M. Bagiński, F. Kwiatkowski, M. Szewczyk

Własny bot na GG

w oparciu o platformy BotAPI i Google App Engine

Platforma GG BotAPI pozwala na bardzo łatwe stworzenie własnego bota, np. takiego jak Infobot dostępny pod numerem GG:100. Infobot jest przykładem bota informacyjnego, ale boty mogą pełnić również inne funkcje. W artykule opiszemy, jak stworzyć bota czatowego, który będzie mógł służyć do komunikacji wśród grupy współpracowników.

DZIAŁANIE PLATFORMY BOTAPI

Rdzeniem platformy jest Botmaster, który obsługuje w tej chwili ponad 4 tysiące botów i 250 milionów wiadomości miesięcznie. Botmaster to napisana w C++ aplikacja serwerowa, która zapewnia połączenie numerów GG botów z siecią, odbieranie i wysyłanie wiadomości oraz obrazków, a także zmianę statusów i opisów. Twórca bota musi jedynie napisać skrypt, który będzie określał, co na poszczególne wiadomości odpowie bot, oraz umieścić go na publicznym serwerze HTTP. Gdy użytkownik napisze wiadomość do bota, Botmaster połączy się z adresem URL skryptu bota, podając jako parametry GET numer GG bota, numer GG użytkownika oraz wiadomość od niego jako ciało żądania POST. Na podstawie tych danych skrypt powinien przygotować odpowiedź, którą Botmaster odeśle do użytkownika. Odpowiedzią może być czysty tekst, ale jeśli chcielibyśmy użyć tekstu formatowanego czy też obrazków, należy skorzystać z biblioteki BotAPI. Tak działają tzw. wiadomości PULL. Możliwe jest również wysyłanie wiadomości PUSH, które wysyłane są w dowolnym momencie na żądanie twórcy bota. Należy tu zaznaczyć, że platforma BotAPI uniemożliwia wysyłanie wiadomości do dowolnych numerów GG (czyli zwyczajne spamowanie). Aby bot mógł wysyłać wiadomość do użytkownika, to użytkownik musi wcześniej przynajmniej jeden raz napisać do bota. W każdym momencie można zrezygnować z otrzymywania wiadomości PUSH, pisząc do bota WYPISZ. Analogicznie można ponownie włączyć otrzymywanie takich wiadomości, pisząc ZAPISZ. Dotyczy to każdego bota działającego na platformie BotAPI i jest niezależne od skryptu bota.

BIBLIOTEKA BOTAPI

W tej chwili udostępniamy bibliotekę dla języków PHP oraz Python. Można ją pobrać na stronie http://boty.gg.pl/pobierz/. Biblioteka zapewnia obsługę binarnego protokołu BotAPI, który umożliwia formatowanie tekstu, używanie kolorów oraz dodawanie do treści wiadomości obrazków. Protokół jest dokładnie opisany na stronie http://boty.gg.pl/dokumentacja/#2, nic więc nie stoi na przeszkodzie, by napisać własnego bota również w innym języku niż PHP czy Python. W niniejszym artykule skupimy się jednak na Pythonie, gdyż jest to jeden z języków wspieranych przez platformę Google App Engine. Jest to bardzo wygodne rozwiązanie, jeśli nie mamy własnego serwera HTTP, na którym moglibyśmy umieścić skrypt bota.

GOOGLE APP ENGINE

Google App Engine (GAE) to platforma developerska przeznaczona do programowania i hostowania aplikacji. Pozwala ona tworzyć aplikacje internetowe o wysokim natężeniu ruchu bez konieczności zarządzania infrastrukturą obsługującą taki ruch. Do przechowywania danych służy baza nazywana Datastore. Platforma obsługuje języki Python, Java oraz Go. Każda aplikacja uruchomiona na GAE posiada darmowe limity użycia. Limitowane są między innymi ruch wychodzący i wchodzący (na każdy przypada 1GB dziennie) oraz ilość przechowywanych danych w Datastore (maksymalnie 5GB). Limity można zwiększać, opłacając poszczególne zasoby zgodnie z cennikiem.

