

Web Application Development

© Alexander Menshchikov, ITMO 2023



Backend

Web Application Architecture



Web browser



Web server



Web application

Architecture. Step 1



HTTP request
wad.itmo.xyz/

/ is routed
to an Application

Architecture. Step 2

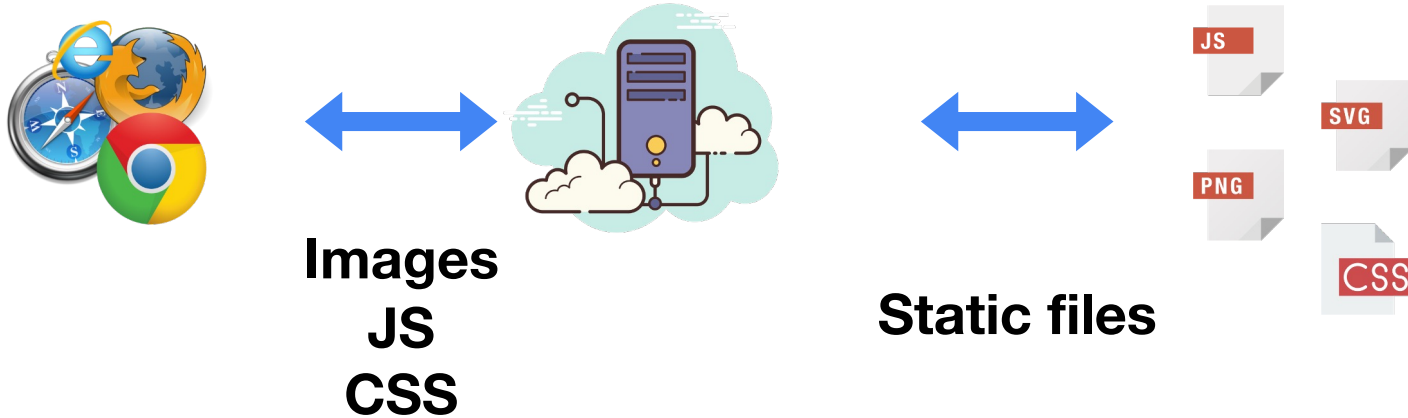


**Send HTML
back to client**

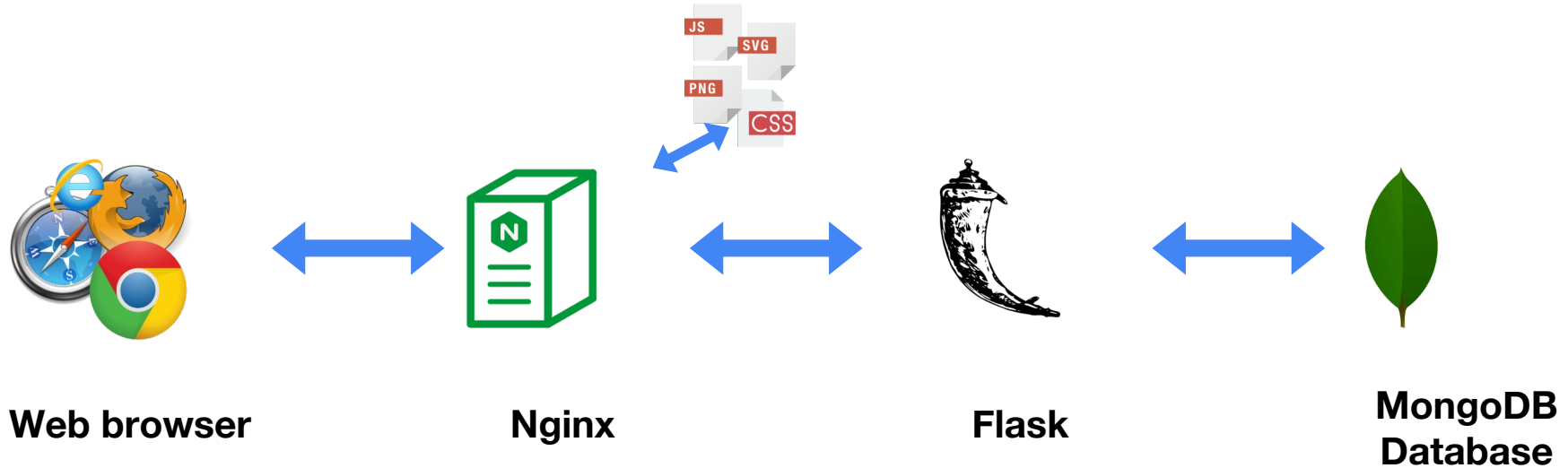


Render HTML

Architecture. Step 3



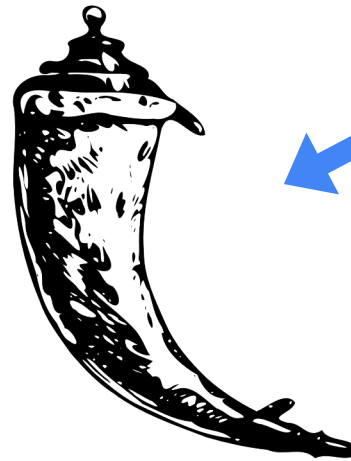
Web Application Architecture



Web Application Architecture



Web browser



Flask



HTTP

HTTP Request

curl http://wad.itmo.xyz -vvv

Method

```
* Trying 185.199.108.153...
* TCP_NODELAY set
* Connected to wad.itmo.xyz (185.199.108.153) port 80 (#0)
> GET / HTTP/1.1
> Host: wad.itmo.xyz
> User-Agent: curl/7.64.1
> Accept: */*
>
```

```
97 db 47 45 54 20 2f 20 48 54 54 50 2f 31 2e 31
0d 0a 48 6f 73 74 3a 20 77 61 64 2e 69 74 6d 6f
2e 78 79 7a 0d 0a 55 73 65 72 2d 41 67 65 6e 74
