Asynchronous WebSockets using Django

VilniusPy

2017-05-18

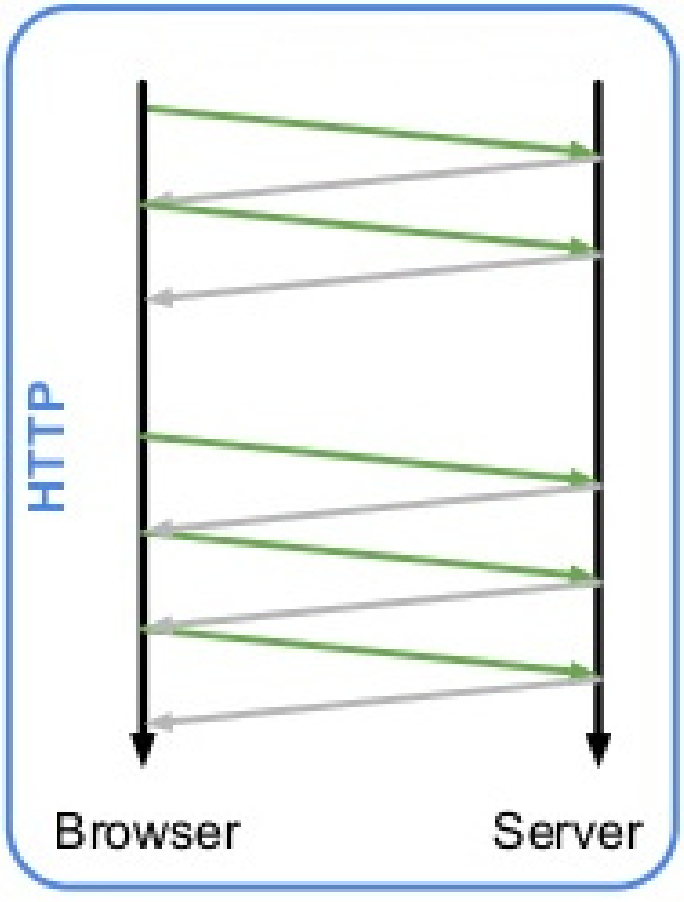
Outline of the talk

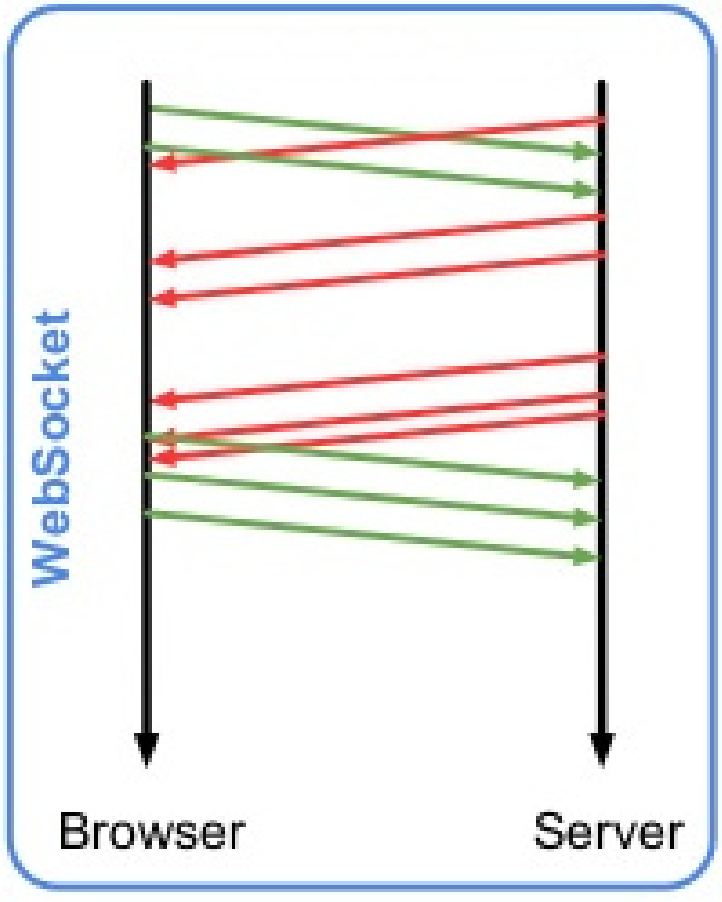
- WebSockets
  - What is Websocket?
  - Why are they important?
- Django channels
  - Introduction
  - Changes to WSGI server
- Example
  - Django code
  - Django channels code

