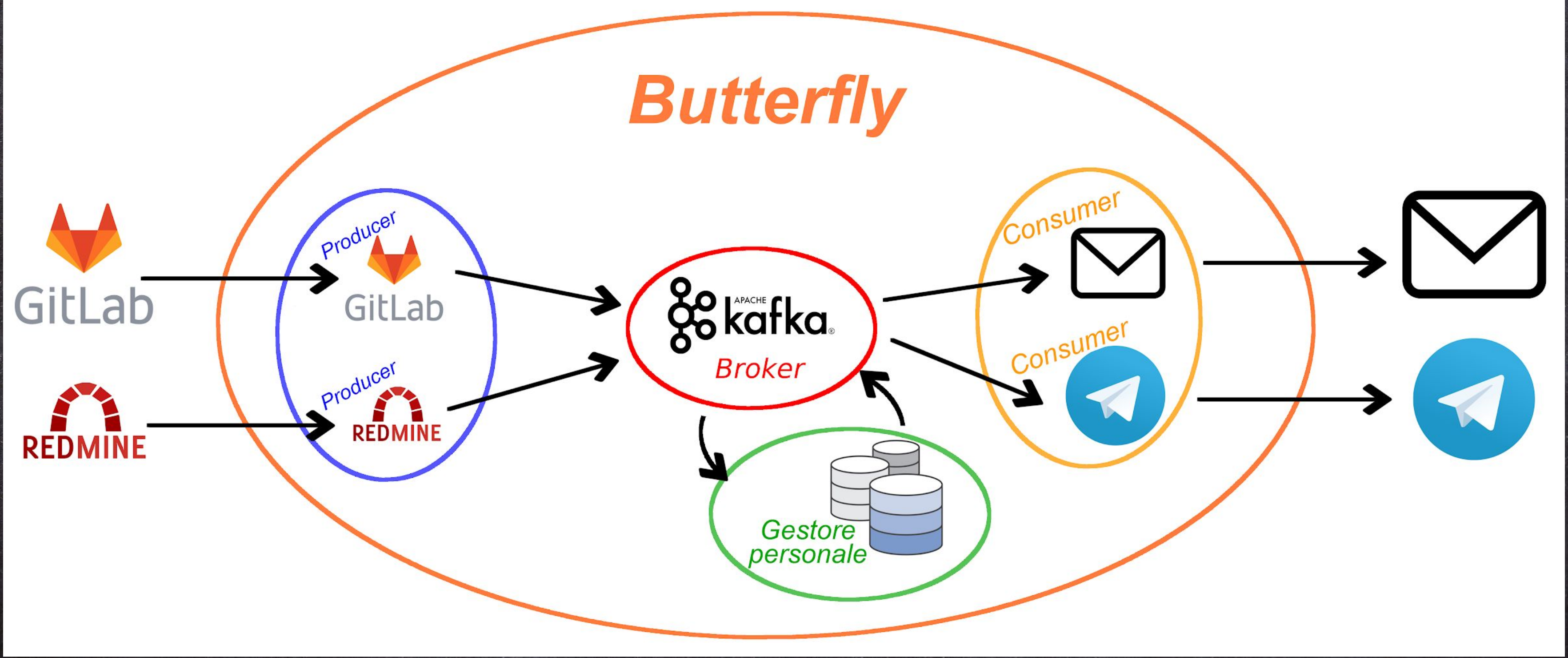
Butterfly

Proof of Concept

Università degli Studi di Padova
26 / 02 / 2019

BUTTERFLY


Butterfly
GitLab
REDMINE
Producer
GitLab
Producer
REDMINE
APACHE kafka
Broker
Gestore personale
Consumer
Consumer

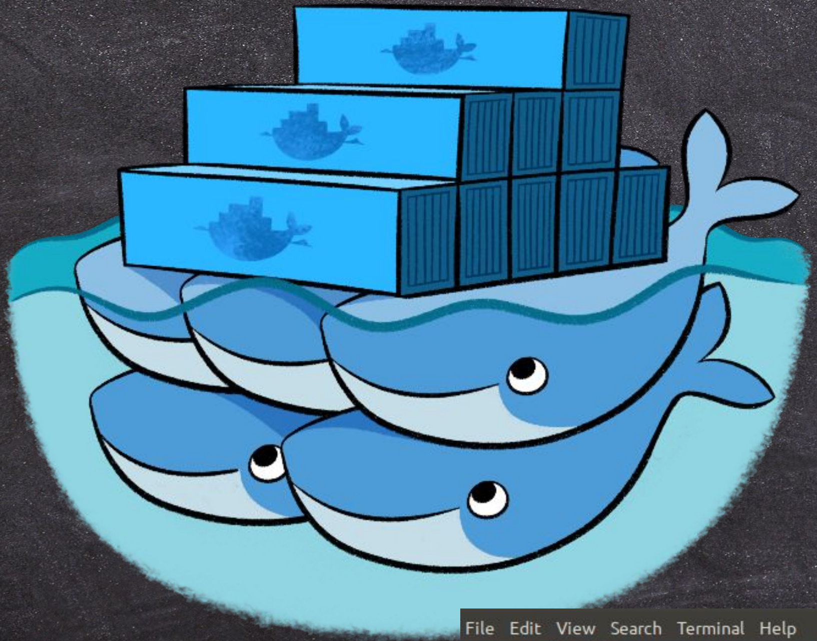
LINGUAGGI UTILIZZATI

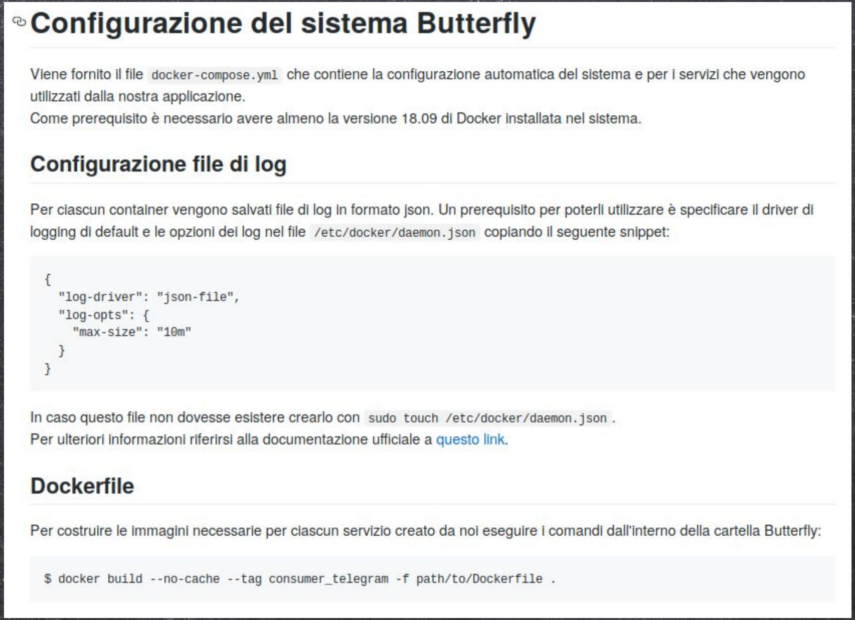
python SOFTWARE FOUNDATION

HTML + CSS

TECNOLOGIE UTILIZZATE

DOCKER





Configurazione del sistema Butterfly

Viene fornito il file `docker-compose.yml` che contiene la configurazione automatica del sistema e per i servizi che vengono utilizzati dalla nostra applicazione.
Come prerequisito è necessario avere almeno la versione 18.09 di Docker installata nel sistema.