KONFIGURACJA ŚRODOWISKA PRACY

Do stworzenia bota będziemy potrzebować numeru GG, na którym go uruchomimy. Jeśli nie dysponujemy wolnym numerem, możemy go założyć, wchodząc na stronę https://login.gadu-gadu.pl/account/register. W artykule numerem naszego bota będzie gg:42990004. Musimy również posiadać konto Google. Jeśli jeszcze takiego nie mamy, możemy założyć je, wchodząc na stronę https://accounts.google.com/SignUp. Nasze środowisko pracy będziemy konfigurować w systemie Windows. Najpierw pobieramy i instalujemy Pythona w wersji 2.5.4 dostępnego na stronie http://www.python.org/ftp/python/2.5.4/python-2.5.4.msi. Pobieramy i instalujemy Google App Engine SDK dla Pythona dla Win: https://developers.google.com/appengine/downloads. Kolejnym krokiem będzie stworzenie nowej aplikacji w Google App Engine. W tym celu logujemy się na stronie https://appengine.google.com i wybieramy przycisk *Create Application*. Jeśli nie mamy jeszcze zweryfikowanego konta Google, będziemy musieli w tym momencie wpisać numer naszej komórki poprzedzony kierunkowym do Polski 48 bez znaku +. Po poprawnej weryfikacji za pomocą otrzymanego SMSa powinien otworzyć się formularz tworzenia aplikacji, w którym wypełniamy tylko dwa pola: *Application Identifier* oraz *Application Title*. W tym artykule *botgg42990004* będzie naszym identyfikatorem aplikacji.

HELLO, BOT!

Nasza pierwsza aplikacja, która posłuży jako skrypt bota, będzie zwracać jedynie krótki tekst, ale pokaże nam, jak wygląda

szkielet aplikacji GAE, którą potem rozwiniemy w pełnoprawnego bota czatowego. Najpierw tworzymy na pulpicie folder o nazwie *botgg42990004*. Następnie umieszczamy w nim plik *main.py* przedstawiony na Listingu 1.

Listing 1. Kod źródłowy pliku main.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from google.appengine.ext import webapp
from google.appengine.ext.webapp.util import run_wsgi_app

class Bot(webapp.RequestHandler):
   def post(self):
      self.response.out.write('Hello, Bot!')

application = webapp.WSGIApplication([('/',
Bot)],debug=True)

def main():
   run_wsgi_app(application)

if __name__ == "__main__":
   main()
```

Kolejne linie kodu odpowiadają za zaimportowanie modułów biblioteki GAE. Główna klasa **Bot** posiada tylko jedną metodę **post**, która zostanie wywołana w przypadku żądania typu POST na adres skryptu. Opcjonalnie moglibyśmy oczywiście stworzyć również metodę get dla żądań GET, ale ponieważ Botmaster wysyła żądania typu POST, nie będzie nam ona potrzebna. Osobom, które nie miały do tej pory styczności z Pythonem, przyda się kilka istotnych informacji. Przede wszystkim bardzo ważne są wcięcia w kodzie. To one określają, do jakiego bloku kodu należy dana linia. W Pythonie nie stosuje się do tego, tak jak w wielu innych językach, nawiasów klamrowych. Prawidłowo użyte wcięcia są więc wymuszane przez interpreter Pythona.

Przejdźmy teraz do rejestracji bota na platformie BotAPI GG. Wchodzimy na stronę https://boty.gg.pl/rejestracja/ i wypełniamy wszystkie pola odpowiednimi danymi. Następnie na podany przez nas adres e-mail otrzymamy maila z linkiem aktywującym drugi krok rejestracji. W formularzu podajemy jako adres do wiadomości PULL ten, który zarejestrowaliśmy na Google App Engine - w naszym przypadku będzie to http://botgg42990004.appspot.com/. Musimy również potwierdzić, że to my jesteśmy właścicielami skryptu. W tym celu pobieramy plik HTML, który posłuży do weryfikacji, i umieszczamy go w folderze *botgg42990004*. W naszym przypadku plik ma nazwę *botggafc1248b.html*

Aby uruchomić naszego bota na GAE, potrzebujemy jeszcze pliku konfiguracyjnego o nazwie *app.yaml* przedstawionego na Listingu 2.

Listing 2. Plik konfiguracyjny app.yaml

