Бэкенд разработка на python

Лекция 10

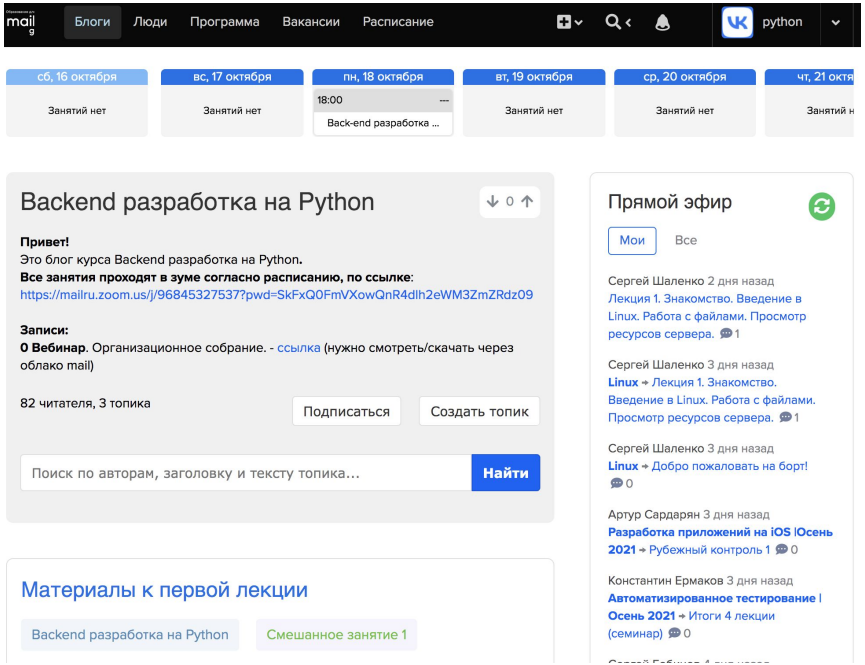
Очереди и задачи, real time сообщения

Кандауров Геннадий

образование

Напоминание отметиться на портале

+ оставить отзыв после лекции



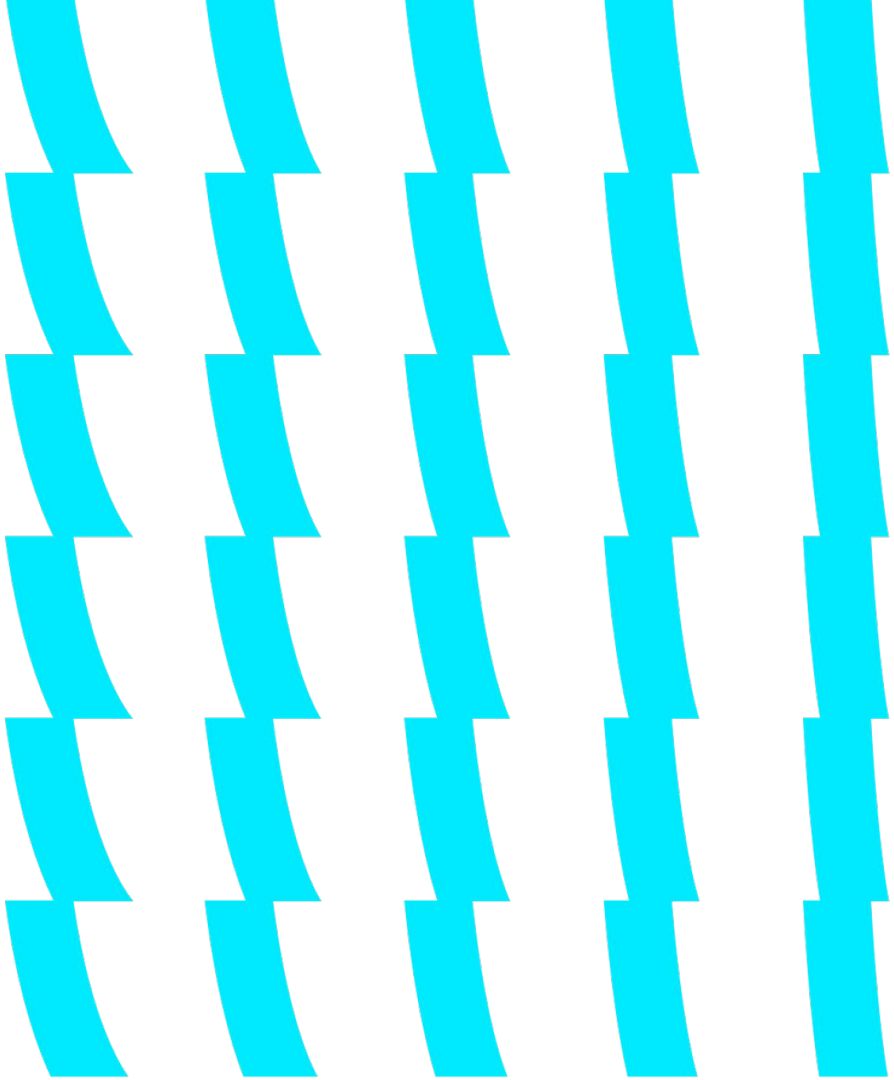
Квиз по прошлой лекции

Содержание занятия

- Очереди и задачи
- Celery, cron
- Real time сообщения
- Web сокеты

Очереди и задачи

Сделать что-то асинхронно?