Configurazione file di log

Per ciascun container vengono salvati file di log in formato json. Un prerequisito per poterli utilizzare è specificare il driver di logging di default e le opzioni dei log nel file `/etc/docker/daemon.json` copiando il seguente snippet:

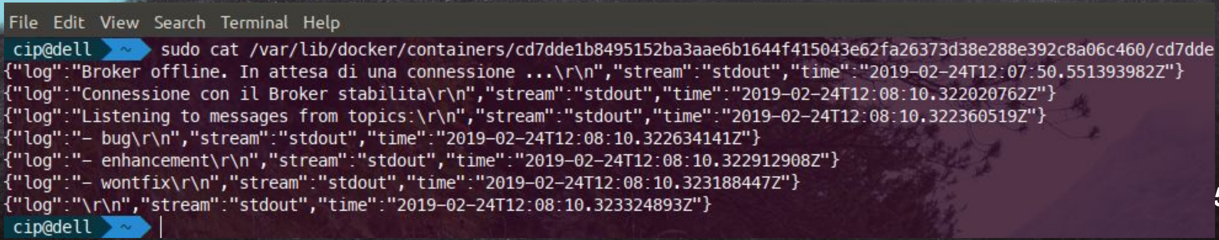
```
{
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "10m"
  }
}
```

In caso questo file non dovesse esistere crearlo con `sudo touch /etc/docker/daemon.json`.
Per ulteriori informazioni riferirsi alla documentazione ufficiale a questo link.

Dockerfile

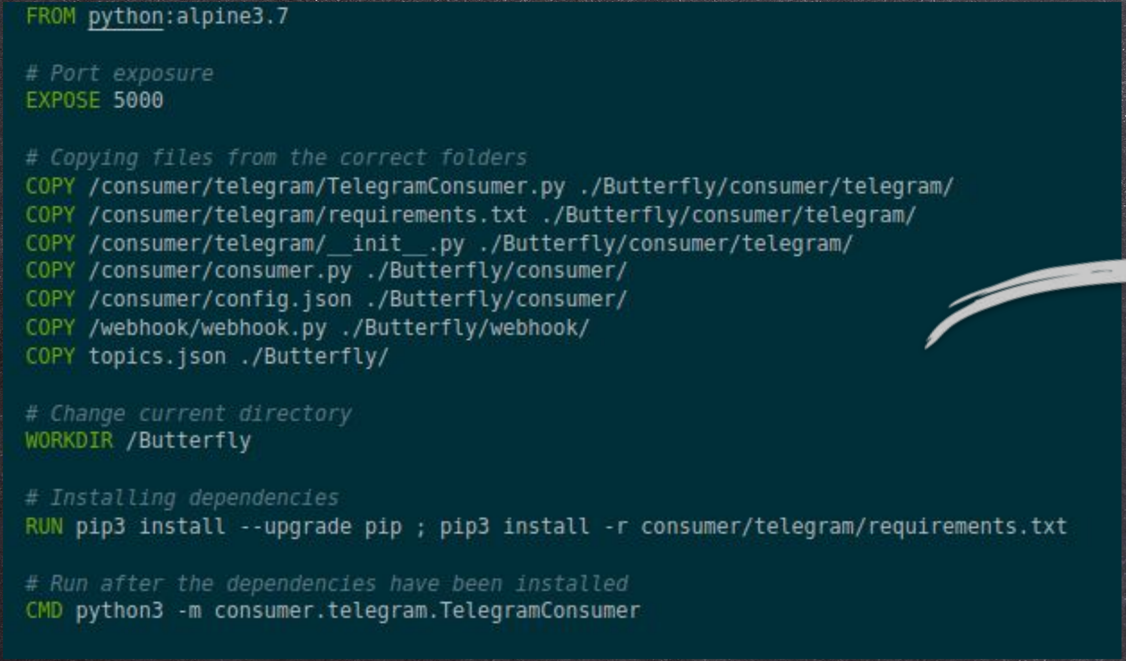
Per costruire le immagini necessarie per ciascun servizio creato da noi eseguire i comandi dall'interno della cartella Butterfly:

```
$ docker build --no-cache --tag consumer_telegram -f path/to/Dockerfile .
```



```
cip@dell ~ sudo cat /var/lib/docker/containers/cd7dde1b8495152ba3aae6b1644f415043e62fa26373d38e288e392c8a06c460/cd7dde
{"log":"Broker offline. In attesa di una connessione ...\r\n","stream":"stdout","time":"2019-02-24T12:07:50.551393982Z"}
{"log":"Connessione con il Broker stabilita\r\n","stream":"stdout","time":"2019-02-24T12:08:10.322020762Z"}
{"log":"Listening to messages from topics:\r\n","stream":"stdout","time":"2019-02-24T12:08:10.322360519Z"}
{"log":"- bug\r\n","stream":"stdout","time":"2019-02-24T12:08:10.322634141Z"}
{"log":"- enhancement\r\n","stream":"stdout","time":"2019-02-24T12:08:10.322912908Z"}
{"log":"- wontfix\r\n","stream":"stdout","time":"2019-02-24T12:08:10.323188447Z"}
{"log":"\r\n","stream":"stdout","time":"2019-02-24T12:08:10.323324893Z"}
cip@dell ~ |
```