WebSockets (a.k.a. Asynchronous WebSockets)

- WebSocket is a new computer communication protocol
- It enables a continuous bi-directional communication channel between client and server
- Transmissions are made over a single long-held TCP connection
- Messages are instantly distributed with little overhead resulting in a very low latency connection
- Communication can be asynchronous

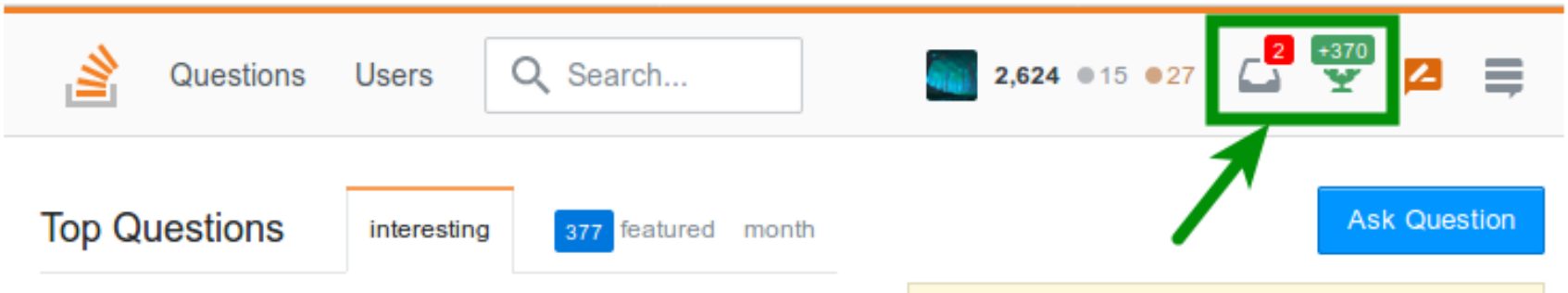
WebSockets VS HTTP





Why WebSockets are important?

