

master ▾ TOC-Project-2020 / utils.py / <> Jump to ▾

ttw225 feat(TOC-Demo): well done

1 contributor

23 lines (14 sloc) | 488 Bytes

```python
import os

from linebot import LineBotApi, WebhookParser
from linebot.models import MessageEvent, TextMessage, TextSendMessage


channel_access_token = os.getenv("LINE_CHANNEL_ACCESS_TOKEN", None)


def send_text_message(reply_token, text):
    line_bot_api = LineBotApi(channel_access_token)
    line_bot_api.reply_message(reply_token, TextSendMessage(text=text))

    return "OK"


"""
def send_image_url(id, img_url):
    pass

def send_button_message(id, text, buttons):
    pass
"""
```

Writing a HTTP server - FSM

fsm.py - building the fsm for LINE text message

master ▾ TOC-Project-2020 / fsm.py / <> Jump to ▾

ttw225 feat(TOC-Demo): well done

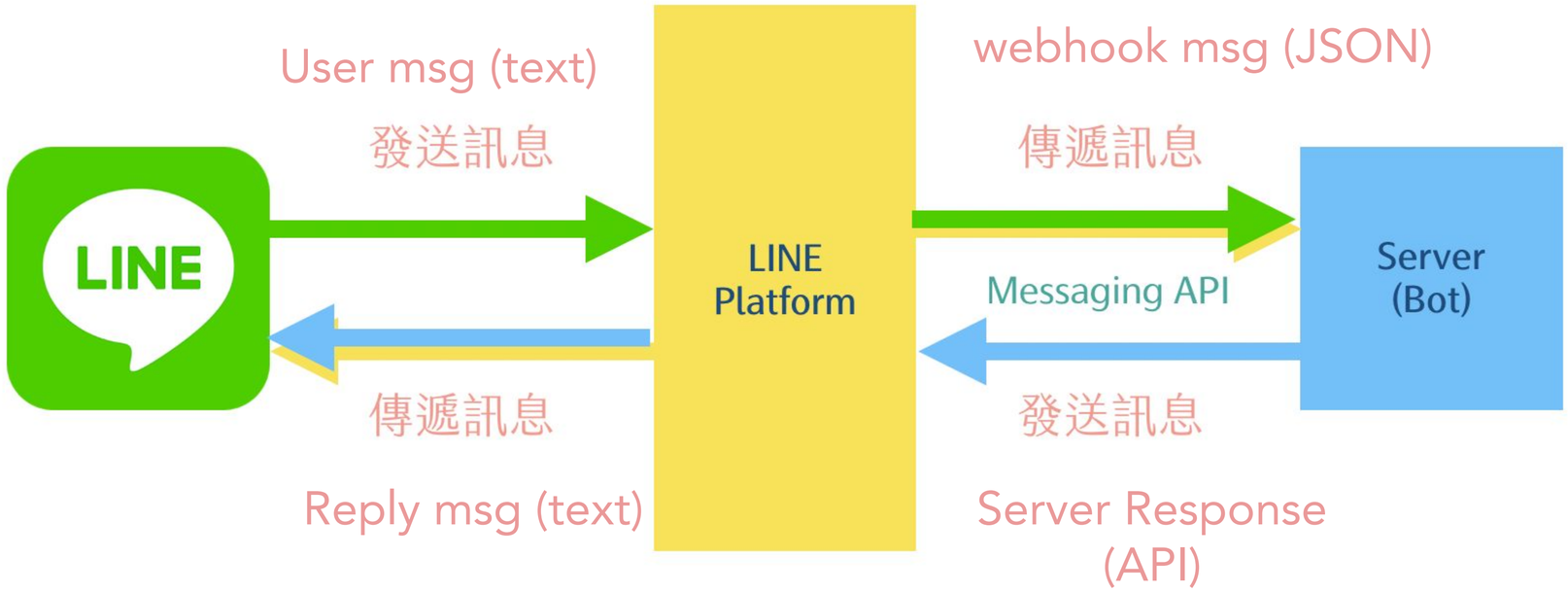
1 contributor

36 lines (25 sloc) | 993 Bytes

```python
from transitions.extensions import GraphMachine

from utils import send_text_message


class TocMachine(GraphMachine):
    def __init__(self, **machine_configs):
        self.machine = GraphMachine(model=self, **machine_configs)

    def is_going_to_state1(self, event):
        text = event.message.text
        return text.lower() == "go to state1"

    def is_going_to_state2(self, event):
        text = event.message.text
        return text.lower() == "go to state2"

    def on_enter_state1(self, event):
        print("I'm entering state1")

        reply_token = event.reply_token
        send_text_message(reply_token, "Trigger state1")
        self.go_back()
```

Writing a HTTP server - Recall

Below shows API call underneath the system. Main work of this project is to build the Server (bot).