```
application: botgg42990004
version: 1
runtime: python
api_version: 1

handlers:
- url: /botggafc1248b.html
   static_files: botggafc1248b.html
   upload: botggafc1248b.html

- url: /.*
   script: main.py
```

Przed zapisaniem *app.yaml* w folderze *botgg42990004* podmieniamy nazwę aplikacji na tę, którą zarejestrowaliśmy na GAE oraz nazwę pliku weryfikacyjnego na taką, jaką faktycznie posiada. W przypadku plików typu *yaml* również należy pamiętać o stosowaniu właściwych wcięć. Na koniec uruchamiamy Google App Engine Launcher i wybieramy z menu *File → Add Existing Application*, wskazując ścieżkę do naszego katalogu botgg42990004. Następnie wybieramy przycisk *Deploy* i podajemy dane do konta Google. Po udanym deployu aplikacji możemy wrócić do drugiego kroku rejestracji i wybrać przycisk *Potwierdź teraz*. W ten sposób uruchomiliśmy naszego pierwszego bota GG na platformie BotAPI. Aby sprawdzić, czy działa, wystarczy wysłać wiadomość na numer bota. Odpowie Hello, Bot!

FUNKCJE BOTA

Bot czatowy, który napiszemy, będzie służył do komunikacji między pracownikami jednego działu czy też osób pracujących nad jednym projektem. Bot będzie posiadał zdefiniowaną listę administratorów. Osoby te będą miały uprawnienia do dodawania oraz usuwania numerów GG, które będą otrzymywać wiadomości z bota. Gdy osoba, która została dodana przez administratora, wyśle do bota wiadomość, ta zostanie rozesłana do pozostałych użytkowników bota. Nasz bot będzie również umożliwiał przesyłanie obrazków. W Tabeli 1 znajduje się lista wszystkich funkcji, jakie będą obsługiwane.

BOT CZATOWY

Zastąpimy teraz nasz prosty przykład *Hello, Bot!* skryptem bota, który widać na **Listingach od 5 do 20**. Skrypt składa się z trzech klas - głównej klasy **Bot**, klasy **User** oraz pomocniczej klasy **DailyScrum**. Ta pierwsza zastąpi klasę z poprzedniego przykładu i będzie stanowić główny kod bota. Druga określa model danych, jakie będziemy przechowywać. Dzięki trzeciej klasie zademonstrujemy, jak skorzystać z cyklicznych powiadomień o Daily Scrumie (przydatne dla zespołów pracujących w tej metodyce). W naszym nowym skrypcie importujemy dodatkowo dwa moduły biblioteki BotAPI - **MessageBuilder** oraz **PushConnection**, które ułatwią nam stworzenie bota. Pierwszy implementuje tworzenie wiadomości zgodnej z protokołem BotAPI - zawierającej formatowanie, kolory oraz obrazki zawarte w wiadomości. Drugi odpowiada za autoryzację i wysyłanie wiadomości typu PUSH oraz zmianę statusu i opisu bota. Bibliotekę należy pobrać ze strony https://boty.gg.pl/pobierz/. Po rozpakowaniu archiwum zip z biblioteką pliki *MessageBuilder.py* oraz *PushConnection.py* znajdziemy w katalogu python.

Funkcja	Opis	Dostępność
/pomoc	wyświetla wszystkie dostępne polecenia	administrator/użytkownik
/dodaj <gg>	dodaje nowego użytkownika czatu	administrator
/usuń <gg>	usuwa użytkownika czatu	administrator
/lista	wyświetla użytkowników czatu	administrator/użytkownik
/nick <nick>	ustawia wyświetlaną nazwę użytkownika	administrator/użytkownik
/opis <opis>	zmienia opis bota	administrator/użytkownik

Tabela 1. Lista funkcji bota czatowego

Główna metoda klasy **Bot** - **post** - widoczna na Listingu 9 - jest wywoływana przy każdym żądaniu typu POST naszego skryptu. Dziedziczy ona po klasie **RequestHandler** z modułu **webapp**. Gdy użytkownik wyśle wiadomość do bota, adres skryptu zostanie wywołany z parametrami *from*, oznaczającym numer GG osoby piszącej, oraz *to*, oznaczającym numer GG bota. Treść wiadomości użytkownika

Znacznik	Efekt
[b]tekst pogrubiony[/b]	**tekst pogrubiony**
[i]tekst pochylony[/i]	*tekst pochylony*