Major evolution of client/server web technology
Enables true page responsiveness



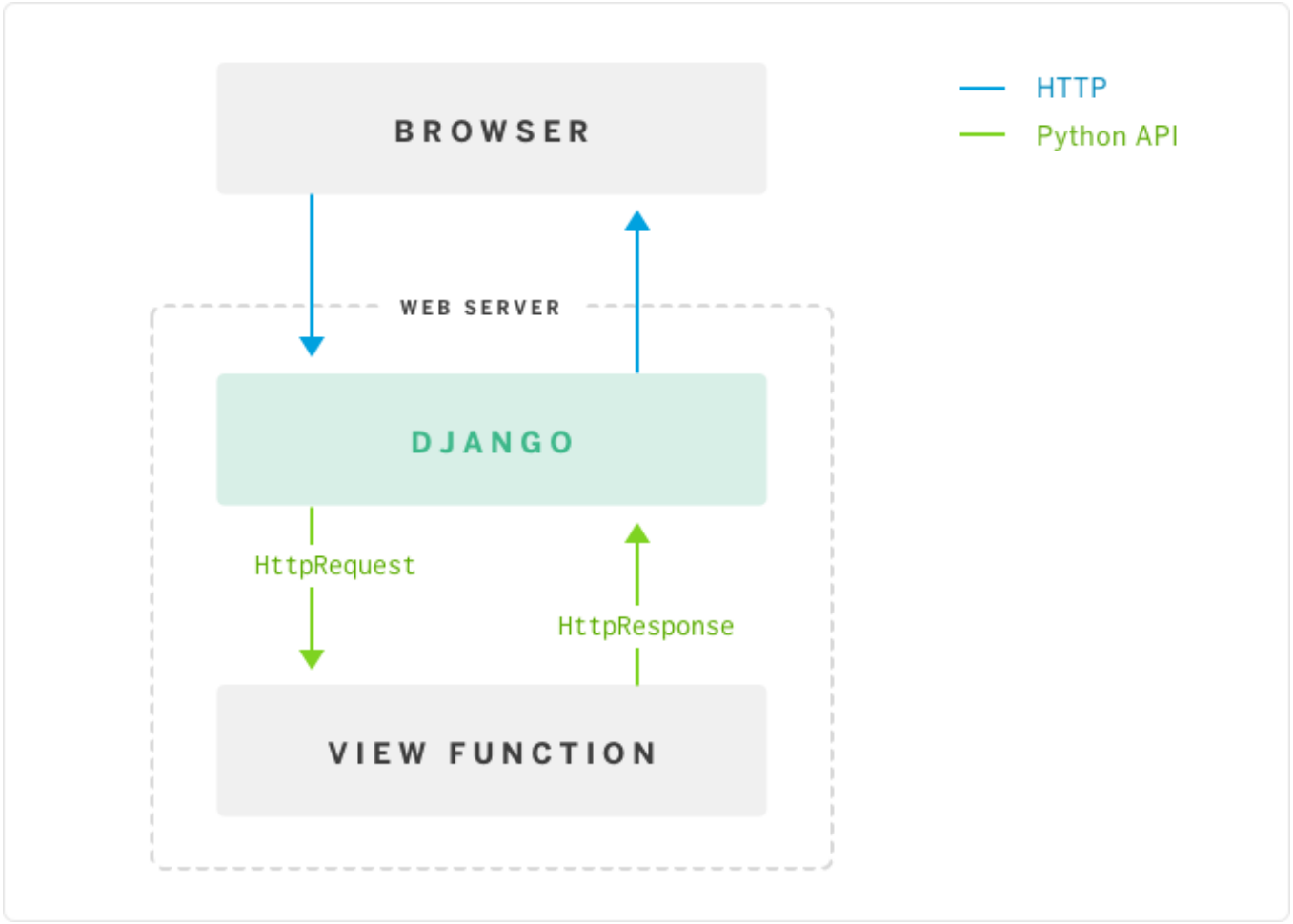
Loading...