Writing a HTTP server - Clone & Trace

1. Clone the project
2. Build environment
3. Trace the code
4. Refer the official documentation
5. Extend your bot

Test server locally

ngrok - secure introspectable tunnels to localhost

- Map your localhost:8000 to https domain

Extra reference:
ngrok reference

Deployment

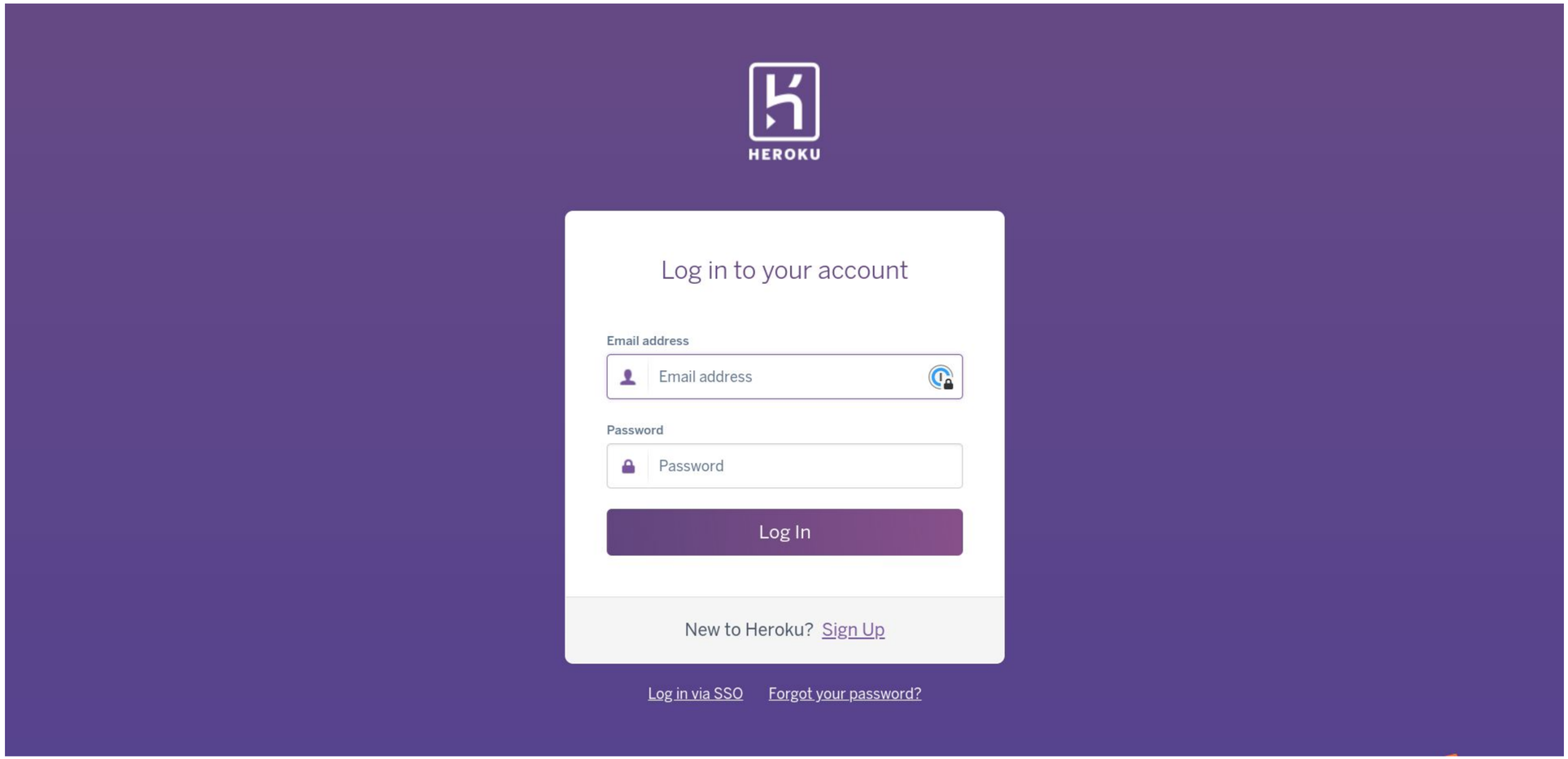
Web deployment is the process of deploying the code (html, css, javascript and server code) from source control or source artifacts to a hosting platform. This is usually in the cloud or on a local server.

Method and Choice

- A local machine with public static IP
- Ngrok (Basic)