Фоновые задачи

- отправка уведомлений (email, sms, push, desktop)
- периодическое обновление данных
- генерация отчетов

Celery

Celery - распределённая очередь заданий, реализованная на языке Python, служит для хранения отложенных задач

Преимущества:

- выполнение некоторого кода в фоновом режиме
- возможность ускорения времени ответа сервера

```
pip install celery
```

Celery: основные понятия

Брокер (broker) - служит для передачи сообщений (задач) между так называемыми исполнителями (workers).
Celery общается с брокером по протоколу AMQP.

- Redis (производительность с celery выше)
- RabbitMQ

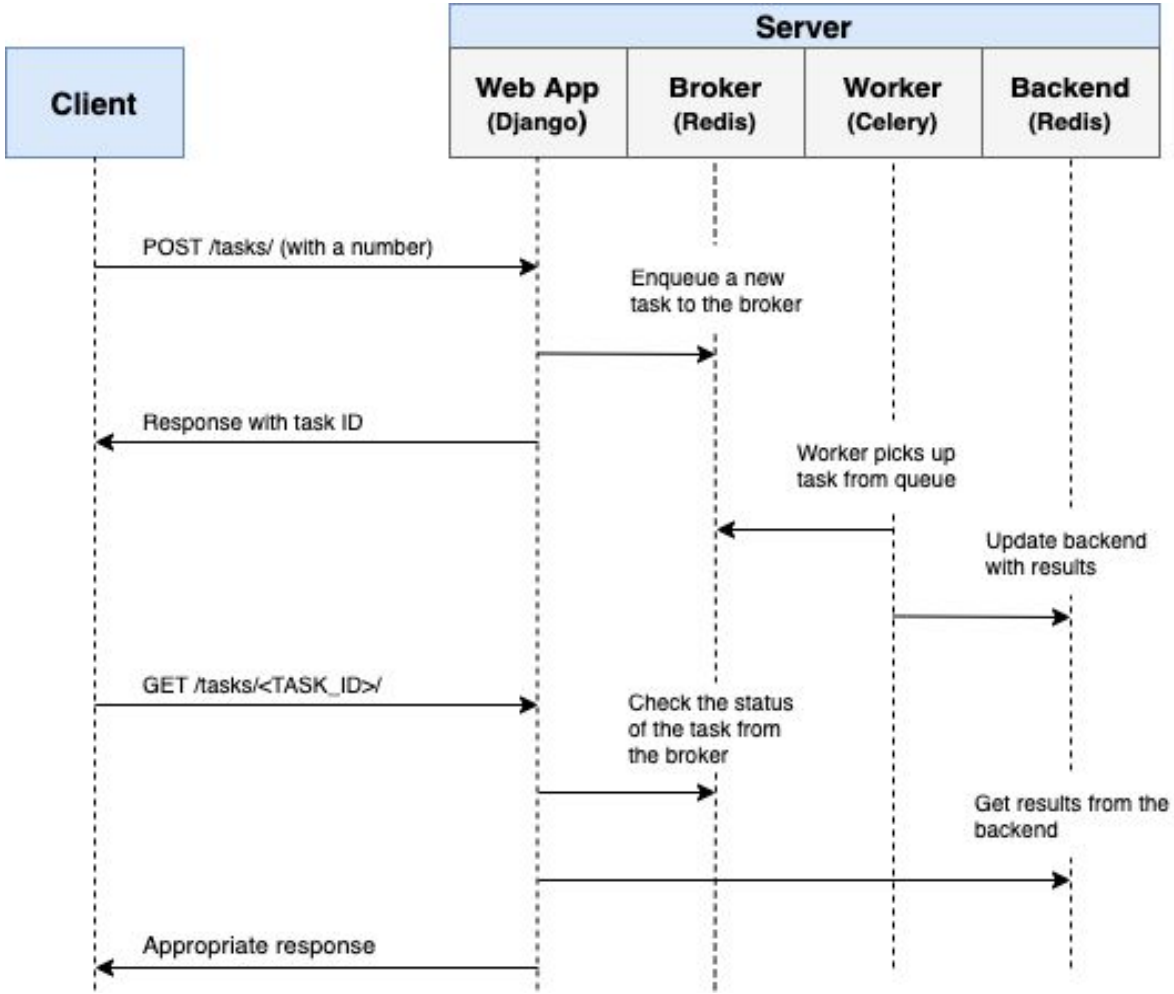
Бэкенд (backend) - хранилище результатов выполнения задач

- Memcached
- СУБД

Поднимаем redis

```
redis-server /usr/local/etc/redis.conf
```

Celery


Server
Client
Web App (Django)
Broker (Redis)
Worker (Celery)
Backend (Redis)
POST /tasks/ (with a number)
Enqueue a new task to the broker
Response with task ID
Worker picks up task from queue
Update backend with results
GET /tasks/<TASK_ID>/
Check the status of the task from the broker
Get results from the backend
Appropriate response