[u]tekst podkreślony[/u]	tekst podkreślony
[color=ff0000]kolorowy tekst[/color]	kolorowy tekst
[br]	przejście do nowej linii

Tabela 2. Znaczniki obsługiwane przez metodę addBBcode

zostanie wysłana w ciele żądania POST jako string kodowany UTF-8. Dane te można łatwo wyciągnąć z pola *request*. Metoda **post** sprawdza, o jaką funkcję chodziło użytkownikowi i wywołuje przypisaną do niej metodę bądź, gdy nie jest to funkcja, wywołuje metodę **broadcast** rozsyłającą wiadomość do pozostałych użytkowników bota. W każdej z tych metod sprawdzane są uprawnienia użytkownika do danej czynności przy pomocy metody **permissionGranted**.

KLASA MESSAGEBUILDER

Do stworzenia wiadomości protokołowej, która zostanie wysłana do odbiorcy, należy użyć klasy **MessageBuilder**. Posiada ona między innymi metodę **addBBcode**, która ułatwia tworzenie wiadomości z formatowaniem. Obsługuje ona formatowanie w popularnym standardzie BBCode. W Tabeli 2 znajdują się obsługiwane przez metodę **addBBcode** znaczniki.

Przykładowe użycie metody **addBBcode** widoczne jest na Listingu 19 w metodzie **cmdShowHelp**. Gdy wiadomość będzie gotowa, należy użyć metody **reply**, która spowoduje zwrócenie przez skrypt binarnej wiadomości protokołowej BotAPI. Domyślnie taka wiadomość trafi jako odpowiedź tylko do osoby, która pisała do bota. Możemy jednak użyć metody **setRecipients** (Listing 8), która pozwala na stworzenie listy odbiorców wiadomości. Należy oczywiście pamiętać o tym, że zostanie ona wysłana tylko do osób, które odezwały się wcześniej przynajmniej raz do bota. Klasa **MessageBuilder** pozwala również na dodanie do wiadomości obrazka - zarówno poprzez podanie ścieżki do pliku, jak i podanie danych binarnych. Służy do tego metoda **addImage** (Listing 13).

KLASA PUSHCONNECTION

Klasa **PushConnection** zawiera zestaw przydatnych metod służących do obsługi bota. Aby z nich skorzystać, musimy wcześniej dokonać autoryzacji. W tym celu umieszczamy w kodzie, co widać na Listingu 6, login i hasło do BotAPI, które otrzymaliśmy w mailu informującym o poprawnej rejestracji bota.
Jedną z metod udostępnionych przez klasę **PushConnection** jest **setStatus**, która pozwala zmienić status i opis bota. Jej przykładowe użycie widać na Listingu 18 w metodzie **cmdSetDescription**. Kolejną metodą, którą wykorzystujemy w bocie, jest getImage (Listing 13, metoda **broadcast**) umożliwiająca pobranie obrazków przesłanych w wiadomości od użytkownika do bota. Botmaster pozwala na przesłanie do pięciu obrazków w jednej wiadomości do bota. W połączeniu z metodą **addImage** opisaną wyżej możemy pozwolić użytkownikom bota czatowego na dołączanie do wiadomości obrazków, co widać na Rysunku 1. Klasa **PushConnection** posiada również metodę **push** pozwalającą wysłać na żądanie właściciela bota wiadomość do użytkownika bądź listy użytkowników bota. Metoda ta przyjmuje jako parametr wiadomość zdefiniowaną w obiekcie klasy **MessageBuilder**. Przykład użycia tej metody widać na Listingu 8, który przedstawia klasę **DailyScrum** wysyłającą cykliczne powiadomienia do użytkowników bota czatowego.

CRON W GOOGLE APP ENGINE

Usługa cron w GAE pozwala na skonfigurowanie wykonywania zaplanowanych zadań o określonym czasie lub regularnie. Zadania są automatycznie wywoływane przez crona za pomocą żądań HTTP GET o czasie uprzednio zdefiniowanym w pliku *cron.yaml*. Nasz bot będzie każdego dnia od poniedziałku do piątku o godzinie 10:45 wysyłał do wszystkich osób dodanych do bota wiadomość przypominającą o Daily Scrumie. W tym celu umieszczamy w folderze *botgg42990004* plik *cron.yaml* przedstawiony na Listingu 3.

Listing 3. Plik cron.yaml