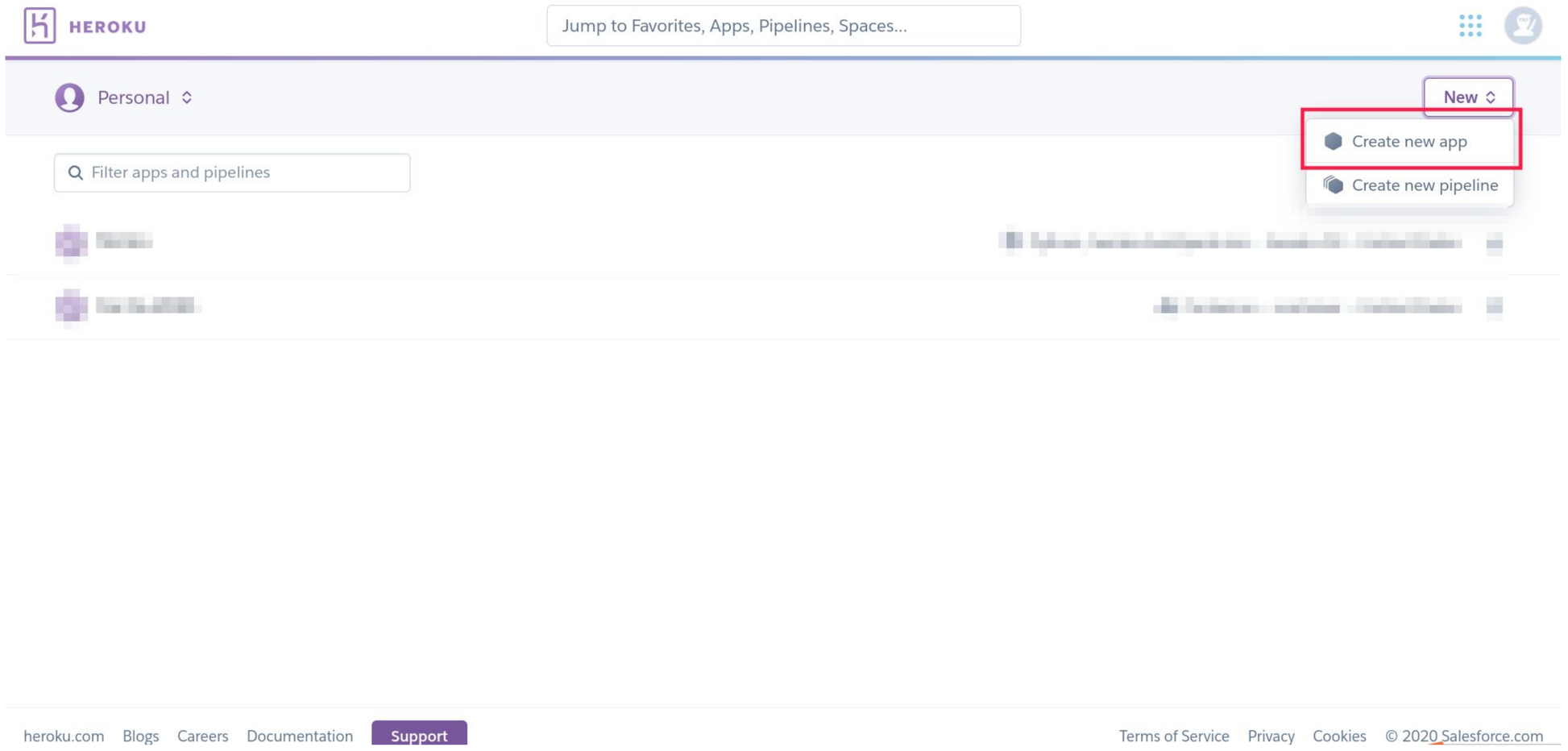
Deploy with Heroku

Register Heroku and create an app



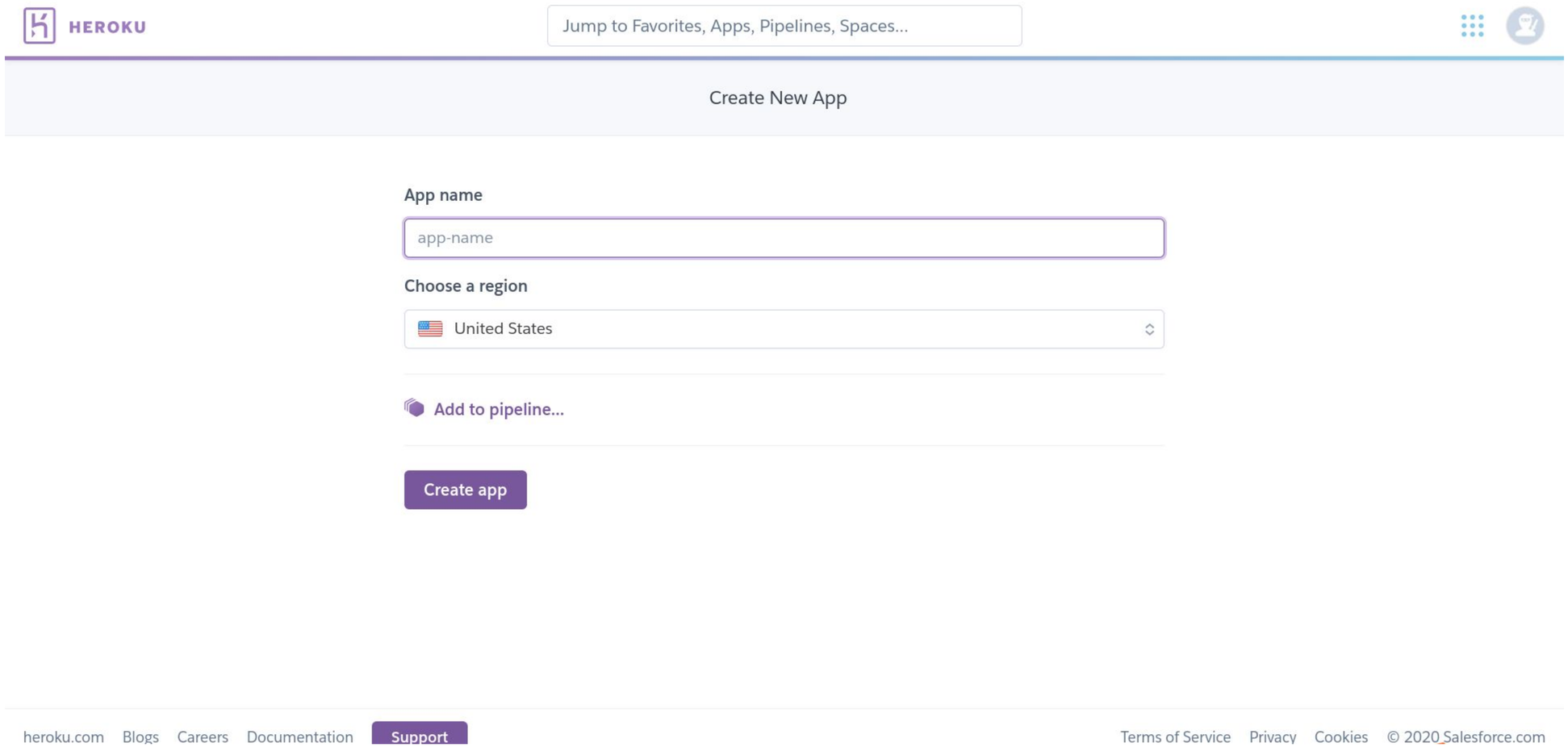
