УДК 004.4
Лавренчук С.В., Ілюшик Р.С.
Луцький національний технічний університет

ПЕРЕВАГИ СТВОРЕННЯ REAL-TIME ДОДАТКІВ НА БАЗІ АСИНХРОННОЇ БІБЛІОТЕКИ CHANNELS ТА ФРЕЙМВОРКА DJANGO

Лавренчук С.В., Ілюшик Р.С. Переваги створення real-time додатків на базі асинхронної бібліотеки Channels та фреймворка Django. У статті розглянуто сучасні технології розробки web-додатків, заокрема досліджено переваги використання асинхронної бібліотеки Channels над іншими технологіями для створення інтерактивних додатків.
Ключові слова: Django, Channels, python, real-time додатки, WebSocket

Лавренчук С.В., Илюшик Р.С. Преимущества создания real-time приложений на базе асинхронной библиотеки Channels и фреймворка Django. В статье рассмотрены современные технологии разработки web-приложений, в особенности исследованы преимущества использования асинхронной библиотеки Channels над другими технологиями для создания интерактивных приложений.
Ключевые слова: Django, Channels, python, real-time приложения, WebSocket

Lavrenchuk S.V., Iliushyk R.S. Advantages of creating real-time applications based on the asynchronous library of Channels and the framework of Django. The article discusses modern technologies for developing web applications, and explores the advantages of using the Asynchronous Library Channels over other technologies to create interactive applications.
Keywords: Django, Channels, python, real-time applications, WebSocket

Постановка наукової проблеми та аналіз досліджень.

Останнім часом активно розвиваються веб-технології та засобів розробки сайтів. Якщо ще десять років тому використовувалися нескладні динамічні веб-сторінки з перезавантаженням при кожному запиті до сервера, то з появою Ajax (Asynchronous JavaScript And XML) веб-сторінки стали формуватися в фоновому режимі, а не перезавантажуватися повністю. Це дало змогу зменшити навантаження на сервер, економити трафік, пришвидшити оновлення сторінок і додати їм інтерактивності. Саме в цей час починає розвиватися і широко впроваджуватися web 2.0 – користувачі Інтернету стають активними: з'являються коментарі на сайтах, форуми, блоги, соціальні мережі тощо. При цьому логіка обробки даних ускладнюється, тому що частина процесів переноситься на сторону клієнта, тому з'являються веб-фреймворки – програмні каркаси, які спрощують розробку за рахунок автоматизації деяких процесів та позбавляють від необхідності написання рутинного стандартного коду. Поступово веб-сайти перетворюються в повноцінні інтерактивні додатки, (наприклад, Google Docs можна використовувати замість Microsoft Word), з'являються 3D-ігри, можливості браузерів щораз розширюються. Разом з тим, деякі розробники вже задумуються про web 3.0 [6], який би функціонував у вигляді веб-платформи як гібриду операційної системи та соціальної мережі, дозволив би вирішити проблеми копірайту та плагіату, безслідного видалення своїх даних з мережі тощо. Тому при створенні сайтів не доцільно використовувати застарілі технології, потрібно впроваджувати ефективні сучасні засоби.

За даними сайту dou.ua [5], більшість новачків в Україні, які планують вивчати програмування і взяли участь в інтернет-опитуванні, хочуть вивчати саме Python в 2018 році, крім того що популярність цієї мови зростає, вона стала домінуючою платформою в Data Science (при аналізі великих об'ємів даних).

Найпопулярнішим фреймворком для Python є Django [3], він використовується в таких великих і відомих сайтах, як Instagram, YouTube, Google та ін., але при створені інтерактивних веб-додатків за допомогою мови програмування Python, перевагу віддають не Django, а фреймворкам типу Tornado, Twisted, побудованих на асинхронних підходах (подієво-орієнтованих).

WebSockets – це сучасна технологія, яка дає можливість запускати інтерактивні з'єднання між браузером користувача і сервером. З цим інтерфейсом, можна надсилати повідомлення на сервер і отримувати назад відповідь, керовану подіями (event-driven responses), без потреби робити повторні запити на сервер.