3a 20 63 75 72 6c 2f 37 2e 36 34 2e 31 0d 0a 41
63 63 65 70 74 3a 20 2a 2f 2a 0d 0a 0d 0a
```

```
..GET / HTTP/1.1
..Host: wad.itmo
.xyz..User-Agent
: curl/7.64.1..A
ccept: */*....
```

HTTP Response

curl http://wad.itmo.xyz -vvv

 **Status**

```
< HTTP/1.1 301 Moved Permanently
< Server: GitHub.com
< Content-Type: text/html
< Location: https://wad.itmo.xyz/
< Content-Length: 162
< Date: Thu, 02 Apr 2020 11:30:02 GMT
<
<html>
<head><title>301 Moved Permanently</title></head>
<body>
<center><h1>301 Moved Permanently</h1></center>
<hr><center>nginx</center>
</body>
</html>
* Connection #0 to host wad.itmo.xyz left intact
```

URI

Diagram illustrating the components of a URI (Uniform Resource Identifier) using the example: `http://john.doe:password@www.example.com:123/forum/questions/?tag=networking&order=newest#top`

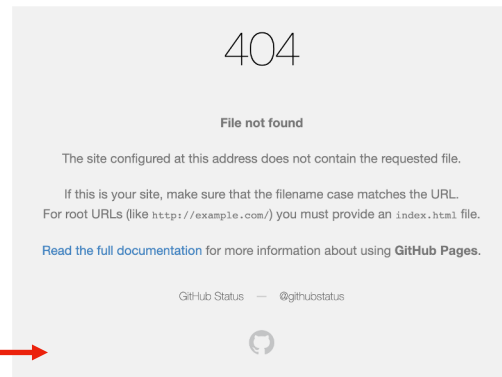
The components are labeled as follows:

- scheme**: `http`
- authority**: `john.doe:password@www.example.com:123` (further divided into **userinfo**: `john.doe:password`, **host**: `www.example.com`, and **port**: `123`)
- path**: `/forum/questions/`
- query**: `?tag=networking&order=newest`
- fragment**: `#top`

<https://wad.itmo.xyz/index.html>

<https://wad.itmo.xyz/>

<https://wad.itmo.xyz/qwerty> →



HTTP Status codes

- 1xx: Informational
 - 2xx: Success
 - 3xx: Redirection
 - 4xx: Client Error
 - 5xx: Server Error
- 200 OK
 - 301 Moved Permanently
 - 400 Bad Request
 - 401 Unauthorized
 - 403 Forbidden
 - 404 Not Found
 - 500 Internal Server Error
 - 502 Bad Gateway
 - 504 Gateway Timeout.

HTTP Headers

https://en.wikipedia.org/wiki/List_of_HTTP_header_fields

Request

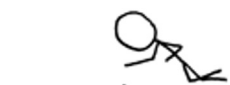
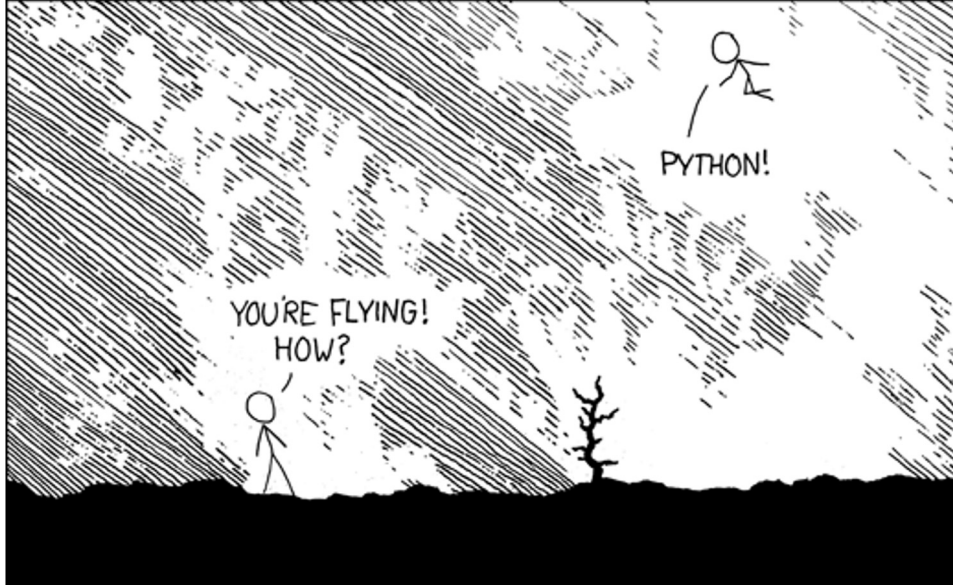
- Authorization
- Content-Type
- Cookie
- Host

Response

- Location
- Server
- Set-Cookie

Demo

Python Flask



I LEARNED IT LAST
NIGHT! EVERYTHING
IS SO SIMPLE!

HELLO WORLD IS JUST
`print "Hello, world!"`

I DUNNO...
DYNAMIC TYPING?
WHITESPACE?

COME JOIN US!
PROGRAMMING
IS FUN AGAIN!
IT'S A WHOLE
NEW WORLD
UP HERE!