Deploy with Heroku

Register Heroku and create an app



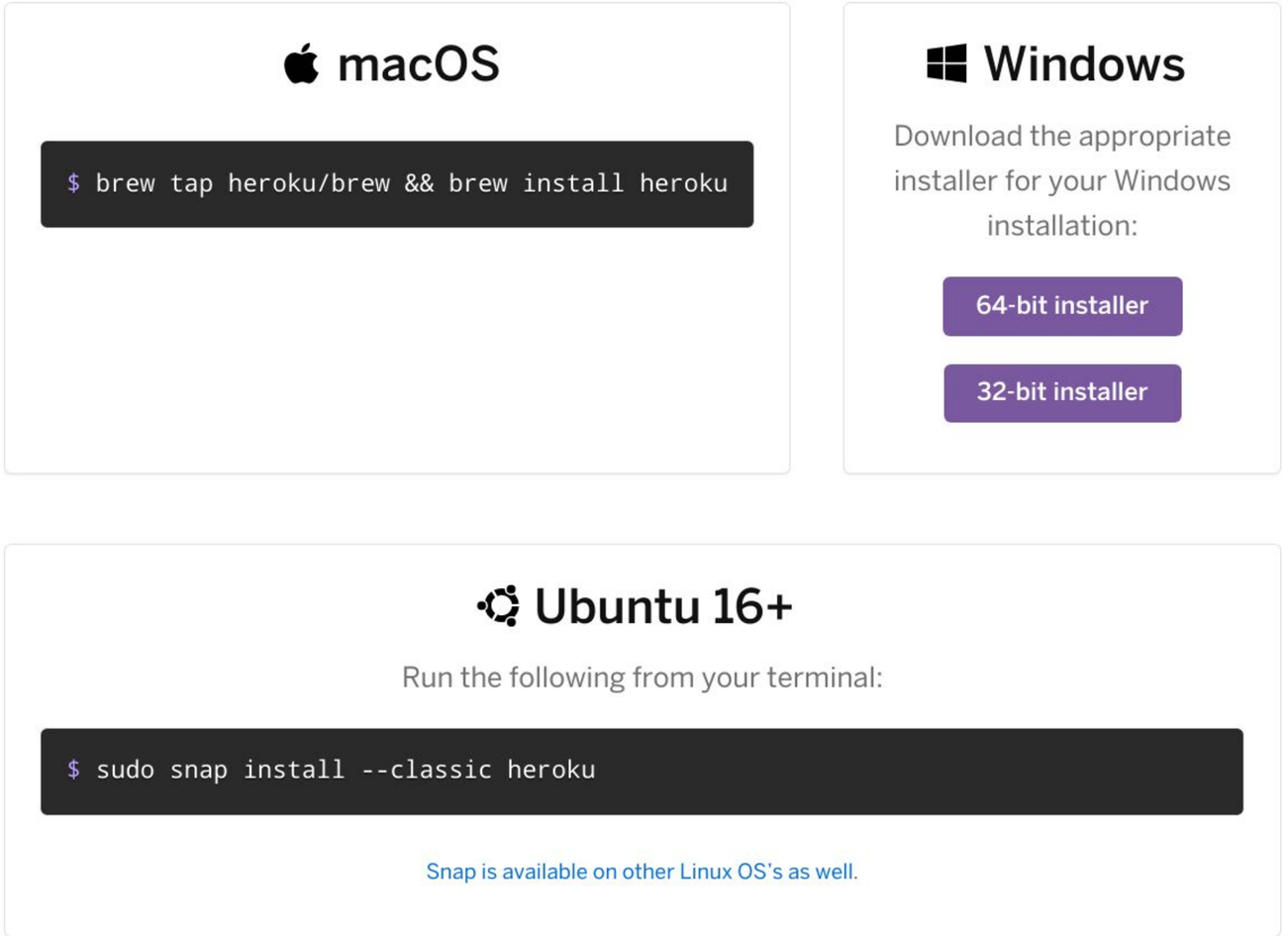
Deploy with Heroku

Fill in app name