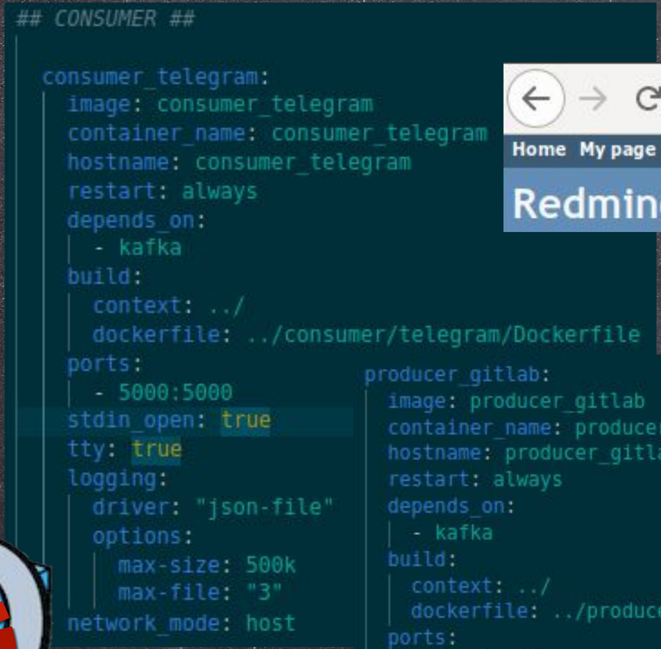
DOCKERFILE



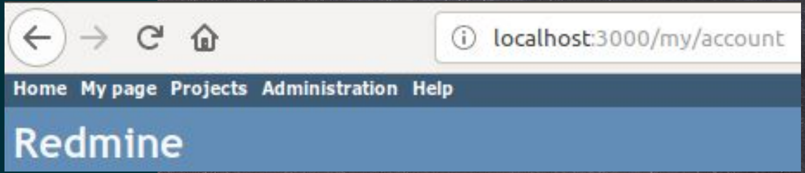
```
FROM python:alpine3.7

# Port exposure
EXPOSE 5000

# Copying files from the correct folders
COPY /consumer/telegram/TelegramConsumer.py ./Butterfly/consumer/telegram/
COPY /consumer/telegram/requirements.txt ./Butterfly/consumer/telegram/
COPY /consumer/telegram/__init__.py ./Butterfly/consumer/telegram/
COPY /consumer/consumer.py ./Butterfly/consumer/
COPY /consumer/config.json ./Butterfly/consumer/
COPY /webhook/webhook.py ./Butterfly/webhook/
COPY topics.json ./Butterfly/

# Change current directory
WORKDIR /Butterfly

# Installing dependencies
RUN pip3 install --upgrade pip ; pip3 install -r consumer/telegram/requirements.txt

# Run after the dependencies have been installed
CMD python3 -m consumer.telegram.TelegramConsumer
```

```
$ docker build --no-cache --tag consumer_telegram -f consumer/telegram/Dockerfile .
```

DOCKERCOMPOSE



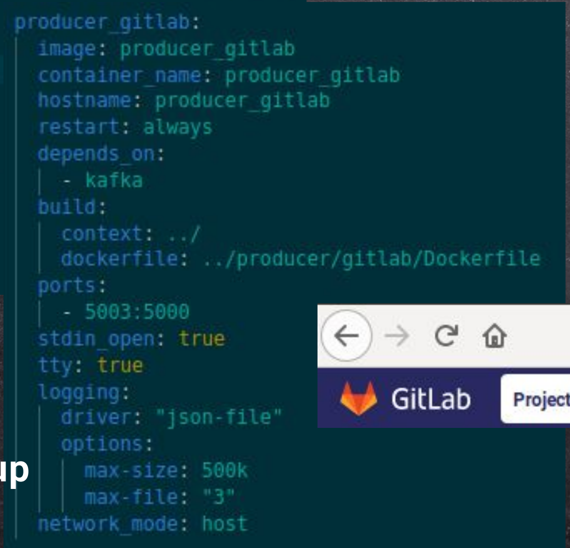
```
## CONSUMER ##

  consumer_telegram:
    image: consumer_telegram
    container_name: consumer_telegram
    hostname: consumer_telegram
    restart: always
    depends_on:
      - kafka
    build:
      context: ../
      dockerfile: ../consumer/telegram/Dockerfile
    ports:
      - 5000:5000
    stdin_open: true
    tty: true
    logging:
      driver: "json-file"
      options:
        max-size: 500k
        max-file: "3"
    network_mode: host
```



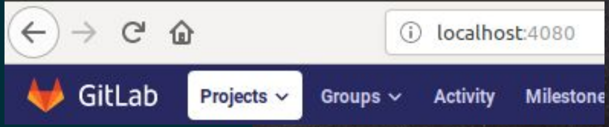
localhost:3000/my/account

Home My page Projects Administration Help

Redmine



```
producer_gitlab:
  image: producer_gitlab
  container_name: producer_gitlab
  hostname: producer_gitlab
  restart: always
  depends_on:
    - kafka
  build:
    context: ../
    dockerfile: ../producer/gitlab/Dockerfile
  ports:
    - 5003:5000
  stdin_open: true
  tty: true
  logging:
    driver: "json-file"
    options:
      max-size: 500k
      max-file: "3"
  network_mode: host
```