```
cron:
- description: Daily Scrum
   url: /scrum
   schedule: every mon,tue,wed,thu,fri 10:45
   timezone: Europe/Warsaw
```

Natomiast w kodzie bota umieszczamy obsługę adresu http://botgg42990004.appspot.com/scrum przedstawioną na Listingu 20 oraz klasę **DailyScrum** widoczną na Listingu 8 zajmującą się tworzeniem i wysłaniem wiadomości.

Rysunek 1. Okno rozmowy na bocie czatowym

GOOGLE APP ENGINE DATASTORE

Datastore to obiektowa, pozbawiona schematu (ang. *schemaless*) baza zapewniająca niezawodne, skalowalne przechowywanie danych. Cechami Datastore są między innymi brak planowanych przestojów, wsparcie dla transakcji atomowych, wysoka dostępność przy odczycie i zapisie oraz silna spójność (ang. *consistency*) przy odczycie. Pracę z Datastore rozpoczynamy od zdefiniowania struktury danych poprzez jej model. Listing 7 przedstawia klasę **User**, która dziedziczy po klasie **db.Model** oraz tworzy jedno pole o nazwie nick. Porównując tradycyjną, relacyjną bazę danych z Datastore oraz biorąc pod uwagę terminologię zaproponowaną przez GAE, to nazwę klasy User będziemy nazywać *kind* i jest ona odpowiednikiem nazwy tabeli, nick to *property* i należy o nim myśleć jako o kolumnie tabeli. Wpisy powiązane z danymi, które będziemy umieszczać

w bazie, określane są jako *entities*. Tak jak w przypadku tradycyjnych baz danych musimy zdefiniować typ danych przechowywanych w *property*. W naszym przypadku będzie to StringProperty, który może przechowywać w unicode do 500 znaków. Każdy wpis powiązany z danymi (*entity*) posiada swój własny unikalny klucz (*key name*), który można zdefiniować przy zapisie danych do Datastore. Skorzystamy z tej możliwości i jako klucz będziemy ustawiać numer GG użytkownika bota, co widać na Listingu 14. Datastore API udostępnia dwa interfejsy służące do tworzenia zapytań o dane. Pierwszy interfejs jest obiektowy, udostępniony przez klasę **Query**, która wystawia zestaw metod pozwalających na pobieranie danych. Na Listingu 16 przedstawiona jest metoda ustawiająca nick, która korzysta tylko z dwóch metod tej klasy: **filter** oraz **get. Filter** pozwala wyciągać dane spełniające określony warunek, korzystając z operatorów. Natomiast **get** wykonuje zapytanie i zwraca pierwszy znaleziony wynik lub None. Drugi interfejs udostępnia klasa **GqlQuery**, która pozwala na tworzenie zapytań w języku GQL. Jego składnia podobna jest do języka SQL, co można zobaczyć na Listingu 11 zawierającym metodę **getUserData**. Obie klasy Query i GqlQuery są zdefiniowane w module *google.appengine.ext.db*. Datastore pozwala również na zakładanie własnych indeksów na kolumnach. Aby je określić, umieszczamy w folderze *botgg42990004* plik konfiguracyjny o nazwie *index.yaml* przedstawiony na Listingu 4. Linia *direction* określa kierunek sortowania wpisów – w tym przypadku jest to ascending, czyli rosnąco.

Listing 4. Plik index.yaml