Deploy with Heroku

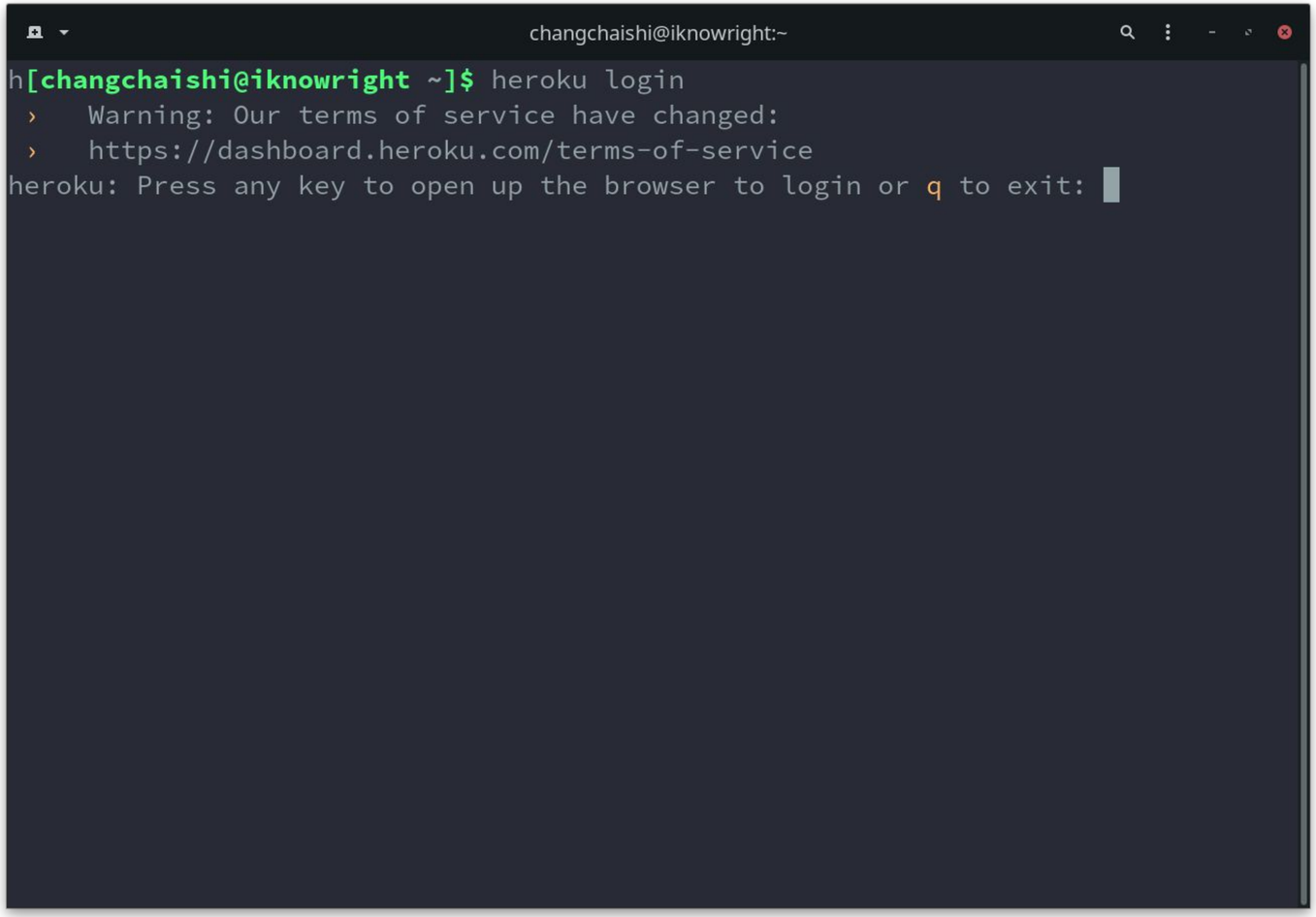
[Install Heroku-Cli]



Deploy with Heroku

Login Heroku-Cli (Terminal depends on your OS)