Celery: конфигурирование

https://docs.celeryproject.org/en/latest/django/first-steps-with-django.html

в качестве бэкенда используем **Redis**

Celery: первый таск

```
from application import app

@app.task()
def add_together(a, b):
    return a + b

# запускаем celery, отправляем таск в очередь
celery -A application worker

>>> from tasks import add_together
>>> add_together.delay(23, 42)

#loglevel
-l, --loglevel
DEBUG, INFO, WARNING, ERROR, CRITICAL, FATAL
```

Celery: разделение по очередям

Celery: очереди с приоритетом

```
app.conf.task_routes = {
    'feed.tasks.import_feed': {'queue': 'feeds'}
}

app.conf.task_routes = {'feed.tasks.*': {'queue': 'feeds'}}

app.conf.task_routes = ([
    ('feed.tasks.*', {'queue': 'feeds'}),
    ('web.tasks.*', {'queue': 'web'}),
    (re.compile(r'(video|image)\.tasks\..*'), {'queue': 'media'}),
])
```

Celery: пишите короткие таски

```
from utils import generate_report, send_email

@app.task()
def send_report():
    filename = generate_report()
    send_email(subject, message, attachments=[filename])
```

Celery: таймауты

Установка таймаутов на время выполнения:

- Через декоратор **@app.task()**, передавая **soft_time_limit**, **time_limit**
- Установив глобальный timelimit для всех тасков в очереди

Celery: chain

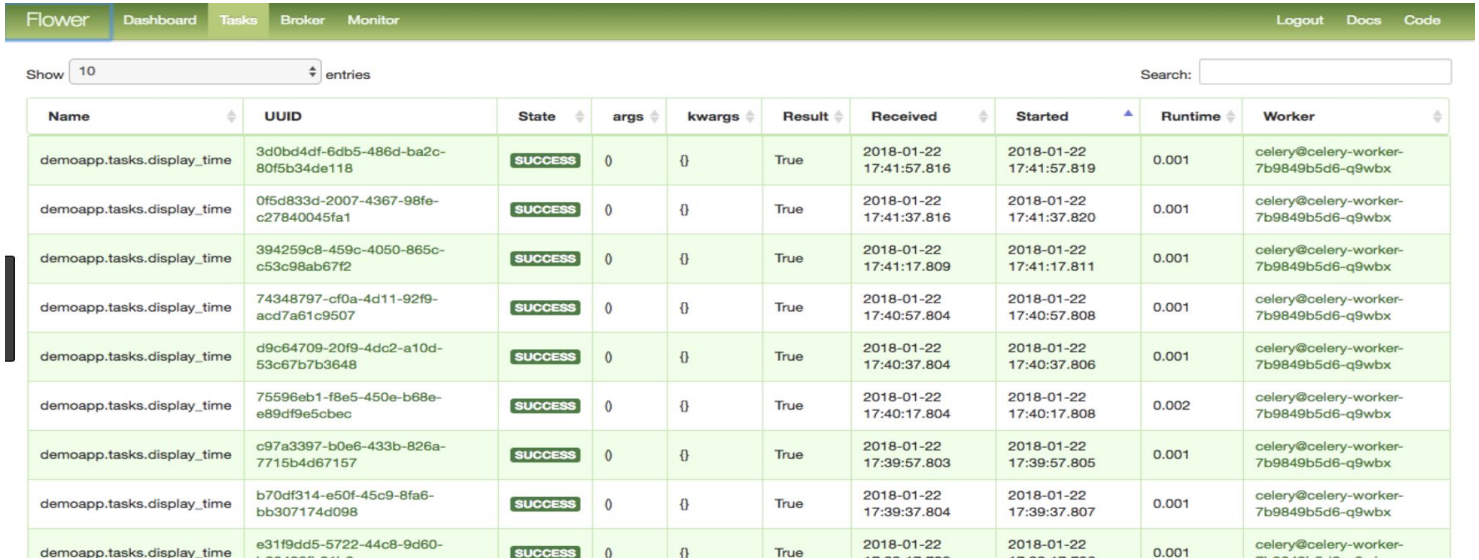
```
from celery import chain

@shared_task
def add(a, b):
    return a + b

result = chain(add.s(2, 2), add.s(4), add.s(8))()
result.get()
```

Celery: мониторинг выполнения тасков

```
pip install flower
celery -A application flower --port=5555
http://localhost:5555
```