Завдяки проекту Channels ми можемо поєднувати асинхронний код з синхронним ядром Django, що дозволяє проектам Django обробляти не тільки HTTP, але й протоколи, що потребують тривалих підключень, зокрема і WebSockets.

Виклад основного матеріалу й обґрунтування отриманих результатів.

За замовчуванням веб-додаток створений за допомогою Django наслідує модель типу запит-відповідь: запит надходить, він направляється на відповідний сервер, сервер генерує відповідь, відповідь надсилається клієнту, і все виконується в одному процесі. Це забезпечує WSGI інтерфейс (Web Server Gateway Interface) — стандарт взаємодії між Python-програмою, яка виконується на стороні сервера, і самим веб-сервером.

Асинхрона бібліотека Channels для Django (сумісна, як з Python v3.x, так і з Python 2.7), умовно кажучи, змінює модель роботи програми з "запит-відповідь" на "подія-реакція" (не потрібно плутати зі стандартною реалізацією подій в Django) [7]. Це означає, що запит по протоколу HTTP, повідомлення, отримане через WebSocket, а також, наприклад, вхідний лист на електронну пошту або звичайне SMS – все це буде представлено для додатка як "повідомлення", на яке можна відреагувати. Кожне повідомлення приходить на певний канал (залежить від типу повідомлення). При цьому можлива, але не обов'язкова, відправка листа у відповідь, яка автоматично буде перетворена до потрібного формату (відповіді по протоколу HTTP, повідомленням через веб-сокети і.т.д). Відправлення відповіді відбувається за відповідним каналом. Все це відбувається абсолютно прозоро і розробнику потрібно турбуватися тільки про отримання потрібних даних з повідомлень і виконання потрібних дій у відповідь. Звичайно, при відправці листа у відповідь розробнику доведеться дотримуватися формату для конкретного протоколу (наприклад, задати код статусу для HTTP відповіді). Проте, дана абстракція значно полегшує розробку проекту, котрий використовує, наприклад, Websockets.

Channels побудовано на інтерфейсі ASGI (інтерфейс асинхронного серверного шлюзу), який є спадкоємцем WSGI. Бібліотека Channels використовує багатошарову архітектуру, запит по HTTP або повідомлення через WebSocket проходить через наступні рівні:

- Рівень інтерфейсу(Web Server). Це обробники звичних нам протоколів взаємодії між додатком і сервером, наприклад WebSocket, WSGI.
- Рівень каналу(Channel Layer): тобто, брокер. В якості даного рівня можуть виступати Redis, SQL база даних, область пам'яті або власна структура даних.
- Рівень оброблювачів(Worker Processes): процеси, що стежать за надходженням повідомлень в канал (черга) і реагують на них тим чи іншим чином (зазвичай викликом відповідних функцій-обробників).