VS

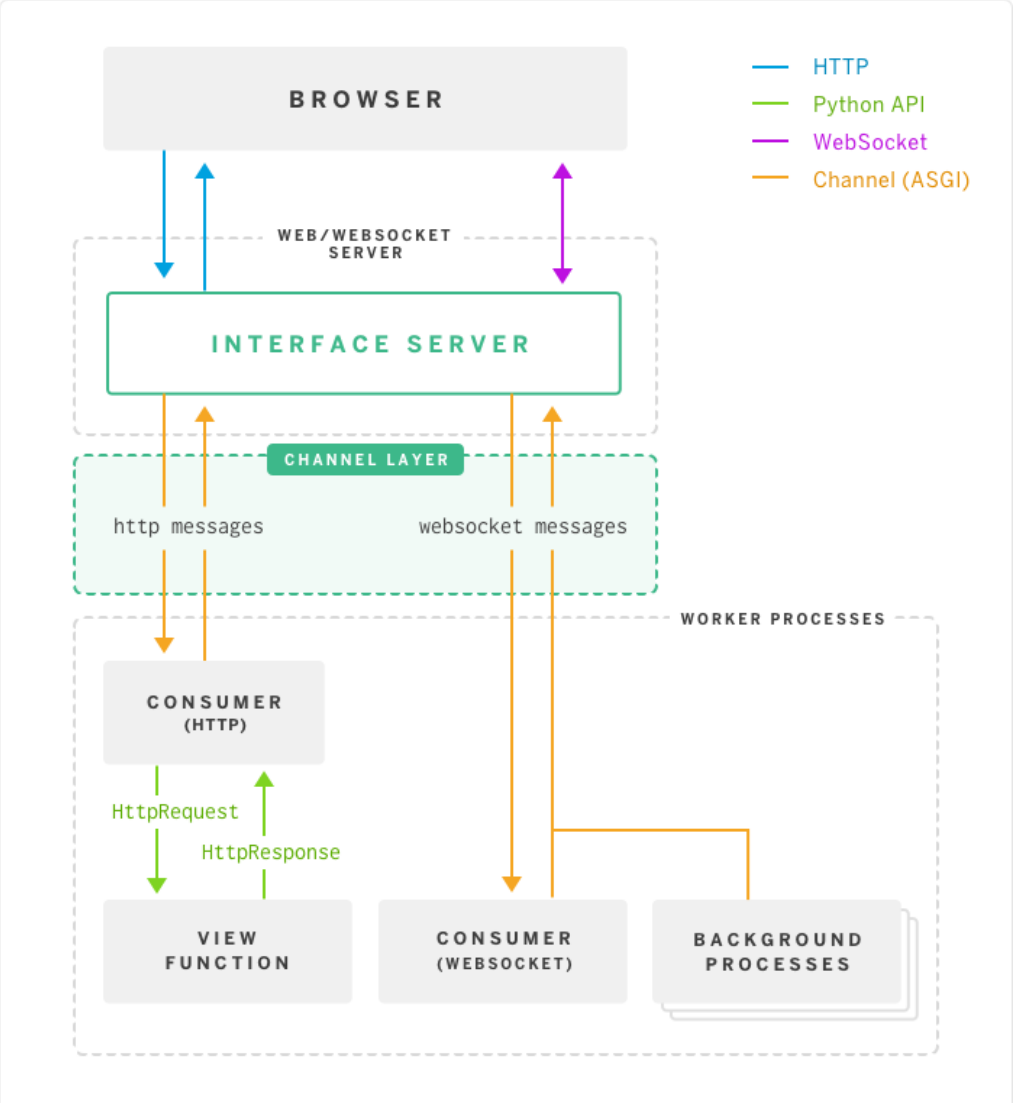
Django channels introduction

- Channels is an offical Django project which extends Django
- It enables WebSocket handling in similar way to views
- Background tasks can be run in the same server as Django

Django WSGI server



Django ASGI server



models.py

This example is taken from Jacob Kaplan-Moss blog. And his git repository.

```
class Room(models.Model):
    name = models.TextField()
    label = models.SlugField(unique=True)

class Message(models.Model):
    room = models.ForeignKey(Room, related_name='messages')
    handle = models.TextField()
    message = models.TextField()
    timestamp = models.DateTimeField(default=timezone.now,
                                     db_index=True)
```

All previous Django functionality works as before

urls.py

```
urlpatterns = [
    url(r'^$',  views.about, name='about'),
    url(r'^new/$', views.new_room, name='new_room'),
    url(r'^(?P<label>[\w-]{,50})/$', views.chat_room, name='chat_room'
]
```

views.py

```
def chat_room(request, label):
    # If the room with the given label doesn't exist,
    # automatically create it upon first visit (a la etherpad).
    room, created = Room.objects.get_or_create(label=label)

    # We want to show the last 50 messages, ordered most-recent-last
    messages = reversed(room.messages.order_by('-timestamp')[:50])

    return render(request, "chat/room.html", {
        'room': room,
        'messages': messages,
    })

def new_room(request):
    new_room = None
    while not new_room:
        with transaction.atomic():
            label = haikunator.haikunate()
            if Room.objects.filter(label=label).exists():
                continue
            new_room = Room.objects.create(label=label)
    return redirect(chat_room, label=label)
```