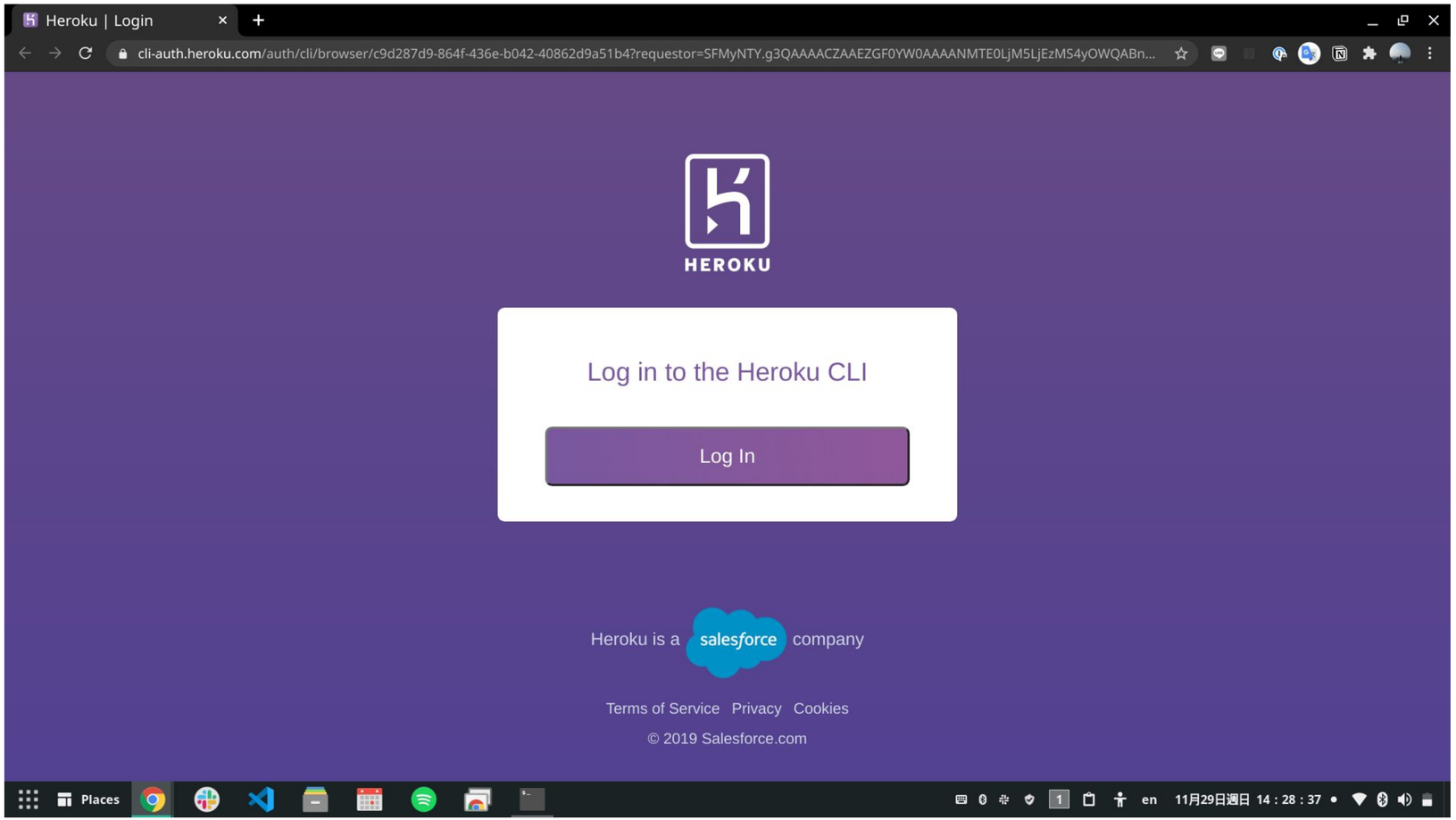
- Terminal will redirect you to browser-login