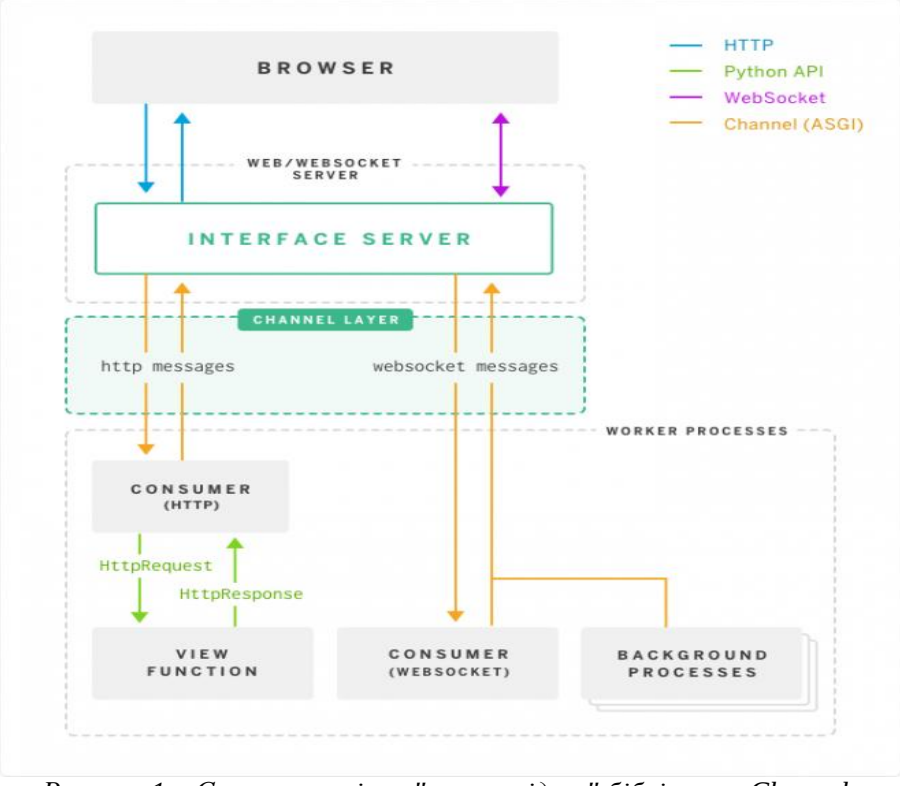

Рисунок 1 – Схема механізму "запит-відгук" бібліотеки Channels

Розробка і підтримка додатків в даному випадку полегшується за рахунок можливості використання «routing», сесій для каналів, розбиття їх по групах, оповіщення клієнтів про зміну даних на сервері.

За замовчуванням додаток Django Channels дозволяє легко використовувати режим налагодження в IDE Pycharm і налаштувати точки зупину в будь-якій частині коду. Також, за рахунок багатошарової архітектури можливе більш детальне налаштування сервера під конкретні потреби.

Зазвичай, Django використовує HTTP для спілкування між клієнтом і сервером: Клієнт надсилає HTTP-запит на сервер. Джанго аналізує запит, витягує URL-адресу, а потім порівнює його з представленням. Сервер обробляє запит і повертає HTTP-відповідь клієнту. На відміну від HTTP, протокол WebSockets дозволяє двонаправлене спілкування, тобто сервер може надсилати дані клієнту без запиту користувача. За допомогою HTTP лише клієнт, який зробив запит, отримує відповідь. За допомогою WebSockets сервер може спілкуватися з кількома клієнтами одночасно.

Отже, Django Channels створюють простий механізм написання веб-програм у Django, які підтримують протокол HTTP2 та Websocket. Це дає змогу впроваджувати асинхронний вміст (чат-кімнати та живі канали, наприклад, за допомогою яких можна робити оновлення без необхідності опитування) в уже наявні на Django сайти.

Серед основних переваг даної асинхроної бібліотеки є її швидкодія та легкість застосування в порівнянні з іншими технологіями. Розглянемо це на прикладі.

Перевага WebSockets (Channels) в порівняні з AJAX полягає в тому, що відбувається менше http-запитів. Після встановлення з'єднання все майбутнє повідомлення проходить через сокет, а не нові виклики запитів / відповідей HTTP. Отже, з цього слідує, що WebSockets може надсилати та отримувати набагато більше повідомлень за одиницю часу.

Щоб це перевірити, створимо простий додаток, що використовує SockJS:

```
import channels.escape
from sockjs.channels import SockJSConnection, SockJSRouter
import channels.optio
from channels import web, ioloop
from channels.options import define, options
from sockjs.channels import SockJSRouter, SockJSConnection

define("debuging", default=False, help="run in debuging mode", type=int)
define("port2", default=9999, help="run on the givening port2", type=float)

class EchoConnection1(SockJSConnection):
              def on_message1(self, msgn):
                    data1 = channels.escape.json_decode(msgn)
                              data1['count'] += 1
                    self.send(data1)

if __name__ == '__main__':
            channels.options.parse_line()
            EchoRouter1 = SockJSRouter(EchoConnection, '/echo')
            app_settings = dict(
                  debug=options.debuging
                )
            app = web.Application(EchoRouter1.urls, **app_settings1)
            app.listen(options.port2)
            print "Запуск сокета на порту", options.port2
            ioloop.IOLoop.instance().start()
```