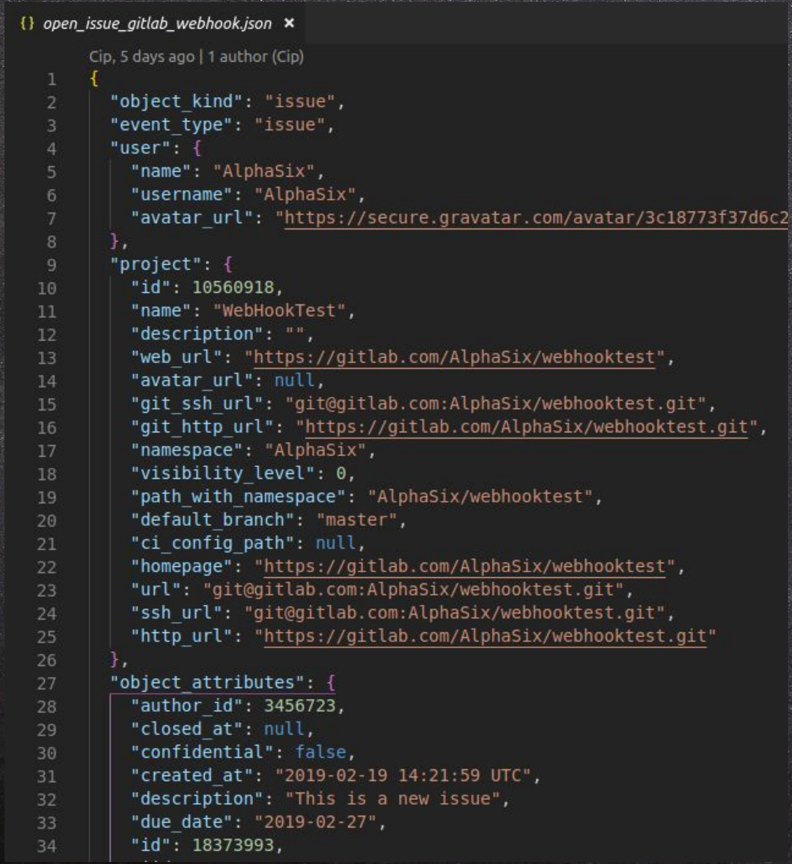


localhost:4080

GitLab Projects Groups Activity Milestone

```
$ docker-compose up
```

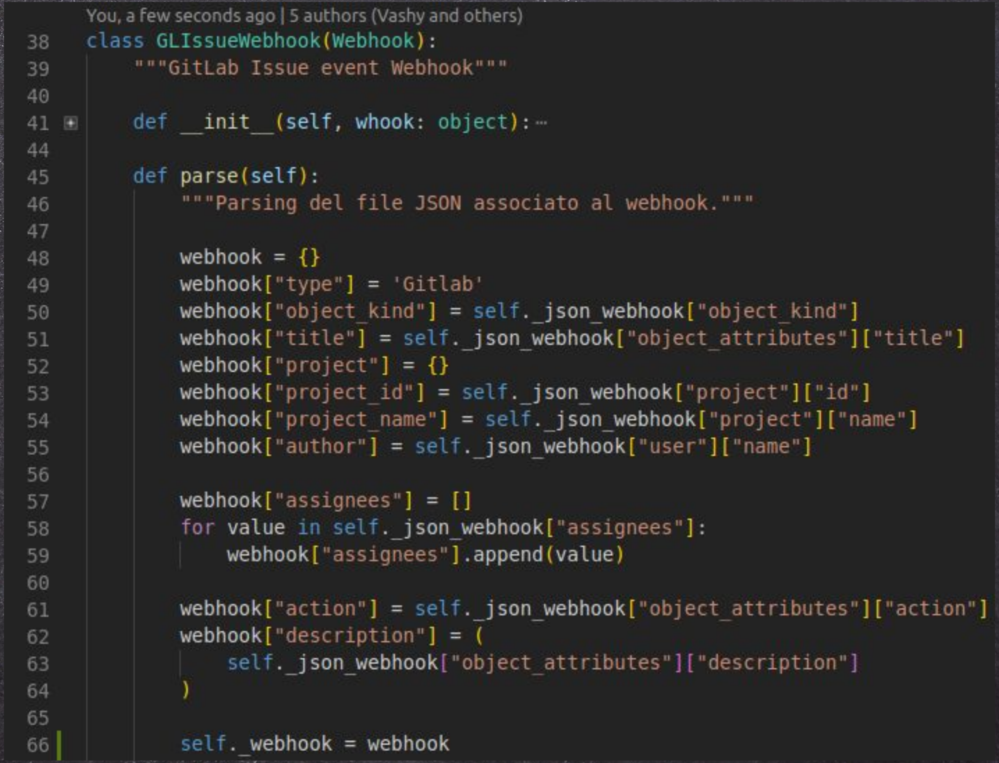
PRODUCER GITLAB



{} open_issue_gitlab_webhook.json ×

Cip, 5 days ago | 1 author (Cip)

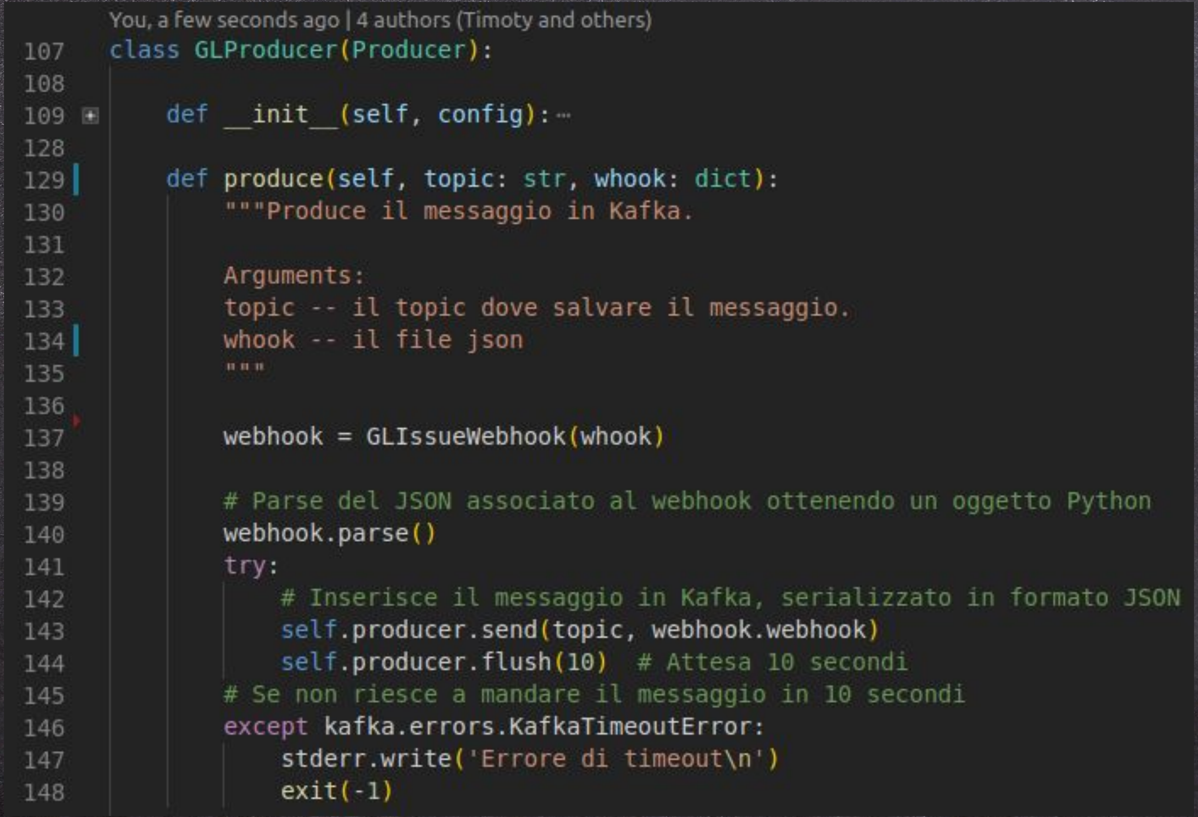
```json
{
  "object_kind": "issue",
  "event_type": "issue",
  "user": {
    "name": "AlphaSix",
    "username": "AlphaSix",
    "avatar_url": "https://secure.gravatar.com/avatar/3c18773f37d6c2
  },
  "project": {
    "id": 10560918,
    "name": "WebHookTest",
    "description": "",
    "web_url": "https://gitlab.com/AlphaSix/webhooktest",
    "avatar_url": null,
    "git_ssh_url": "git@gitlab.com:AlphaSix/webhooktest.git",
    "git_http_url": "https://gitlab.com/AlphaSix/webhooktest.git",
    "namespace": "AlphaSix",
    "visibility_level": 0,
    "path_with_namespace": "AlphaSix/webhooktest",
    "default_branch": "master",
    "ci_config_path": null,
    "homepage": "https://gitlab.com/AlphaSix/webhooktest",
    "url": "git@gitlab.com:AlphaSix/webhooktest.git",
    "ssh_url": "git@gitlab.com:AlphaSix/webhooktest.git",
    "http_url": "https://gitlab.com/AlphaSix/webhooktest.git"
  },
  "object_attributes": {
    "author_id": 3456723,
    "closed_at": null,
    "confidential": false,
    "created_at": "2019-02-19 14:21:59 UTC",
    "description": "This is a new issue",
    "due_date": "2019-02-27",
    "id": 18373993,
```