Deploy with Heroku

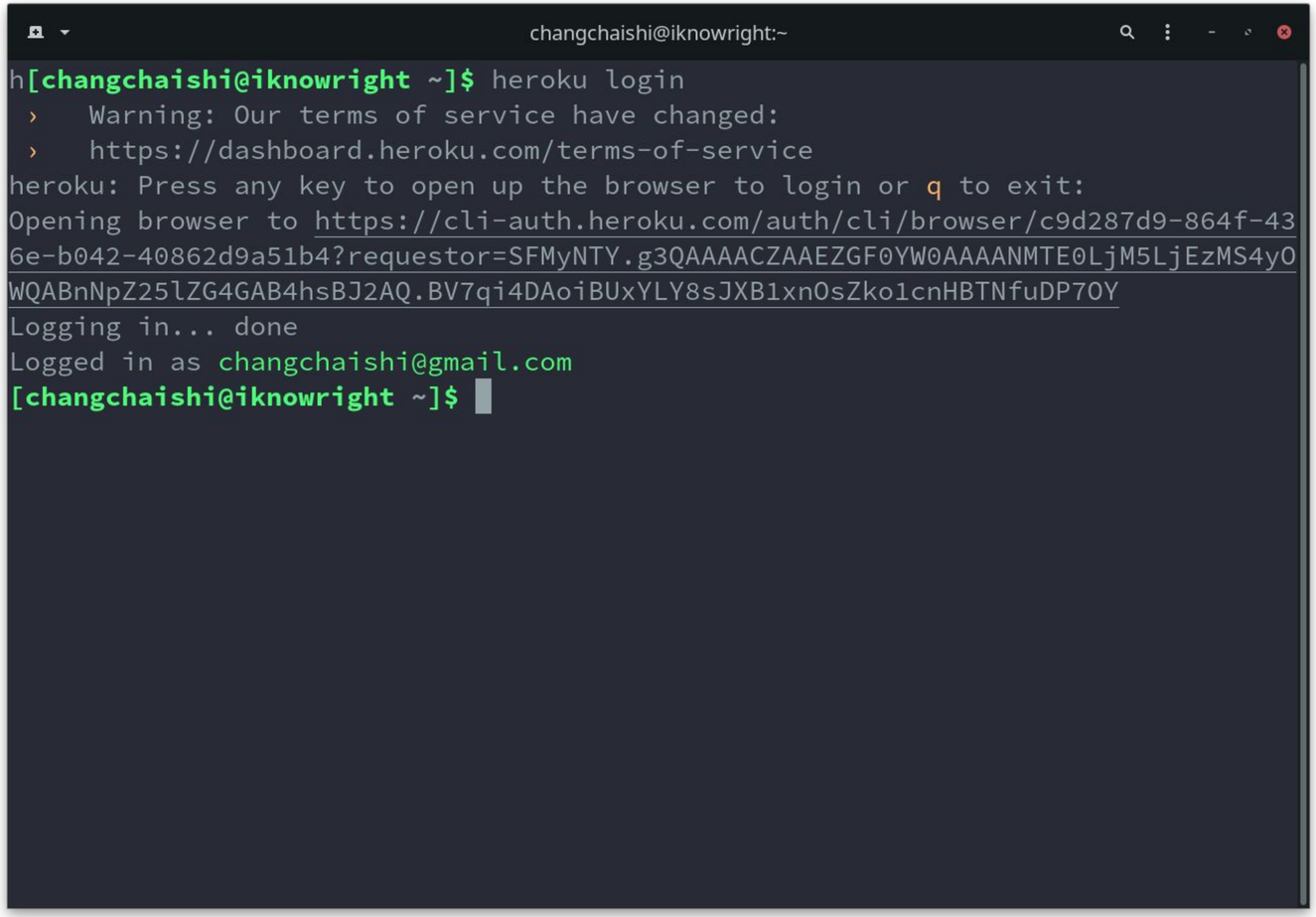
Login Heroku-Cli (Terminal depends on your OS)

- Terminal will redirect you to browser-login



Deploy with Heroku

- Back to the CLI, you can see the logged in info.



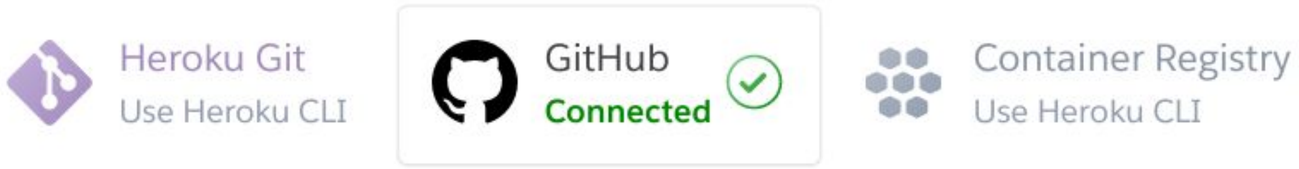
Deploy with Git

Using Git:

- What is git? Version control system
- Difference between git and github
- Local vs Remote
- Github vs Gitlab vs Heroku