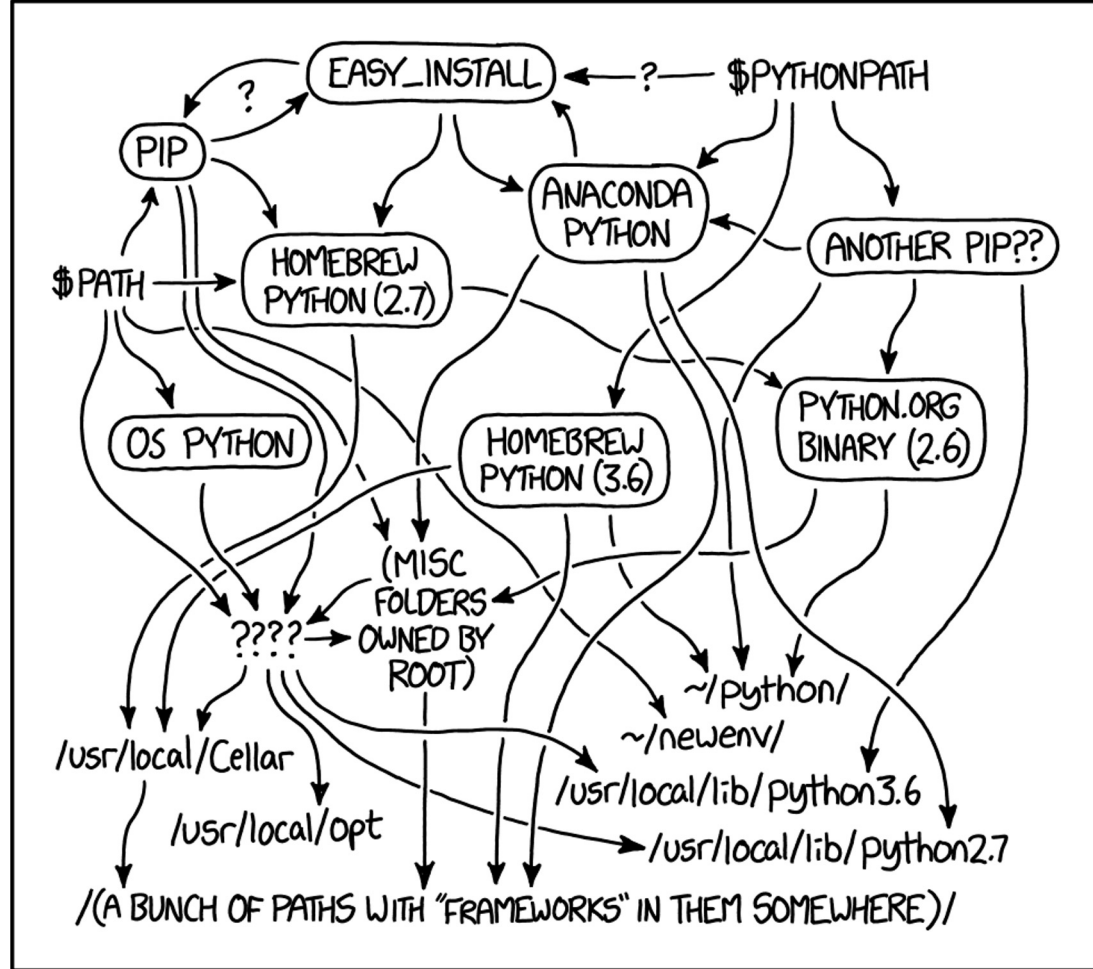
BUT HOW ARE
YOU FLYING?

I JUST TYPED
`import antigravity`
THAT'S IT?



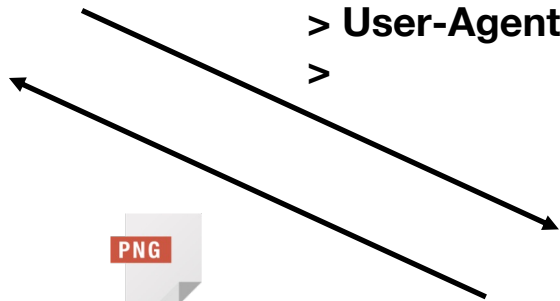
... I ALSO SAMPLED
EVERYTHING IN THE
MEDICINE CABINET
FOR COMPARISON.

BUT I THINK THIS
IS THE PYTHON.



