netguru

Jak obsłużyć WebSockets używając Django Channels?

netguru

Jakub Samsel

Python Developer

jakub.samsel@netguru.pl

Plan prezentacji

- Kilka słów o WebSockets i tradycyjnym protokole HTTP
- Przedstawienie biblioteki Django Channels
- Przedstawienie aplikacji wykorzystującej WebSockets

netguru

1. Kilka słów o WebSockets

netguru

Websocket

- WebSocket jest protokołem opartym o TCP, zapewniający dwustronną komunikację między klientem a serwerem
- Tworzy w przeglądarce tzw *socket*, który przez adres IP i port utrzymuje obustronny kanał do serwera
- Po zestawieniu połączenia, obie strony mogą wymieniać się danymi w dowolnym momencie, wysyłając pakiet danych (tzw: *frames)*

netguru

W jaki sposób łączymy się z serwerem?

- Klient za pomocą obiektu JavaScript tworzy żądanie inicjalizujące połączenie (ang. *handshake*) jako standardowe zapytanie
- Serwer waliduje zapytanie i wyraża zgodę za pomocą kodu odpowiedzi 101 i zmienia komunikację poprzez socket TCP

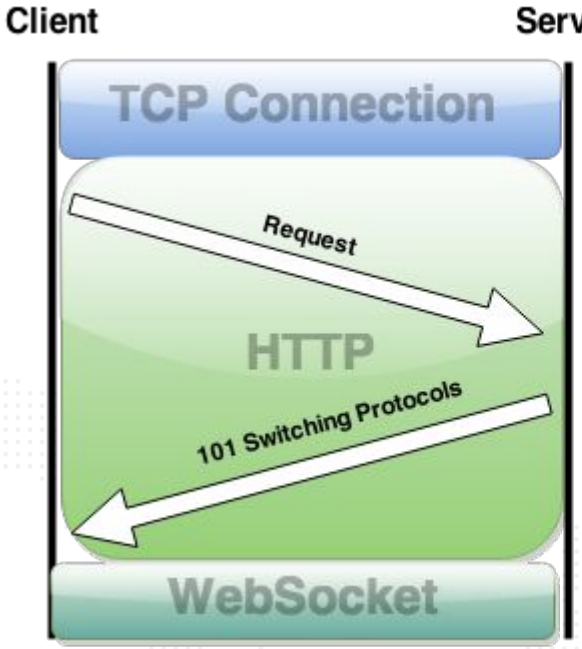
```
var chatSocket = new WebSocket(
    'ws://' + window.location.host +
    '/ws/notification/');

chatSocket.onmessage = function(e) {
};

chatSocket.onclose = function(e) {
};
```

netguru

W jaki sposób łączymy się z serwerem?




```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Version: 13
```

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xC
```

netguru

WebSocket a HTTP

WebSocket a HTTP

- W przeciwieństwie do HTTP, można wysyłać dane w dwóch kierunkach równocześnie przez jedno połączenie TCP
- WebSocket bazuje na HTTP, lecz po nawiązaniu połączenia zastępuje ten protokół
- WebSocket wykorzystuję standardowe porty jak w HTTP: 80, gdy nie są szyfrowane, i 443 dla szyfrowanych połączeń
- WebSocket ma schemat połączeń analogiczny do HTTP: *ws::* dla połączeń nieszyfrowanych i *wss::* dla połączeń szyfrowanych

netguru

2. Przedstawienie biblioteki Django Channels

Django Channels

netguru

- Django Channels(DC) jest biblioteką pozwalającą na obsługę *long-running* połączeń takich jak WebSockets
- Można tworzyć połączenia w sposób synchroniczny, asynchroniczny lub oba naraz

netguru

W jaki sposób działa Django Channels?

W jaki sposób działa Django Channels?

netguru

- Biblioteka “dzieli” połączenia na dwa komponenty: *scope* oraz seria *eventów*
- *Scope* jest zestawem informacji o nadchodzącym połączeniu
- Podczas działania *scope* do WebSocket wywoływane są serie zdarzeń(*events)*
- Podczas “życia” *scope* uruchamiana jest jedna instancja aplikacji obsługująca eventy. Aby to zrobić wykorzystywane są tzw *customers*

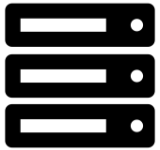
netguru

Customers

Customers

- Customer jest podstawową jednostką w bibliotece. To ona przyjmuje połączenie i zarządza eventami
- Kiedy nadchodzi żądanie połączenia od klienta na podany adres, DC stara się znaleźć ten adres w zdefiniowanych przez nas i połączyć z odpowiednim *customer*
- Następnie *customer* przejmuje zarządzanie połączeniem

Customers



http://127.0.0.1:8000/ws/notification/

```
from django.conf.urls import url

from . import consumers

websocket_urlpatterns = [
    url(r'^ws/chat/(?P<room_name>[^/]+)/$', consumers.ChatConsumer),
    url(r'^ws/notification/$', consumers.NotificationConsumer),
]
```

ws://127.0.0.1:8000/ws/notification/

101

```
class NotificationConsumer(WebsocketConsumer):
    def connect(self):
        self.accept()
```

netguru

Jak to obsłużyć za pomocą własnego Customer?

Customers