```
indexes:

- kind: User
   properties:
   - name: __key__
      direction: asc
   - name: nick
      direction: asc
```

Poniżej znajduje się cały kod bota czatowego podzielony na listingi omówione w artykule. Należy pamiętać o poprawnym ustawieniu zmiennych globalnych widocznych na Listingu 6 oraz o wcięciach.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
```

Listing 5. Importowanie potrzebnych bibliotek

```
import re
from google.appengine.ext import webapp
from google.appengine.ext.webapp.util import run_wsgi_app
from google.appengine.ext import db
from google.appengine.api.datastore import Key
from MessageBuilder import *
from PushConnection import *
```

Listing 6. Definicje zmiennych globalnych

```
BOT_GG = 42990004
PushConnection.BOTAPI_LOGIN = 'wojtek@gg.pl'
PushConnection.BOTAPI_PASSWORD = 'bNpOPRFZ3dCWm4Jj'
ADMIN_LIST = [16303,16305,16492]
PERMISSION_IS_ADMIN = 1
PERMISSION_IS_ADDED = 2
PERMISSION_HAS_NICK = 4
```

Listing 7. Klasa definiująca model danych

```
class User(db.Model):
   nick = db.StringProperty(multiline=False)
```

Listing 8. Klasa wysyłająca cykliczne powiadomienia

```
class DailyScrum(webapp.RequestHandler):
   def get(self):
      M = MessageBuilder()
      P = PushConnection(BOT_GG)
      M.addBBcode('[b]<[color=0000ff]Bot[/color]>[/b] 10:45 - czas na Daily Scrum')
      recipients = Bot.getUsers().keys()
      M.setRecipients(recipients)
      P.push(M)
```

Listing 9. Główna klasa bota

```
class Bot(webapp.RequestHandler):
   def post(self):
      self.msg = self.request.body
      self.uin = self.request.get('from')
      self.M = MessageBuilder()

   functions = {
      'dodaj' : self.cmdAddUser,
      'usun' : self.cmdDelUser,
      'nick' : self.cmdSetNick,
      'lista' : self.cmdListUsers,
      'opis' : self.cmdSetDescription,
```

```
        'pomoc' : self.cmdShowHelp
    }
    r = re.compile(r'^\/(dodaj|usu[nń]|nick|lista|opis|pomoc)')
    arguments = ''
    function = ''

    if r.search(self.msg):
        function, arguments = (re.sub(r'^/','',self.msg.strip()).split(' ',1) + [''])[:2]
        if function == 'usuń':
            function = 'usun'
        functions[function](arguments.strip())
    else:
        self.broadcast(self.uin, self.msg)
    self.response.out.write(self.M.reply(self))
```

Listing 10. Sprawdzenie uprawnień użytkownika

```
    def permissionGranted(self, mask):
        if mask & PERMISSION_IS_ADMIN and int(self.uin) not in ADMIN_LIST:
            self.M.addBBcode('[b][color=ff0000]Nie masz uprawnień administratora.[/color][/b]')
            return False
    elif (mask & PERMISSION_IS_ADDED or mask & PERMISSION_HAS_NICK) and not self.userExists(self.uin):
        self.M.addBBcode('[b][color=ff0000]Nie masz uprawnień do wysyłania wiadomości.[/color][/b]')
        return False
    elif mask & PERMISSION_HAS_NICK and not self.userHasNick(self.uin):
        self.M.addBBcode('[b][color=0000ff]Skorzystaj najpierw z polecenia[/color] [color=008800]/nick[/color][/b]')
        return False
    else:
        return True
```

Listing 11. Metody określające, czy użytkownik istnieje i czy ma ustawionego nicka

```
    def getUserData(self, uin):
    return db.GqlQuery("SELECT * FROM User WHERE __key__ = KEY('User', :1)", uin)

    def userExists(self, uin):
        q = self.getUserData(uin)
     return q.count() == 1

    def userHasNick(self, uin):
        q = self.getUserData(uin)
        return q.count() == 1 and q.get().nick is not None

    def userNick(self, uin):
        nick = ''
        q = self.getUserData(uin)
        if q.count() == 1 and q.get().nick is not None:
            nick = q.get().nick.encode('utf-8')
        return nick
```