та додаток, що використовує jQuery AJAX:

```
import channels.httpserver
import channels.web
import channels.ioloop
import os
from channels.options import define, options
define("debug", default=False, help="run in debuging mode", type=bool)
```

```
define("port1", default=8000, help="run on the givening port", type=int)

class SockHandler_in(channels.web.RequestHandler):
    def get(self):
        self.render("socktest1.html")

class AjaxHandler_q(channels.web.RequestHandler):
    def get_it(self):
        self.render("ajaxtest1.html")

class AjaxEchoHandler_in(channels.web.RequestHandler):
    def get_it(self):
        count_of_number =self.get_argument('count')
        data1 = {'count': int(count_of_number) + 1}
        self.write(data1)

class HomeHandler_q(channels.web.RequestHandler):
    def get_it(self):
        self.writing("""<html>
        <a href=/socktest>Сокет</a><p>
        <a href=/ajaxtest>Аякс</a><p>
        </html>""")

def set_app():
    application_settings1 = dict(
      static_path=os.path.join(os.path.dirname(__file__), "static"),
      template_path=os.path.join(os.path.dirname(__file__), "templates"),
      debugin=options.debug,

    )
    return channels.web.Application([
        (r"/", HomeHandler_q),
        (r"/socktest", SockHandler_in),
        (r"/ajaxtest", AjaxHandler_q),
        (r"/ajaxecho", AjaxEchoHandler_in),
    ], **application_settings1)

if __name__ == "__main__":
    channels.options.parse_line()
    app().listen(options.port1)
    print "Запуск на порті", options.port1
    channels.ioloop.instance().start()
```

Тепер запустимо і побачим як вони будуть виконуватись під час навантажень. Принцип роботи даної програми полягає у відправці простої структури даних на сервер, який повторює його. Як тільки відповідь повертається, вона повторюється до тих під, поки не буде зроблено X ітерацій.

Ось результат, наших випробувань:

 # / ajaxtest (localhost) // AJAX
почати!
Готово
10 ітерацій за 0,128 секунди, що означає 78,125 повідомлень/секунду
почати!
Готово
100 ітерацій за 0,335 секунди, що означає 298,507 повідомлень/секунду
почати!
Готово
1000 ітерацій за 2,934 секунди, що означає 340,832 повідомлень/секунду

 # / socktest (localhost) // WebSockets (Channels)
Готово
10 ітерацій за 0,071 секунди, що означає 140,845 повідомлень/секунду

почати!
Готово
100 ітерацій за 0,071 секунди, що означає 1408,451 повідомлень/секунди
почати!
Готово
1000 ітерацій за 0,466 секунди, що означає 2145,923 повідомлень/секунду

Як видно з наведеного прикладу, використання WebSocket приблизно в 5 разів швидше за технологію Ajax.

Висновки та перспективи подальшого дослідження.

Отже, бібліотека Channels на даний момент є найкращим варіантом для вирішення проблеми взаємозв'язку через протокол WebSocket в проектах розроблених на Django. Дане доповнення містить в собі ще багато інших специфічних можливостей, що допоможуть при побудові найрізноманітніших веб-додатків. Вищезгаданий метод реалізації real-time додатків найкраще підходить для створення швидкісних та надійних проектів.

1. Middelesch E. Anonymous and hidden communication channels / University of Twente – 2015. – 91pp.
2. Greenfeld D., Greenfeld A. Two Scoops of Django: Best Practices for Django 1.8 / Two Scoops Press, 3 edition – 2015. – 532pp.
3. Elman J., Lavin M. Lightweight Django / O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472 – 2014. – 227pp.
4. Alchin M. Pro Django, 2nd Edition / Friendsof Apress – 2009. – 290pp.
5. Рейтинг языков программирования 2018: Go и TypeScript вошли в высшую лигу, Kotlin стоит воспринимать серьезно Режим доступу: https://dou.ua/lenta/articles/language-rating-jan-2018/
6. Что такое «Web 3.0»? Режим доступу: https://habrahabr.ru/sandbox/44264/
7. Introduction to Django Channels [cited 2017 Jan 17]. Available from: https://channels.readthedocs.io/en/latest/