You, a few seconds ago | 5 authors (Vashy and others)

```python
class GLIssueWebhook(Webhook):
    """GitLab Issue event Webhook"""

    def __init__(self, whook: object):⋯

    def parse(self):
        """Parsing del file JSON associato al webhook."""

        webhook = {}
        webhook["type"] = 'Gitlab'
        webhook["object_kind"] = self._json_webhook["object_kind"]
        webhook["title"] = self._json_webhook["object_attributes"]["title"]
        webhook["project"] = {}
        webhook["project_id"] = self._json_webhook["project"]["id"]
        webhook["project_name"] = self._json_webhook["project"]["name"]
        webhook["author"] = self._json_webhook["user"]["name"]

        webhook["assignees"] = []
        for value in self._json_webhook["assignees"]:
            webhook["assignees"].append(value)

        webhook["action"] = self._json_webhook["object_attributes"]["action"]
        webhook["description"] = (
            self._json_webhook["object_attributes"]["description"]
        )

        self._webhook = webhook
```

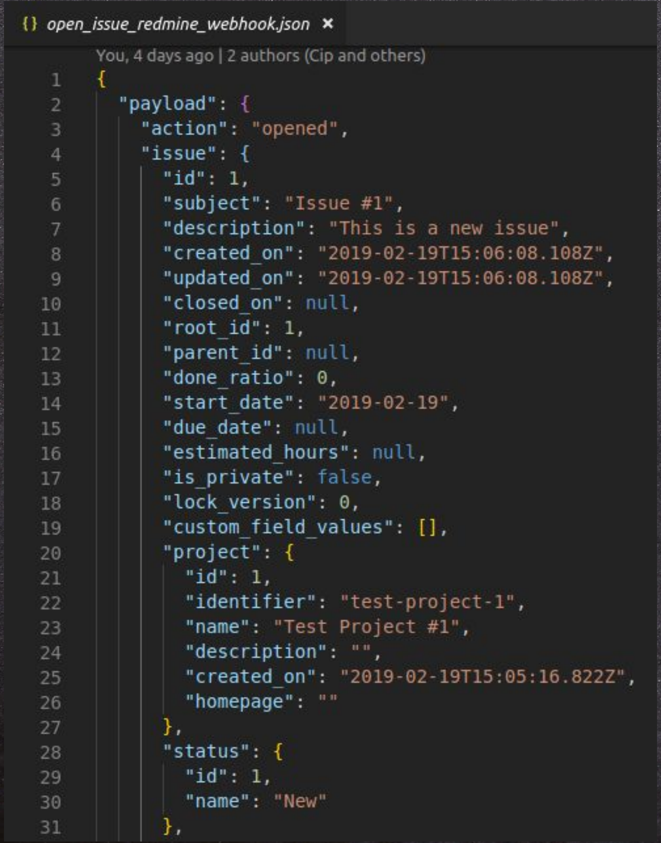
PRODUCER GITLAB



You, a few seconds ago | 4 authors (Timoty and others)

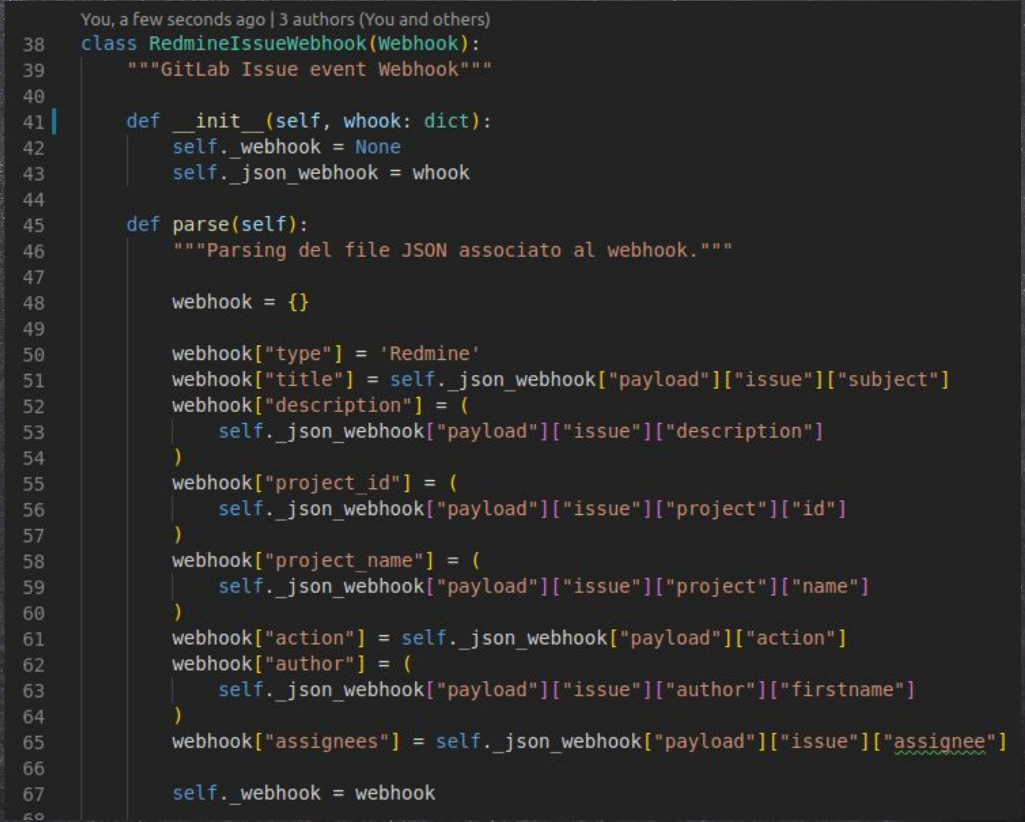
```python
class GLProducer(Producer):

    def __init__(self, config):⋯

    def produce(self, topic: str, whook: dict):
        """Produce il messaggio in Kafka.

        Arguments:
        topic -- il topic dove salvare il messaggio.
        whook -- il file json
        """

        webhook = GLIssueWebhook(whook)

        # Parse del JSON associato al webhook ottenendo un oggetto Python
        webhook.parse()
        try:
            # Inserisce il messaggio in Kafka, serializzato in formato JSON
            self.producer.send(topic, webhook.webhook)
            self.producer.flush(10)  # Attesa 10 secondi
        # Se non riesce a mandare il messaggio in 10 secondi
        except kafka.errors.KafkaTimeoutError:
            stderr.write('Errore di timeout\n')
            exit(-1)
```