Heroku: you can choose between

- Github
- Heroku
- Docker registry



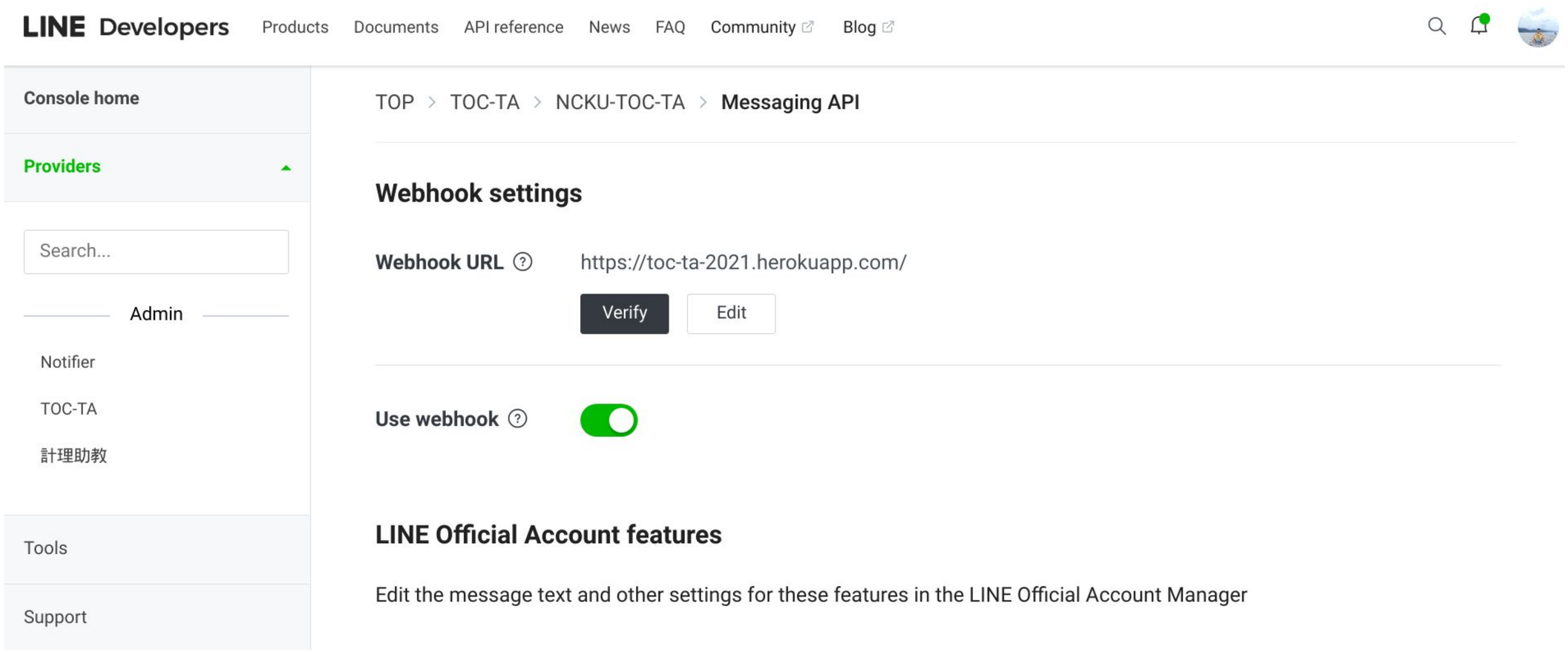
Deploy with Git


```
[/your_project_dir] $ heroku git:remote -a {HEROKU_APP_NAME}
[/your_project_dir] $ git add .
[/your_project_dir] $ git commit -m "commit msg"
[/your_project_dir] $ git push -f heroku master
[/your_project_dir] $ Heroku logs --tail -a {HEROKU_APP_NAME}
```

After pushing your code to heroku, heroku will automatically build your app, use heroku cli to check build logs

Binding Deployed APP to Line Console

Tell Line which webhook url to POST. (Your server url).



Grading

- Basic (60%)
- Present (10%)
- Functionality (10%)
- Creativity (10%)
- Bonus (5~10%)

Basic (60%)

- Able to show or generate FSM Diagram
- Bot running properly
- At least 4 states for FSM
- More than 3 transactions departing from initial state

Present (10%)

- Demo Smoothness
- Documentation (README and etc.)
- Illustration

Functionality (10%)

- State complexity (Not amount of states)
  - Transition continuity
  - Transition correctness
  - Notice that the design of state structure not to be redundant
- CRUD (create read update delete)

Creactivity (10%)

LineBot genre, theme :

- Sports
- Game
- Service
- News
- Techs
- Tools

Bonus (5 ~ 10%)

- Deploy
  - AWS - Amazon Web Service
  - Heroku - PaaS (Platform as a service)
  - Microsoft Azure
  - GCP - Google cloud platform
- Extra functionality or technics
  - Line API
  - image/sound/video
  - Web crawling
  - Machine learning
  - Blockchain
  - Others

Topic Reference

台灣事實查核中心

疾管家

Cofacts 真的假的

Development Reference

Line Developer 註冊

LINE 開發環境

Heroku

LineBot + python + Heroku

ngrok

Project Alternatives

Bot Platform: Line (exclusively)

Programming Language / Framework: Unrestricted

- Python
- node.js
- golang
- anything else that supports FSM graph

Precautions

LINE

- Line Reply Token - can only reply once
- Line Bot (Free ver.) - friend limit: 50 people

Heroku

- Heroku - dyno will be in sleep mode when no activities found in the server

Questions?

主旨 : [Final Project] 簡述遇到的問題

內文 :

```
- 姓名學號 :
- 系統環境 :
- 系統版本 :
- 套件版本 :
- 在哪一個步驟遇到的問題 :
- 詳述問題 :
- 完整的錯誤訊息 :
- 已經試過的解決方法 :
 - 在這個問題上已經花費的時間 :
```

2018 FAQ

Thanks