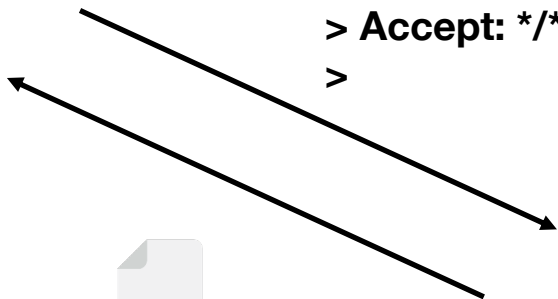


```
> GET /img/apple.png HTTP/1.1  
> Host: localhost  
> User-Agent: curl/7.64.1  
>
```





```
> GET / HTTP/1.1  
> Host: itmo.xyz  
> User-Agent: curl/7.64.1  
> Accept: */*  
>
```



```
< HTTP/1.1 200 OK  
< Server: Werkzeug/1.0.0 Python/3.7.1  
< Date: Thu, 02 Apr 2020 13:26:37 GMT  
< Content-Type: text/html; charset=UTF-8  
< Transfer-Encoding: chunked  
< Connection: keep-alive  
< Visit <a href='https://github.com/itmo-wad'>github.com/itmo-wad</a>
```

VENV and Flask installation

Create and active virtual environment

Linux/MacOS

```
user@wad ~ % python3 -m venv my_env  
user@wad ~ % source my_env/bin/activate
```

Windows

```
py -m venv my_env  
.\my_env\bin\activate
```

Install flask

```
pip install flask
```

Deactivate virtual environment

Linux/MacOS

```
(my_env) user@wad ~ % deactivate
```

Windows

```
deactivate
```

Flask

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'

app.run(host="localhost", port=5000)
```



localhost:5000

Hello, World!

Demo

Templates

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def index():
    return render_template("index.html")

@app.route('/contacts')
def contacts():
    return render_template("contacts.html")

app.run(host="localhost", port=5000, debug=True)
```

- ▼ static
 - robots.txt
- ▼ templates
 - contacts.html
 - index.html
- 01_hello.py
- 02_templates.py

<http://localhost:5000/>

<http://localhost:5000/contacts>

Demo

Forms

```
<form action="/" method="POST">
```

```
  First name:<br>
```

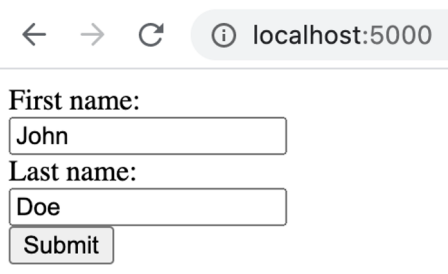
```
  <input type="text" name="fname" value="John"><br>
```

```
  Last name:<br>
```

```
  <input type="text" name="lname" value="Doe"><br>
```

```
  <input type="submit" value="Submit">
```

```
</form>
```



A screenshot of a web browser window. The address bar shows 'localhost:5000'. The page content displays a form with two text input fields. The first field is labeled 'First name:' and contains the text 'John'. The second field is labeled 'Last name:' and contains the text 'Doe'. Below these fields is a 'Submit' button.

Forms

```
from flask import Flask, render_template, request

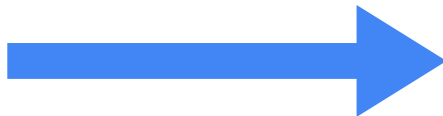
app = Flask(__name__)

@app.route('/', methods=["GET", "POST"])
def index():
    if request.method == "GET":
        return render_template("form.html")
    else:
        lname = request.form.get("lname")
        fname = request.form.get("fname")
        return render_template("cabinet.html", lname=lname, fname=fname)

app.run(host="localhost", port=5000, debug=True)
```

Hello, {{ fname }} {{ lname }}

Back



← → ↻ ⓘ localhost:5000

Hello, John Doe

[Back](#)