Name	UUID	State	args	kwargs	Result	Received	Started	Runtime	Worker
demoapp.tasks.display_time	3d0bd4df-6db5-486d-ba2c-80f5b34de118	SUCCESS	()	{}	True	2018-01-22 17:41:57.816	2018-01-22 17:41:57.819	0.001	celery@celery-worker-7b9849b5d6-q9wbx
demoapp.tasks.display_time	0f5d833d-2007-4367-98fe-c27840045fa1	SUCCESS	()	{}	True	2018-01-22 17:41:37.816	2018-01-22 17:41:37.820	0.001	celery@celery-worker-7b9849b5d6-q9wbx
demoapp.tasks.display_time	394259c8-459c-4050-865c-c53c98ab67f2	SUCCESS	()	{}	True	2018-01-22 17:41:17.809	2018-01-22 17:41:17.811	0.001	celery@celery-worker-7b9849b5d6-q9wbx
demoapp.tasks.display_time	74348797-cf0a-4d11-92f9-acd7a61c9507	SUCCESS	()	{}	True	2018-01-22 17:40:57.804	2018-01-22 17:40:57.808	0.001	celery@celery-worker-7b9849b5d6-q9wbx
demoapp.tasks.display_time	d9c64709-20f9-4dc2-a10d-53c67b7b3648	SUCCESS	()	{}	True	2018-01-22 17:40:37.804	2018-01-22 17:40:37.806	0.001	celery@celery-worker-7b9849b5d6-q9wbx
demoapp.tasks.display_time	75596eb1-f8e5-450e-b68e-e89df9e5cbec	SUCCESS	()	{}	True	2018-01-22 17:40:17.804	2018-01-22 17:40:17.808	0.002	celery@celery-worker-7b9849b5d6-q9wbx
demoapp.tasks.display_time	c97a3397-b0e6-433b-826a-7715b4d67157	SUCCESS	()	{}	True	2018-01-22 17:39:57.803	2018-01-22 17:39:57.805	0.001	celery@celery-worker-7b9849b5d6-q9wbx
demoapp.tasks.display_time	b70df314-e50f-45c9-8fa6-bb307174d098	SUCCESS	()	{}	True	2018-01-22 17:39:37.804	2018-01-22 17:39:37.807	0.001	celery@celery-worker-7b9849b5d6-q9wbx
demoapp.tasks.display_time	e31f9dd5-5722-44c8-9d60-	SUCCESS	()	{}	True	2018-01-	2018-01-	0.001	celery@celery-worker-

Celery: beat

Особый воркер, которые умеет ставить задачи по расписанию

Типы расписаний:

- timedelta - временной интервал
- crontab - настраиваемое расписание
- solar - солнечные циклы

Запуск:

```
celery beat -A application
```

Celery: beat timedelta

```
celery.conf.beat_schedule = {
    'add-every-30-seconds': {
        'task': 'tasks.add',
        'schedule': 30.0,
        'args': (16, 16)
    },
}

celery.conf.timezone = 'UTC'
```

Celery: beat crontab

```
celery.conf.beat_schedule = {
    # Executes every Monday morning at 7:30 a.m.
    'add-every-monday-morning': {
        'task': 'tasks.add',
        'schedule': crontab(hour=7, minute=30, day_of_week=1),
        'args': (16, 16),
    },
}
```

Celery: beat solar

```
celery.conf.beat_schedule = {
    # Executes at sunset in Melbourne
    'add-at-melbourne-sunset': {
        'task': 'tasks.add',
        'schedule': solar('sunset', -37.81753, 144.96715),
        'args': (16, 16),
    },
}

# возможные параметры: sunrise, sunset, dawn or dusk
# аргументы: solar(event, latitude, longitude)
```

Celery: пример email

https://docs.djangoproject.com/en/3.2/topics/email/

```
# email server config
EMAIL_HOST = 'smtp.googlemail.com'
EMAIL_PORT = 465
EMAIL_USE_TLS = False
EMAIL_USE_SSL = True
EMAIL_HOST_USER = 'your-gmail-username'
EMAIL_HOST_PASSWORD = 'your-gmail-password'

# administrator list
ADMINS = ['your-gmail-username@gmail.com']
```

Celery: пример email

```
from django.core.mail import EmailMessage

msg = EmailMessage(
    "Hello",
    "testing body",
    "some-adm@gmail.com",
    ["some-adm@yandex.ru"],
)

msg.send()
```