chat/room.html

```
<h1>{{ room.label }}</h1>
<ol id="chat">
  {% for message in messages %}
    <li>{{ message.formatted_timestamp }}
        {{ message.handle }} {{ message.message }}</li>
  {% endfor %}
</ol>

<form id="chatform">
  <p>Say something:
    <input id="handle" type="text" placeholder="Your name:">
    <input id="message" type="text" placeholder="message">
    <button type="submit" id="go">Say it</button></p>
</form>

<script type="text/javascript" src="jquery-1.12.1.min.js"></script>
<script type="text/javascript" src="chat.js"></script>
```

chat.js

```
$(function() {
    var ws_scheme = window.location.protocol == "https:" ? "wss" : "ws
    var chatsock = new WebSocket(ws_scheme + '://' +
          window.location.host + "/chat" + window.location.pathname);

    chatsock.onmessage = function(message) {
        var data = JSON.parse(message.data);
        var chat = $("#chat")
        var ele = $('<li>' + data.timestamp + ' ' + data.handle + ' '
        chat.append(ele)
    };

    $("#chatform").on("submit", function(event) {
        var message = {
            handle: $('#handle').val(),
            message: $('#message').val(),
        }
        chatsock.send(JSON.stringify(message));
        $("#message").val('').focus();
        return false;
    });
});
```

settings.py

```
$ pip install channels
$ pip install asgi_redis
```

in settings.py:

```
INSTALLED_APPS += [channels]

CHANNEL_LAYERS = {
    "default": {
        "BACKEND": "asgi_redis.RedisChannelLayer",
        "CONFIG": {
            "hosts": ['redis://localhost:6379'],
        },
        "ROUTING": "chat.routing.channel_routing",
    },
}
```

routing.py

```
from channels.routing import route
from . import consumers

channel_routing = [
    # Wire up websocket channels to our consumers:
    route('websocket.connect', consumers.ws_connect),
    route('websocket.receive', consumers.ws_receive, path='^*'),
    route('websocket.disconnect', consumers.ws_disconnect),
]
```

consumers.py

```
from channels import Group
from channels.sessions import channel_session
from .models import Room

@channel_session
def ws_connect(message):
    prefix, label = message['path'].strip('/').split('/')
    room = Room.objects.get(label=label)
    Group('chat-' + label).add(message.reply_channel)
    message.channel_session['room'] = room.label

@channel_session
def ws_receive(message):
    label = message.channel_session['room']
    room = Room.objects.get(label=label)
    data = json.loads(message['text'])
    m = room.messages.create(handle=data['handle'], message=data['mess
    Group('chat-'+label).send({'text': json.dumps(m.as_dict())})

@channel_session
def ws_disconnect(message):
    label = message.channel_session['room']
    Group('chat-'+label).discard(message.reply_channel)
```

asgi.py

```
import os
import channels.asgi

os.environ.setdefault("DJANGO_SETTINGS_MODULE", "chat.settings")
channel_layer = channels.asgi.get_channel_layer()
```

In the future, Django will probably auto-generate this file, like it currently does for wsgi.py

Running the code

Install redis-server and run it:

```
$ sudo apt-get update
$ sudo apt-get install redis-server
$ redis-server
```

Run the project:

```
$ python manage.py migrate
$ python manage.py runserver
```

Data binding (Inbounding)

Data binding framework automates the process of tying Django models into frontend view (from channels docs)

```
from django.db import models
from channels.binding.websockets import WebsocketBinding

class IntegerValue(models.Model):
    name = models.CharField(max_length=100, unique=True)
    value = models.IntegerField(default=0)

class IntegerValueBinding(WebsocketBinding):
    model = IntegerValue
    stream = "intval"
    fields = ["name", "value"]

    @classmethod
    def group_names(cls, instance):
        return ["intval-updates"]

    def has_permission(self, user, action, pk):
        return True
```

Data binding (Outbounding)

```
from channels.generic.websockets import WebsocketDemultiplexer
from .binding import IntegerValueBinding

class Demultiplexer(WebsocketDemultiplexer):

    consumers = {
        "intval": IntegerValueBinding.consumer,
    }

    def connection_groups(self):
        return ["intval-updates"]
```

Thank you!

VilniusPy