Flask routes

```
@app.route('/user/<username>')
```

```
def show_user_profile(username):
```

```
    # show the user profile for that user
```

```
    return 'User ' + username
```

```
@app.route('/post/<int:post_id>')
```

```
def show_post(post_id):
```

```
    # show the post with the given id,
```

```
    # the id is an integer
```

```
    return 'Post ' + str(post_id)
```

```
@app.route('/path/<path:subpath>')
```

```
def show_subpath(subpath):
```

```
    # show the subpath after /path/
```

```
    return 'Subpath ' + subpath
```

<converter:variable>



string	(default) accepts any text without a slash
int	accepts positive integers
float	accepts positive floating point values
path	like string but also accepts slashes
uuid	accepts UUID strings

Literature



- Documentation: <https://flask.palletsprojects.com/en/1.1.x/>
- Step-by-step tutorial: <https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world>
- Python simple tutorial: <https://pythontutor.ru/>
- HTML Forms: https://www.w3schools.com/html/html_forms.asp

Demo

HTTP data transfer

Input data

```
POST /method1/?method2=1234 HTTP/1.1
Host: itmo.xyz
User-Agent: curl/7.64.1
Accept: */*
Cookie: method4=asdf
Method5: zxcv
Content-Length: 12
Content-Type: application/x-www-form-urlencoded
```

```
method3=abcdHTTP/1.1 301 Moved Permanently
Server: nginx/1.16.1
Date: Thu, 02 Apr 2020 17:51:29 GMT
Content-Type: text/html
Content-Length: 169
Connection: keep-alive
Location: https://itmo.xyz/method1/?method2=1234
```

```
<html>
<head><title>301 Moved Permanently</title></head>
<body>
<center><h1>301 Moved Permanently</h1></center>
<hr><center>nginx/1.16.1</center>
</body>
</html>
```


Input data



Get data in Flask

```
@app.route('/<queryString>', methods=['POST'])
def index(queryString):
    getData = request.args.get("method2")
    postData = request.form.get("method3")
    cookie = request.cookies.get("method4")
    headers = request.headers.get("method5")
    return {
        "getData": getData,
        "postData": postData,
        "cookie": cookie,
        "headers": headers,
        "queryString": queryString
    }

if __name__ == "__main__":
    app.run(host='localhost', port=5000, debug=True)
```