Listing 12. Metoda zwracająca strukturę z użytkownikami

```
    @staticmethod
    def getUsers():
        q = db.GqlQuery("SELECT nick FROM User WHERE nick!=NULL")
        users = {}
        if q.count() > 0:
            for p in q.run():
                users[p.key().name().encode('utf-8')] = p.nick.encode('utf-8')
        return users
```

Listing 13. Metoda rozsyłająca wiadomość do użytkowników bota

```
    def broadcast(self, uin, message):
        if self.permissionGranted(PERMISSION_HAS_NICK):
            sender = self.userNick(uin)
            recipients = self.getUsers().keys()
            self.M.setRecipients(recipients)
            self.M.addBBcode('[b]<[color=0000ff]%s[/color]>[/b] %s' % (sender, message))

            if len(self.request.get('images')) > 0:
                P = PushConnection(BOT_GG)
                imgs = self.request.get('images').split(',')
                for img in imgs:
                    self.M.addImage(P.getImage(img), IMG_RAW)
```

Listing 14. Metoda dodająca nowego użytkownika

```
    def cmdAddUser(self, arguments):
        if self.permissionGranted(PERMISSION_IS_ADMIN):
            r = re.compile(r'^([0-9]+)$').search(arguments)
            if r is None:
                self.M.addBBcode('[b][color=0000ff]Użycie:[/color][/b][br]')
                self.M.addBBcode('[b][color=008800]/dodaj[/color] numer gg[/b]')
            else:
                uin = r.groups()[0]
                if self.userExists(uin):
                    self.M.addBBcode('[b][color=0000ff]Użytkownik został już wcześniej dodany.[/color][/b]')
```

```
            else:
               newUser = User(key_name=uin)
               newUser.put()
               self.M.addBBcode('[b][color=0000ff]Użytkownik został dodany.[/color][/b]
```

Listing 15. Metoda usuwająca istniejącego użytkownika

```
    def cmdDelUser(self, arguments):
       if self.permissionGranted(PERMISSION_IS_ADMIN):
          r = re.compile(r'^([0-9]+)$').search(arguments)
          if r is None:
             self.M.addBBcode('[b][color=0000ff]Użycie:[/color][/b][br]')
             self.M.addBBcode('[b][color=008800]/usuń[/color] numer gg[/b]')
          else:
             uin = r.groups()[0]
             if not self.userExists(uin):
                self.M.addBBcode('[b][color=0000ff]Użytkownik nie został wcześniej dodany.[/color][/b]')
                self.M.addBBcode('[br][b][color=0000ff]Napisz:[/color] [color=008800]/lista[/color][/b]')
             else:
                key = db.Key.from_path('User',uin)
                db.delete(key)
                self.M.addBBcode('[b][color=0000ff]Użytkownik został usunięty.[/color][/b]')
```

Listing 16. Metoda ustawiająca nick

```
    def cmdSetNick(self, arguments):
       if self.permissionGranted(PERMISSION_IS_ADDED):
          r = re.compile(r'^([a-zA-Z0-9ąćęłńóśżźĄĆĘŁŃÓŚŻŹ ]+)$').search(arguments)
          if r is None:
             self.M.addBBcode('[b][color=0000ff]Użycie:[/color][/b][br]')
             self.M.addBBcode('[b][color=008800]/nick[/color] nazwa użytkownika[/b]')
          else:
             nick = r.groups()[0]
             q = db.Query(User)
             q = q.filter('nick =', nick.decode('utf-8'))
             result = q.get()
             if result is not None:
                self.M.addBBcode('[b][color=0000ff]Ten nick jest już zajęty.[/color][/b]')
             else:
                newNick = User(key_name=str(self.uin), nick=nick.decode('utf-8'))
                newNick.put()
                self.M.addBBcode('[b][color=0000ff]Twój nick został ustawiony na:[/color] %s[/b]' % nick)
```