PRODUCER REDMINE


{} open_issue_redmine_webhook.json ×

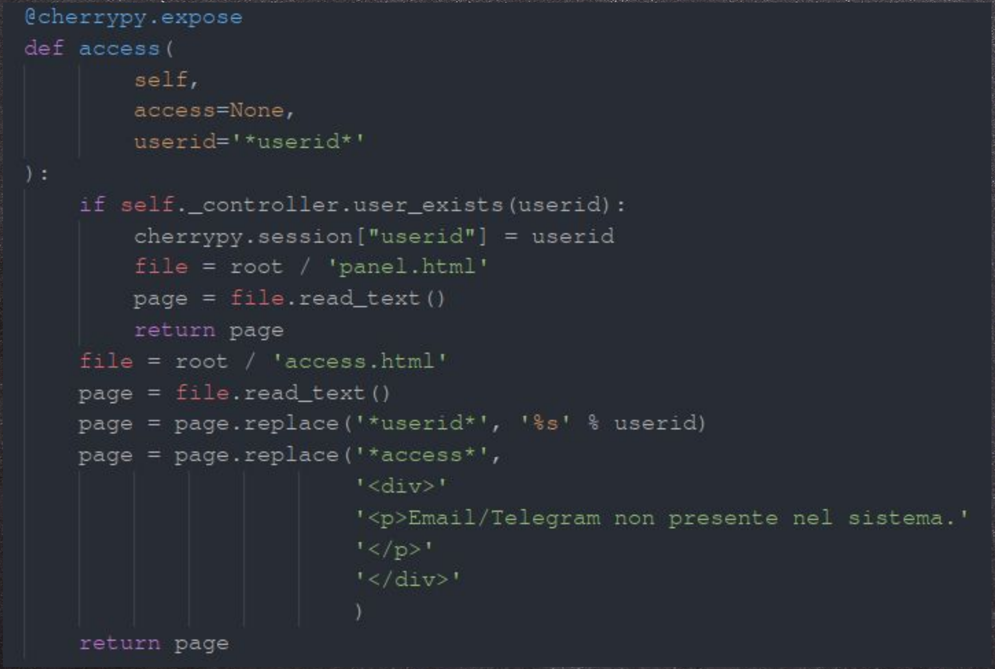
You, 4 days ago | 2 authors (Cip and others)

```json
{
  "payload": {
    "action": "opened",
    "issue": {
      "id": 1,
      "subject": "Issue #1",
      "description": "This is a new issue",
      "created_on": "2019-02-19T15:06:08.108Z",
      "updated_on": "2019-02-19T15:06:08.108Z",
      "closed_on": null,
      "root_id": 1,
      "parent_id": null,
      "done_ratio": 0,
      "start_date": "2019-02-19",
      "due_date": null,
      "estimated_hours": null,
      "is_private": false,
      "lock_version": 0,
      "custom_field_values": [],
      "project": {
        "id": 1,
        "identifier": "test-project-1",
        "name": "Test Project #1",
        "description": "",
        "created_on": "2019-02-19T15:05:16.822Z",
        "homepage": ""
      },
      "status": {
        "id": 1,
        "name": "New"
      },
```

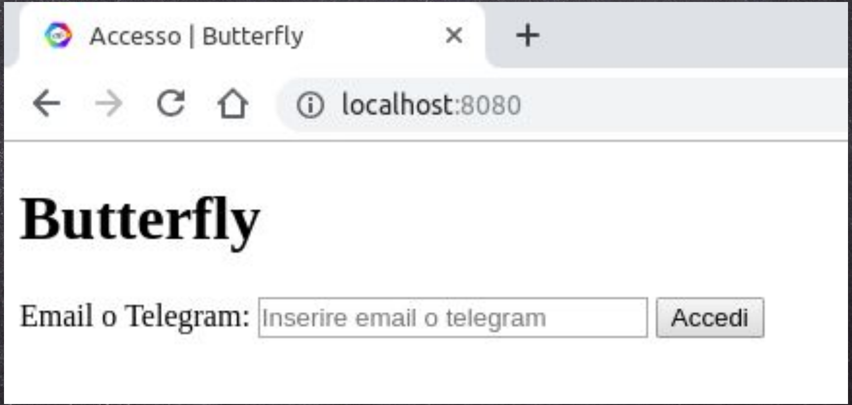

You, a few seconds ago | 3 authors (You and others)

```python
class RedmineIssueWebhook(Webhook):
    """GitLab Issue event Webhook"""

    def __init__(self, whook: dict):
        self._webhook = None
        self._json_webhook = whook

    def parse(self):
        """Parsing del file JSON associato al webhook."""

        webhook = {}

        webhook["type"] = 'Redmine'
        webhook["title"] = self._json_webhook["payload"]["issue"]["subject"]
        webhook["description"] = (
            self._json_webhook["payload"]["issue"]["description"]
        )
        webhook["project_id"] = (
            self._json_webhook["payload"]["issue"]["project"]["id"]
        )
        webhook["project_name"] = (
            self._json_webhook["payload"]["issue"]["project"]["name"]
        )
        webhook["action"] = self._json_webhook["payload"]["action"]
        webhook["author"] = (
            self._json_webhook["payload"]["issue"]["author"]["firstname"]
        )
        webhook["assignees"] = self._json_webhook["payload"]["issue"]["assignee"]

        self._webhook = webhook
```

INTERFACCIA GESTORE PERSONALE



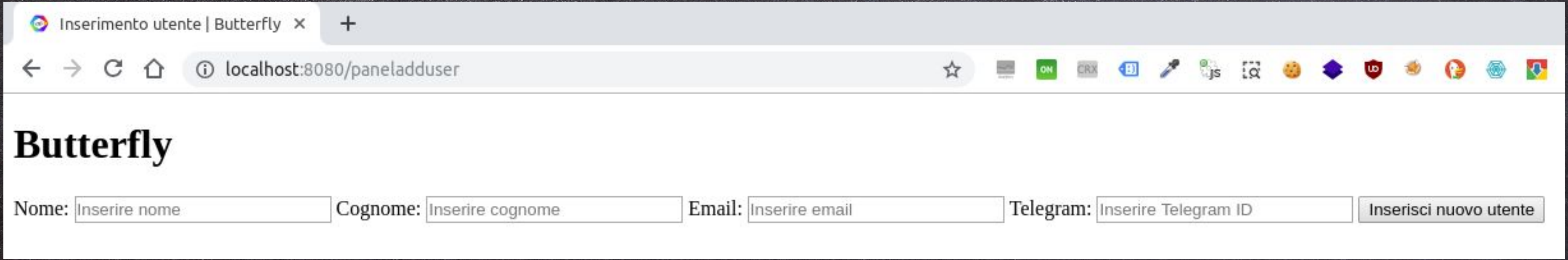
```python
@cherrypy.expose
def access(
        self,
        access=None,
        userid='*userid*'
):
    if self._controller.user_exists(userid):
        cherrypy.session["userid"] = userid
        file = root / 'panel.html'
        page = file.read_text()
        return page
    file = root / 'access.html'
    page = file.read_text()
    page = page.replace('*userid*', '%s' % userid)
    page = page.replace('*access*',
                        '<div>'
                        '<p>Email/Telegram non presente nel sistema.'
                        '</p>'
                        '</div>'
                        )
    return page
```



Butterfly

Email o Telegram:

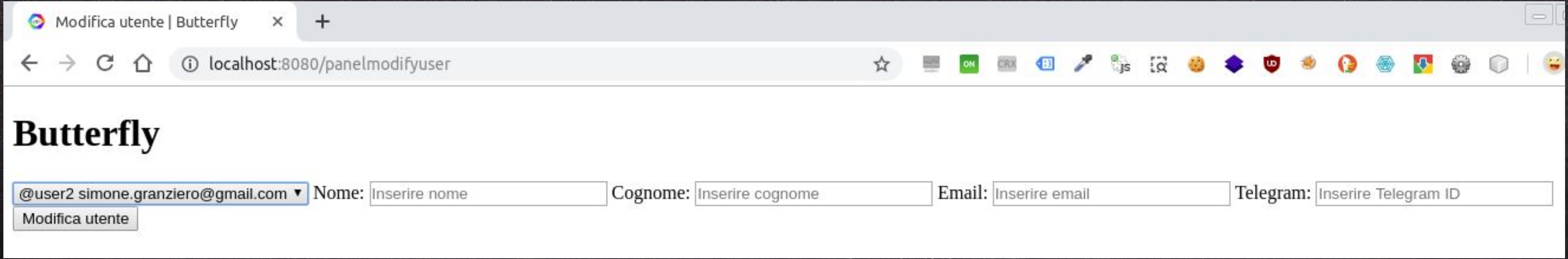
INTERFACCIA GESTORE PERSONALE


Inserimento utente | Butterfly

localhost:8080/paneladduser

Butterfly

Nome: Inserire nome Cognome: Inserire cognome Email: Inserire email Telegram: Inserire Telegram ID Inserisci nuovo utente

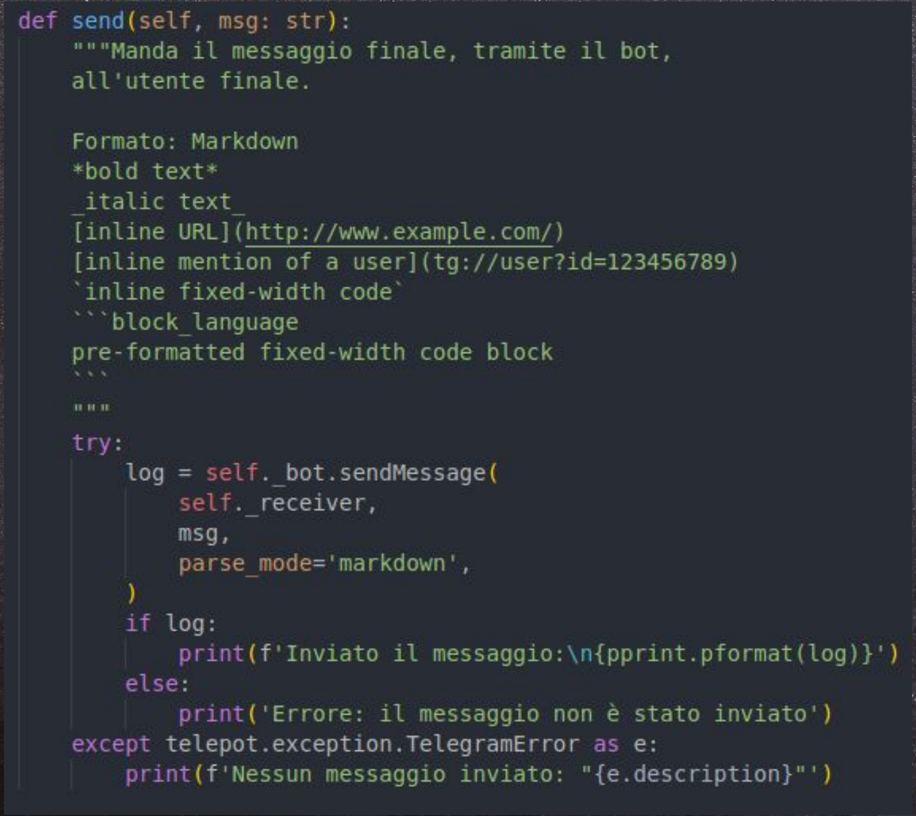

Modifica utente | Butterfly

localhost:8080/panelmodifyuser

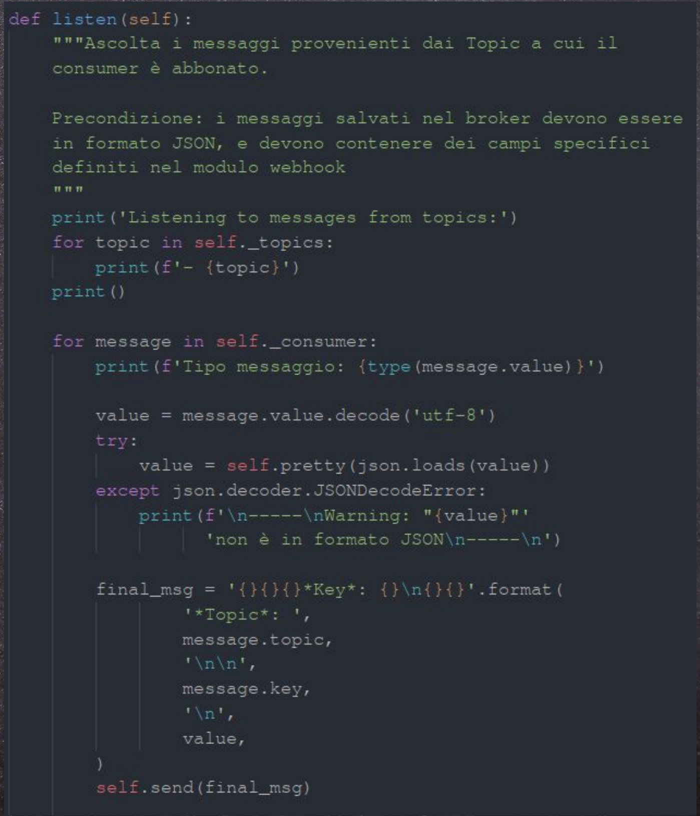
Butterfly

@user2 simone.granziero@gmail.com ▾ Nome: Inserire nome Cognome: Inserire cognome Email: Inserire email Telegram: Inserire Telegram ID
Modifica utente