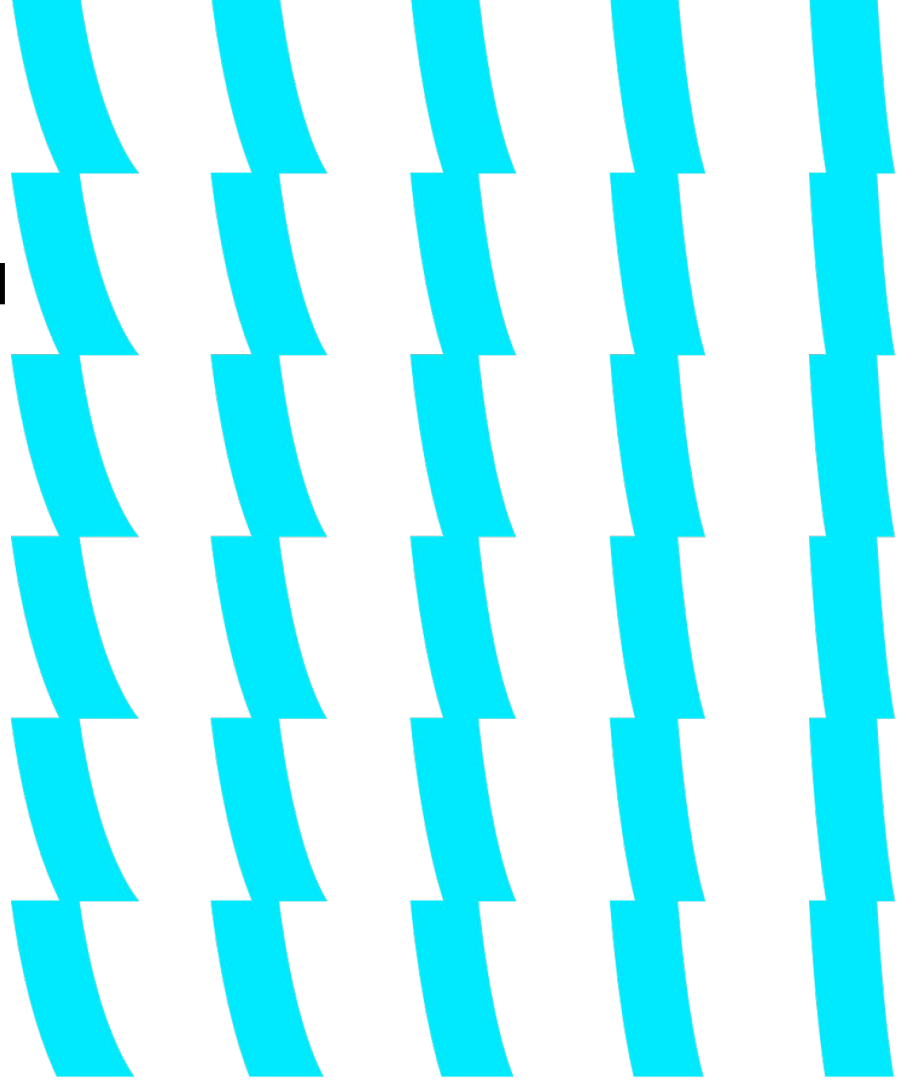
Celery: пример email

```
from django.core.mail import EmailMessage
from application.celery_app import app

@app.task
def send_email(subject, sender, recipients, text):
    message = EmailMessage(
        subject,
        text,
        sender=sender,
        recipients=recipients,
    )
    message.send()

# либо воспользуйтесь готовой функцией
# from django.core.mail import send_mail
```

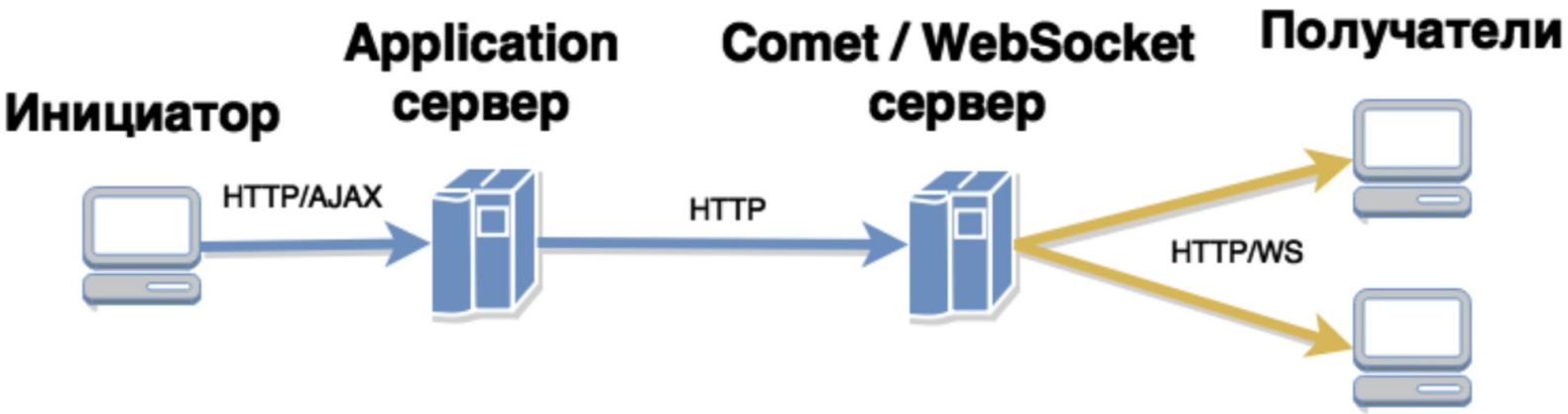
Real time сообщения



Примеры использования

- Чаты и мессенджеры
- Отображение котировок
- Прямые трансляции (a-la twitter)
- Push уведомления
- Сетевой обмен в играх на HTML

Архитектура



Решения

- `Polling` - периодический опрос сервера
- `Comet` (`Long polling`) - polling с долгоживущими запросами
- `Server Push` - бесконечный запрос
- `WebSocket` - специализированный протокол

Polling - периодический опрос

Polling на клиенте

```
var since = 0;
setInterval(function() {
    $.ajax({
        type: 'GET',
        url:  '/messages/',
        data: { channel_id: 5, since: since },
    }).success(function(resp) {
        if (!resp.messages || !resp.messages.length) {
            return;
        }
        handleMessages(resp.messages);
        since = resp.messages[0].id;
    });
}, 5000);
```

Polling на сервере

```
def messages(request):
    channel_id = request.GET.get('channel_id')
    since = request.GET.get('since', 0)
    messages = Messages.filter(
        channel_id=channel_id, id__gt=since
    ).order_by('-id')

    messages = [m.as_data() for m in messages]

    return HttpResponseAjax(messages=messages)
```