Listing 17. Metoda wyświetlająca listę użytkowników

```
    def cmdListUsers(self, arguments):
       if self.permissionGranted(PERMISSION_HAS_NICK):
          users = self.getUsers()
          self.M.addBBcode('[b][color=0000ff]Lista użytkowników bota:[/color][/b]')
          for uin, nick in users.iteritems():
             self.M.addBBcode("[br][b][color=008800]%s[/color] - [u]gg:%s[/u][/b]" % (nick,uin))
```

Listing 18. Metoda ustawiająca opis bota

```
    def cmdSetDescription(self, arguments):
       if self.permissionGranted(PERMISSION_HAS_NICK):
          if len(arguments) > 0:
             P = PushConnection(BOT_GG)
             P.setStatus(arguments, STATUS_BACK)
             self.M.addBBcode('[b][color=0000ff]Opis został ustawiony na:[/color] %s[/b]' % arguments)
          else:
             self.M.addBBcode('[b][color=0000ff]Użycie:[/color][/b][br]')
             self.M.addBBcode('[b][color=008800]/opis[/color] tekst[/b]')
```

Listing 19. Metoda wyświetlająca listę dostępnych poleceń

```
    def cmdShowHelp(self, arguments):
       if self.permissionGranted(PERMISSION_IS_ADDED):
          self.M.addBBcode('[b][color=0000ff]Dostępne polecenia:[/color][/b]')
          if int(self.uin) in ADMIN_LIST:
             self.M.addBBcode('[br][b][color=008000]/dodaj[/color][/b] - dodaje nowego użytkownika')
             self.M.addBBcode('[br][b][color=008000]/usuń[/color][/b] - usuwa istniejącego użytkownika')
          self.M.addBBcode('[br][b][color=008800]/nick[/color][/b] - ustawia nick')
          self.M.addBBcode('[br][b][color=008000]/lista[/color][/b] - wyświetla listę użytkowników')
          self.M.addBBcode('[br][b][color=008000]/opis[/color][/b] - ustawia opis bota')
          self.M.addBBcode('[br][b][color=008000]/pomoc[/color][/b] - wyświetla pomoc')
```

Listing 20. Lista aplikacji Web Server Gateway Interface

```
application = webapp.WSGIApplication([('/', Bot),('/scrum', DailyScrum)],debug=True)

def main():
   run_wsgi_app(application)

if __name__ == "__main__":
   main()
```

Wszystkie potrzebne do stworzenia bota czatowego pliki z powyższego przykładu można pobrać ze strony: http://boty.gg.pl/bot_programistamag.zip. Zainteresowanych stworzeniem własnego projektu w oparciu o platformy BotAPI i Google App Engine zachęcamy do zapoznania się z poniższymi linkami.

W sieci

- http://boty.gg.pl/ – dokumentacja BotAPI, biblioteka oraz przykłady
- https://developers.google.com/appengine/ – strona Google App Engine
- http://forum.gg.pl/forumdisplay.php?54-Boty – oficjalne forum poświęcone platformie BotAPI GG
- http://forum.gg.pl/showthread.php?621 – szczegółowy tutorial opisujący jak założyć bota na platformie BotAPI GG korzystając z Google App Engine

Marcin Bagiński xmoki9@gmail.com

Pracownik GG Network. Na co dzień programuje boty. Tworzy oprogramowanie w PHP i C++ od 10 lat.

Filip Kwiatkowski filip.kwiatkowski@gmail.com

Absolwent Informatyki na Uniwersytecie Marii Curie-Skłodowskiej w Lublinie. Współtwórca projektu Infobot.pl. Obecnie pracuje w GG Network jako programista C++ i poza Infobotem zajmuje się również rozwojem Botmastera. W wolnym czasie fotografuje, jeździ na rowerze oraz buduje z Lego Technic i Mindstorms.

Maciej Szewczyk rodion.infobot@gmail.com

Prawnik z wykształcenia, project manager z wyboru. Współtwórca projektu Infobot.pl. W GG Network odpowiedzialny jest za rozwój platformy BotAPI oraz niestandardowe kampanie reklamowe i promocyjne w oparciu o boty. Prywatnie mąż i tata Adasia.