CONSUMER TELEGRAM

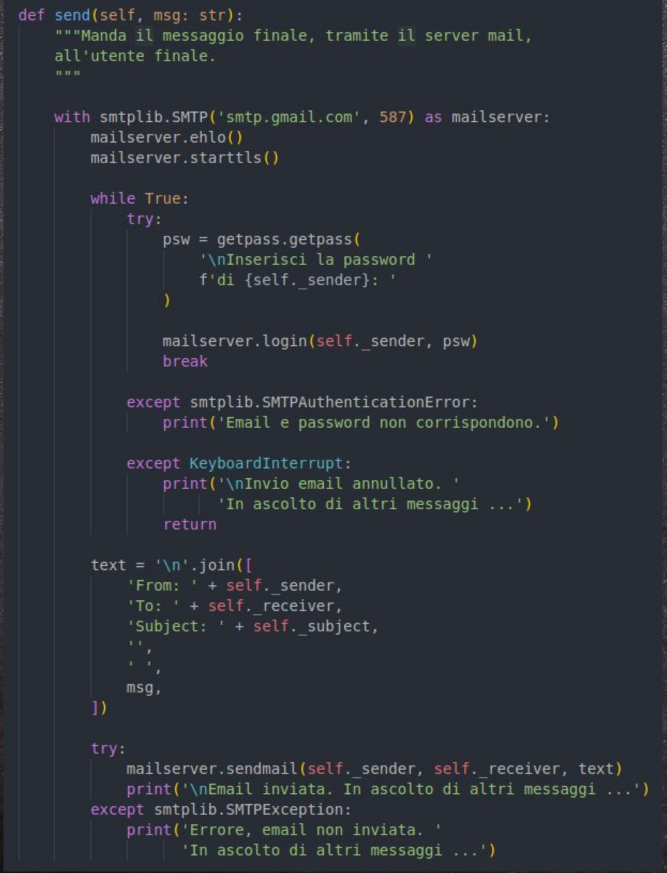


~~~python
def send(self, msg: str):
    """Manda il messaggio finale, tramite il bot,
    all'utente finale.

    Formato: Markdown
    *bold text*
    _italic text_
    [inline URL](http://www.example.com/)
    [inline mention of a user](tg://user?id=123456789)
    `inline fixed-width code`
    ```block_language
    pre-formatted fixed-width code block
    ~~~
    """
    try:
        log = self._bot.sendMessage(
            self._receiver,
            msg,
            parse_mode='markdown',
)
        if log:
            print(f'Inviato il messaggio:\n{pprint.pformat(log)}')
        else:
            print('Errore: il messaggio non è stato inviato')
    except telepot.exception.TelegramError as e:
        print(f'Nessun messaggio inviato: "{e.description}"')

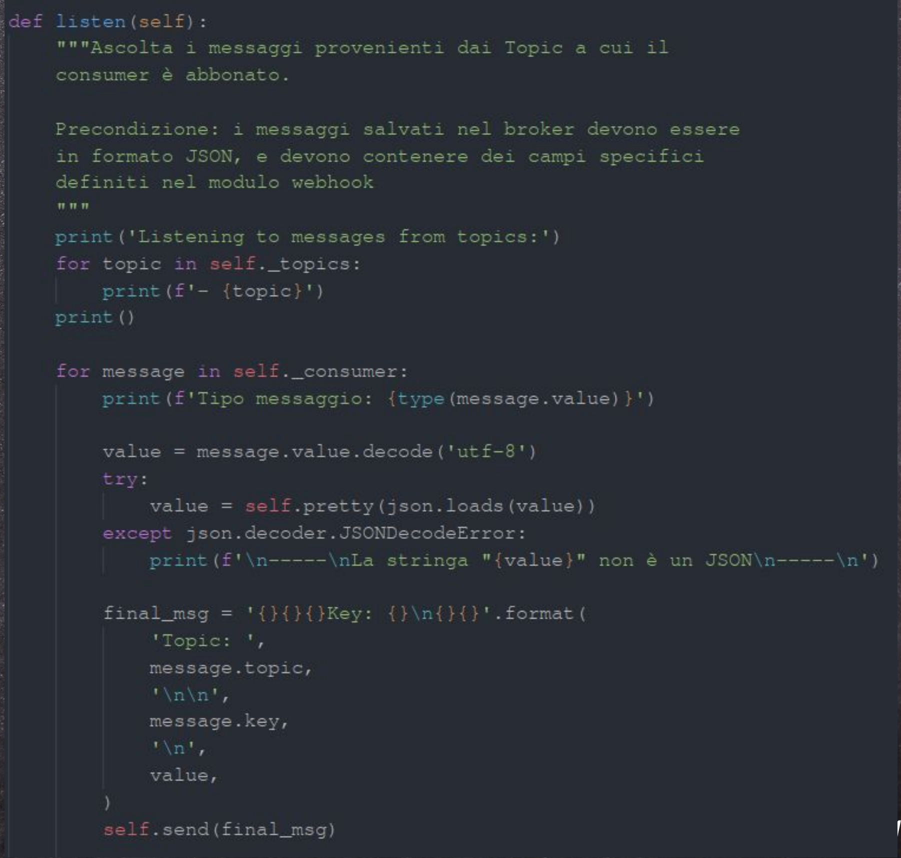


```python
def listen(self):
    """Ascolta i messaggi provenienti dai Topic a cui il
    consumer è abbonato.

    Precondizione: i messaggi salvati nel broker devono essere
    in formato JSON, e devono contenere dei campi specifici
    definiti nel modulo webhook
    """
    print('Listening to messages from topics:')
    for topic in self._topics:
        print(f'- {topic}')
    print()

    for message in self._consumer:
        print(f'Tipo messaggio: {type(message.value)}')

        value = message.value.decode('utf-8')
        try:
            value = self.pretty(json.loads(value))
        except json.decoder.JSONDecodeError:
            print(f'\n-----\nWarning: "{value}"'
                    'non è in formato JSON\n-----\n')

        final_msg = '{}{}{}*Key*: {}\n{}{}'.format(
                '*Topic*: ',
                message.topic,
                '\n\n',
                message.key,
                '\n',
                value,
        )
        self.send(final_msg)
```

CONSUMER EMAIL



```python
def send(self, msg: str):
    """Manda il messaggio finale, tramite il server mail,
    all'utente finale.
    """

    with smtplib.SMTP('smtp.gmail.com', 587) as mailserver:
        mailserver.ehlo()
        mailserver.starttls()

        while True:
            try:
                psw = getpass.getpass(
                    '\nInserisci la password '
                    f'di {self._sender}: '
                )

                mailserver.login(self._sender, psw)
                break

            except smtplib.SMTPAuthenticationError:
                print('Email e password non corrispondono.')

            except KeyboardInterrupt:
                print('\nInvio email annullato. '
                      'In ascolto di altri messaggi ...')
                return

        text = '\n'.join([
            'From: ' + self._sender,
            'To: ' + self._receiver,
            'Subject: ' + self._subject,
            '',
            ' ',
            msg,
        ])

        try:
            mailserver.sendmail(self._sender, self._receiver, text)
            print('\nEmail inviata. In ascolto di altri messaggi ...')
        except smtplib.SMTPException:
            print('Errore, email non inviata. '
                  'In ascolto di altri messaggi ...')
```



```python
def listen(self):
    """Ascolta i messaggi provenienti dai Topic a cui il
    consumer è abbonato.

    Precondizione: i messaggi salvati nel broker devono essere
    in formato JSON, e devono contenere dei campi specifici
    definiti nel modulo webhook
    """
    print('Listening to messages from topics:')
    for topic in self._topics:
        print(f'- {topic}')
    print()

    for message in self._consumer:
        print(f'Tipo messaggio: {type(message.value)}')

        value = message.value.decode('utf-8')
        try:
            value = self.pretty(json.loads(value))
        except json.decoder.JSONDecodeError:
            print(f'\n-----\nLa stringa "{value}" non è un JSON\n-----\n')

        final_msg = '{}{}{}Key: {}\n{}{}'.format(
            'Topic: ',
            message.topic,
            '\n\n',
            message.key,
            '\n',
            value,
        )
        self.send(final_msg)
```