Polling: pros and cons

➕ Простота и надежность реализации

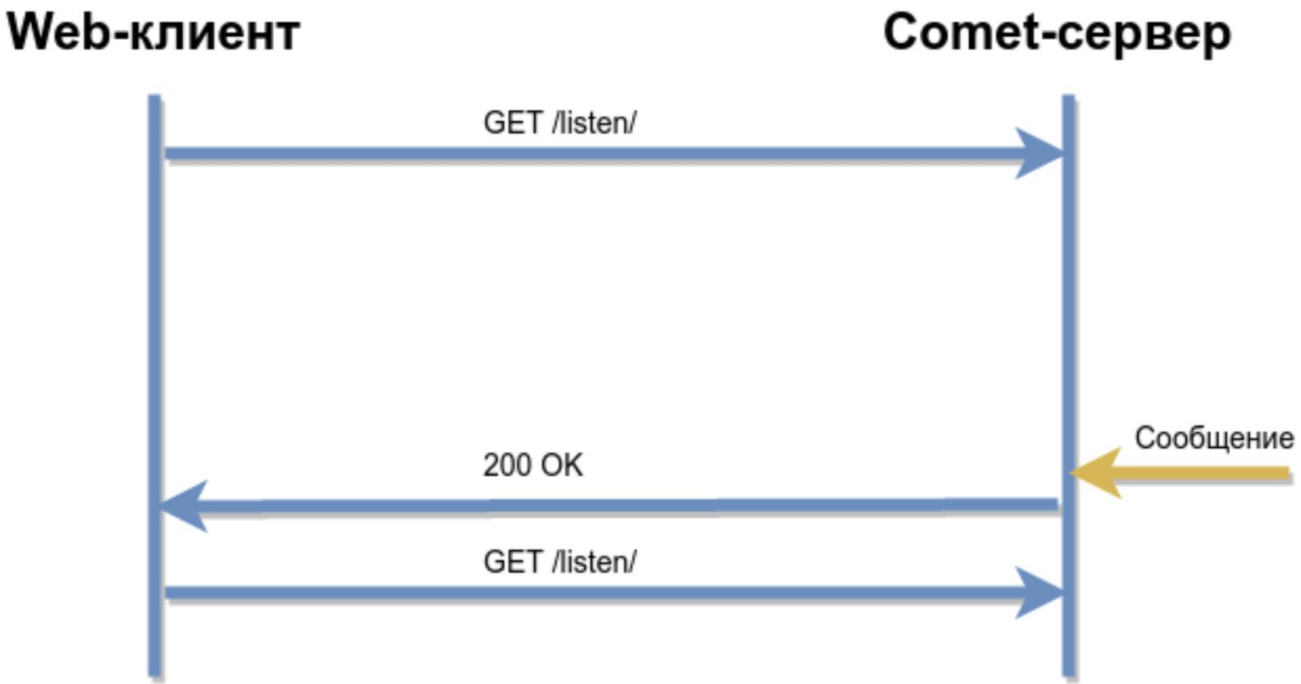
➕ Не требуется дополнительного ПО

➖ Сообщения приходят с задержкой до N секунд

➖ Избыточное число HTTP запросов RPS=CCU/N

➖ Ограничение по числу пользователей

Comet - долгоживущие запросы



Comet на клиенте

```
function getComet() {
    $.ajax({
        type: 'GET',
        url:  '/listen/',
        data: { channel_id: 5 },
    }).success(function(resp) {
        handleMessages(resp.messages);
        getComet();
    }).error(function() {
        setTimeout(getComet, 10000);
    });
}
getComet();
```

Comet на сервере

В технологии comet сервер должен поддерживать одновременно открытыми большое количество соединений, причем каждое соединение находится в ожидании сообщений для него. По этой причине мы не можем использовать классический application- сервер в роли comet-сервера. Для comet-сервера необходима отдельная технология, например:

nginx + push-stream-module

https://github.com/wandenberg/nginx-push-stream-module

Comet: nginx + push-stream-module

```
location /publish/ {
    push_stream_publisher normal; # включаем отправку
    push_stream_channels_path $arg_cid; # id канала
    push_stream_store_messages off; # не храним сообщения
    allow 127.0.0.1;
    deny all;
}

location /listen/ {
     push_stream_subscriber long-polling; # включаем доставку
     push_stream_channels_path  $arg_cid; # id канала
     default_type application/json; # MIME тип сообщения
}
```

Comet на сервере (отправка сообщения)

```
def send_message(request):
    cid = request.POST.get('cid')
    text = request.POST.get('text')
    url = f'http://127.0.0.1/publish?cid={cid}'
    data = {'messages': [text]}
    resp = requests.post(url, body=json.dumps(data)) # мб долгим

    if resp.status_code == 200:
        return HttpResponseAjax()
    else:
        return HttpResponseAjaxError(code=resp.status_code)
```