```
class NotificationConsumer(WebsocketConsumer):
    def connect(self):
        self.accept()

    def disconnect(self, close_code):
        pass


    def receive(self, text_data):
        text_data_json = json.loads(text_data)
        message = text_data_json['message']

        async_to_sync(self.channel_layer.group_send)(
            self.room_group_name,
            {
                'type': 'notification_post',
                'message': message
            }
        )

    def notification_post(self, event):
        message = event['message']

        self.send(text_data=json.dumps({
            'post': message
        }))
```

Customers

netguru

WebsocketConsumer

- connect
- disconnect
- receive

```
def receive(self, text_data):
    text_data_json = json.loads(text_data)
    message = text_data_json['message']

    async_to_sync(self.channel_layer.group_send)(
        self.room_group_name,
        {
            'type': 'notification',
            'message': message
        }
    )
```

Customers

netguru

AsyncWebsocketConsumer

- connect
- disconnect
- receive

```
async de
    self
    if n


    self
    self

    awai


    )

    awai
```

```
async def receive(self, text_data):
    text_data_json = json.loads(text_data)
    message = text_data_json['message']
    user = self.scope['user'].username
    await self.channel_layer.group_send(
        self.room_group_name,
        {
            'type': 'chat_message',
            'message': message,
            'user': user
        }
    )
```

```
st)()
d(
```

Customers

```
data = {
    "message": "Hello world"
}
```

```
def receive(self, text_data):
    text_data_json = json.loads(text_data)
    message = text_data_json['message']

    async_to_sync(self.channel_layer.group_send)(
        self.room_group_name,
        {
            'type': 'notification',
            'message': message
        }
    )
```

```
def notification(self, event):
    message = event['message']

    self.send(text_data=json.dumps({
        'message': message
    }))
```

Home Chat Messages Logout

admin: Hello world
jakubsamsel: Hello world

Send

netguru

Co to te całe *channels layers*?

Channel layers

- Channel layers służą do komunikacji między instancjami aplikacji
- Pozwala ona na transportowanie eventów pomiędzy grupami zdefiniowanymi w np *Customers*
- Django Channels zaleca korzystać z Redis jako *broker* do wysyłania eventów

Channel layers

W uproszczeniu: jeśli ktoś wysyła event to wysyłany jest przez channel layer do konsumentów którzy potrafią obsłużyć typ tego eventu

```python
def receive(self, text_data):
    text_data_json = json.loads(text_data)
    message = text_data_json['message']

    async_to_sync(self.channel_layer.group_send)(
        self.room_group_name,
        {
            'type': 'notification',
            'message': message
        }
    )
```



```python
def notification(self, event):
    message = event['message']

    self.send(text_data=json.dumps({
        'message': message
    }))
```

Channel layers - metody

Pojedynczy kanał

- send

Grupowy kanał

- group_add
- group_discard
- group_send

```
self.channel_layer.group_send(
    self.room_group_name,
    {
        'type': 'chat_message',
        'message': message,
        'user': user
    }
)
```

Channel layers - metody

netguru

Jako że Django Channels metody te działają asynchronicznie, dla konsumenta synchronicznego należy wywołać kod w sposób synchroniczny. Robi się to przy użyciu metody z biblioteki *asgiref: async_to_sync*.

```
async_to_sync(self.channel_layer.group_discard)(
    self.room_group_name,
    self.channel_name
)
```

A jak z konfiguracją tego?

A jak z konfiguracją tego?

- *settings.py*
- *routing.py*
- *myapp/routing.py*

```
INSTALLED_APPS = [
```

```
from channels.auth import AuthMiddlewareStack
from channels.routing import ProtocolTypeRouter, URLRouter
import chat.routing

application = ProtocolTypeRouter({
    # (http->django views is added by default)
    'websocket': AuthMiddlewareStack(
```

```
from django.conf.urls import url

from . import consumers

websocket_urlpatterns = [
    url(r'^ws/chat/(?P<room_name>[^/]+)/$', consumers.ChatConsumer),
    url(r'^ws/notification/$', consumers.NotificationConsumer),
]
```

```
        ),
    })
```

netguru

Jak dostać się do bazy?

Customers

- Django ORM działa w sposób synchroniczny więc jeśli chcesz uzyskać dostęp za pomocą kodu asynchronicznego potrzebujesz specjalnej obsługi by upewnić się że połączenie z bazą zostanie zakończone
- Dla *SyncConsumer* nie musimy nic robić, uruchamiany jest wtedy kod w trybie synchronicznym więc wszystkim się zajmie DC
- Dla *AsyncConsumer* istnieje specjalna metoda pozwalająca nam uruchomić wywołanie do bazy w sposób synchroniczny(*database_sync_to_async*)

Database access


```python
from channels.db import database_sync_to_async

def get_first_post():
    return Post.objects.first()

class ChatConsumer(AsyncWebsocketConsumer):
    async def connect(self):
        self.first_post = await database_sync_to_async(get_first_post)()
```

```python
@database_sync_to_async
def get_first_post():
    return Post.objects.first()

class ChatConsumer(AsyncWebsocketConsumer):
    async def connect(self):
        self.first_post = await get_first_post()
```

netguru

2. Przedstawienie własnej aplikacji wykorzystującej WebSockets

netguru

Źródła

- https://channels.readthedocs.io
- https://www.chip.pl/2013/01/protokol-websocket-internet-w-czasie-rzeczywistym/
- https://sekurak.pl/bezpieczenstwo-protokolu-websocket-w-praktyce/

netguru

Dziękuję!