Comet: pros and cons

➕ Поддержка всеми браузерами

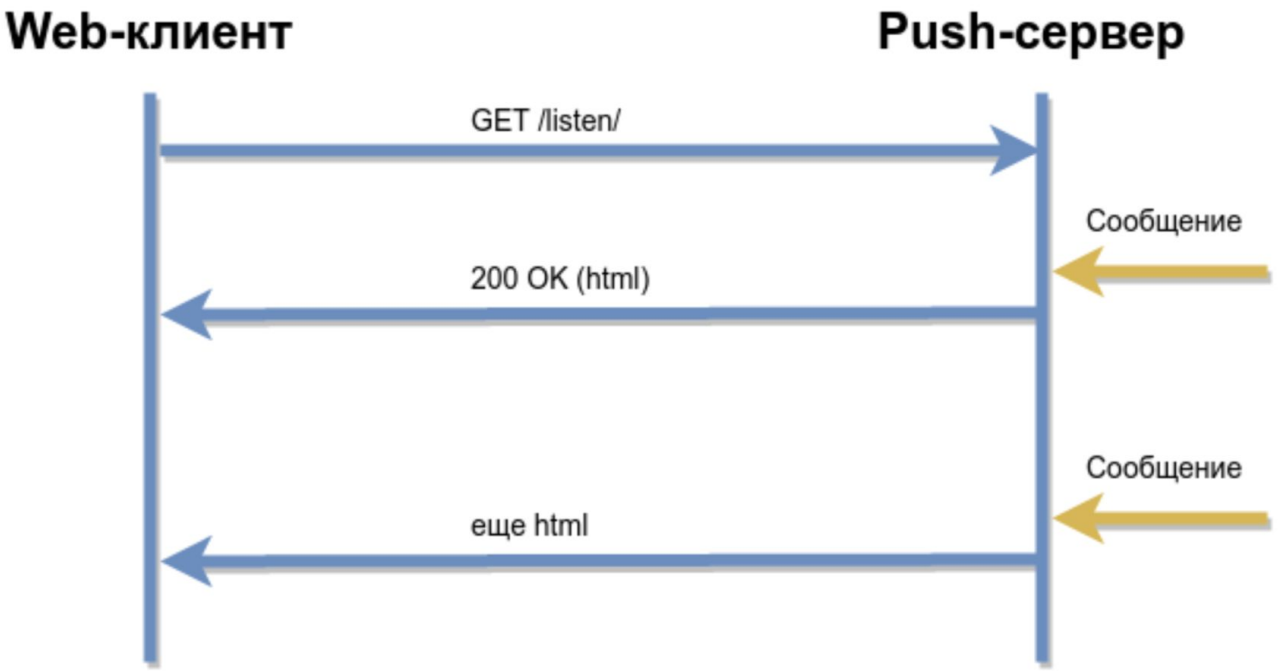
➕ Поддержка большого числа пользователей

➕ Относительная простота реализации

➖ Избыточные HTTP запросы

➖ Half-duplex

Server push - бесконечный запрос



Server push на клиенте

```
<script>
    function handle(message) {
        // любая логика
    }
</script>
<iframe src='/listen/?cid=123'></iframe>


# Ответ сервера:
<script>parent.handle({ message: 'hello' })</script>
```

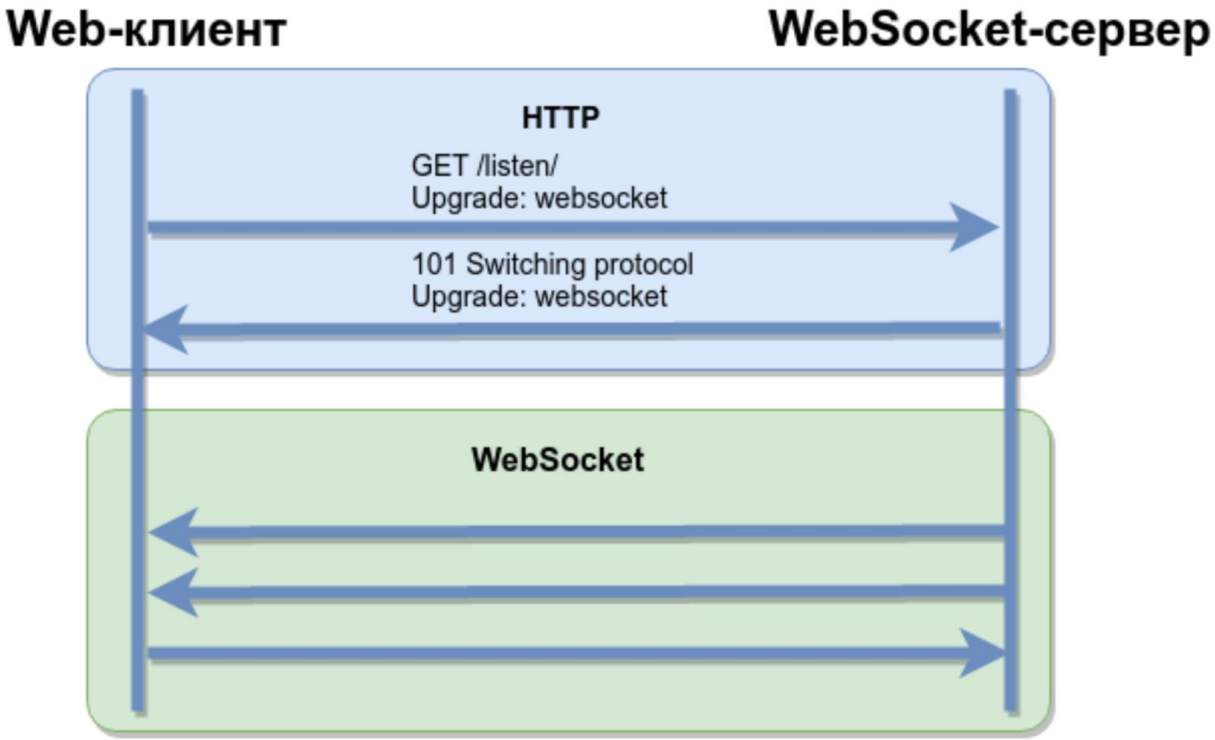
Server push: pros and cons

➕ Поддержка большого числа пользователей

➕ Относительная простота реализации

➖ Поддержка не всеми браузерами

WebSocket


Web-клиент
WebSocket-сервер
HTTP
GET /listen/
Upgrade: websocket
101 Switching protocol
Upgrade: websocket
WebSocket

WebSocket handshake

```
GET /listen HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13

HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

WebSocket на клиенте

```
var socket = new WebSocket('ws://host/echo');  # также wss

socket.onopen = function(event) {
    console.log('ws opened');
    var data = JSON.stringify({ message: "Hello WebSocket" });
    socket.send(data);
};
socket.onmessage = function(event) {
    var resp = JSON.parse(event.data);
    console.log('ws message', resp.message);
};
socket.onclose = function(event) {
    console.log('ws closed')
};
```

WebSocket на сервере

```
class EchoWebSocket(tornado.websocket.WebSocketHandler):

    def open(self):
        print("WebSocket opened")

    def on_message(self, message):
        self.write_message(message)

    def on_close(self):
        print("WebSocket closed")
```

Django: channels https://channels.readthedocs.io/

WebSocket: pros and cons

➕ Минимальный объем трафика

➕ Минимальная нагрузка на сервер

➕ Поддержка большого числа пользователей

➕ Full-duplex

➖ Нет поддержки IE<10, OperaMini, Android<4.4

➖ Требуется специальный WebSocket-сервер

➖ Плохо работает с прокси-серверами

Софт для Real Time сообщений

Real Time Web Technologies Guide -

https://www.leggetter.co.uk/real-time-web-technologies-guide/

Centrifugo - *https://github.com/centrifugal/centrifugo*

Centrifugo

1. Устанавливаем
   https://github.com/centrifugal/centrifugo/blob/master/docs/content/server/install.md

2. Генерируем конфиг
   `centrifugo genconfig`

3. В настройках бекенда регистрируем Centrifugo secret and Centrifugo API key
   (дефолтный адрес `http://localhost:8000/api`)

4. Подключаем библиотеку для клиента
   https://github.com/centrifugal/centrifuge-js

5. Подписываем клиенты на каналы

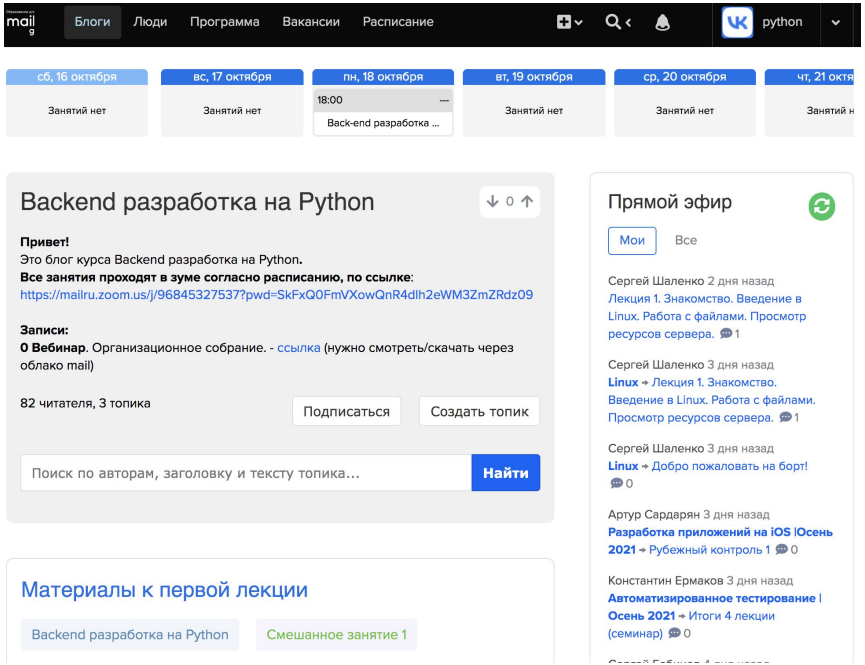
Домашнее задание по лекции #10

1. Написать таск, который отправляет email админу при создании объекта в бд
2. Написать периодический таск на к.-л. действие, например, считать количество пользователей каждые 5 мин и выводить
3. Использовать flower для мониторинга задач
4. Установить и поднять centrifugo, прикрутить к проекту, показать отправку какого-то сообщения с помощью websocket

* При создании объекта в бд, отображать новый объект в списке объектов

Напоминание отметиться на портале Vol 2

+ оставить отзыв



Спасибо за внимание

образование