`curl -X POST -H "Cookie: method4=444" -H "method5: 555" --data "method3=333"`

`http://localhost:5000/111?method2=222`

Demo

Literature

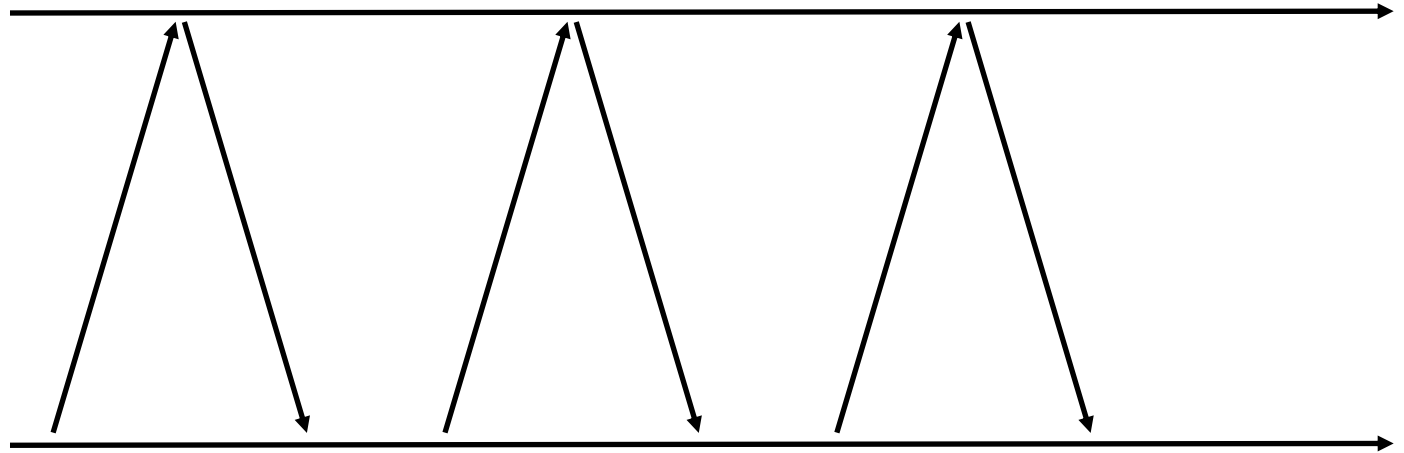


- GET and POST: https://www.w3schools.com/tags/ref_httpmethods.asp
- Cookie: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies>
- URL Encode: https://www.w3schools.com/tags/ref_urlencode.ASP
- POST Encode: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST>

Pub/Sub

Short polling, Long polling, WebSocket

Short polling



Slow updates

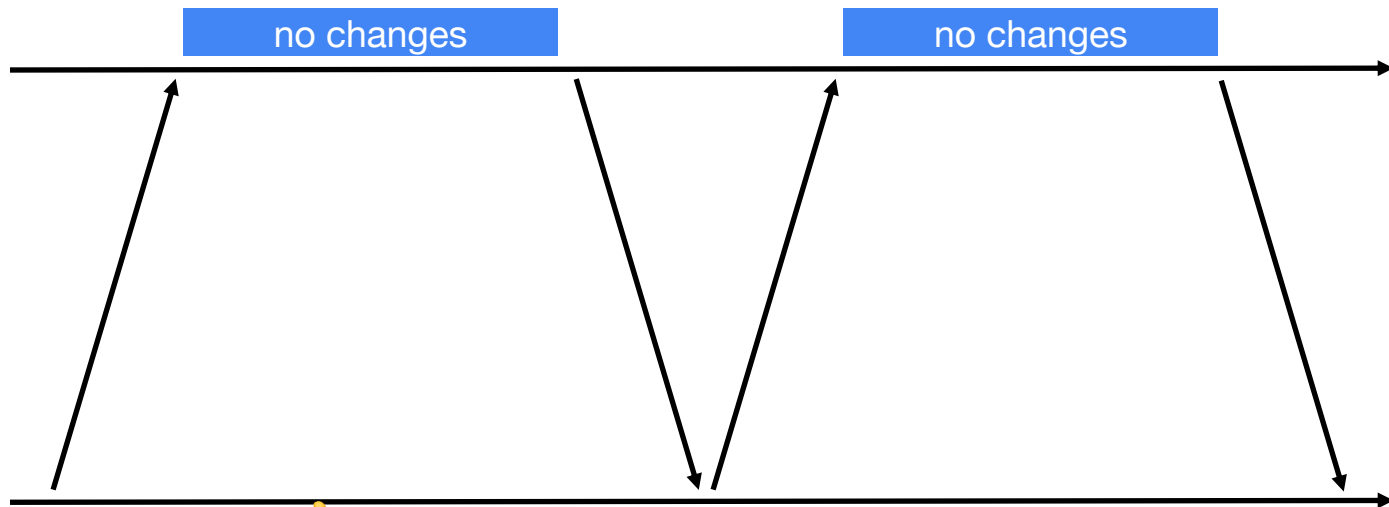


Resource intensive



Simple

Long polling



Fast updates



Less resource intensive



Kludge



Takes 1 worker

Web Sockets



Current aligned Usage relative Date relative Apply filters Show all ?									
IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini *	Android Browser *	Opera Mobile *
		2-3.6		3.1-4					
		¹ 4-5	¹ 4-14	¹ 5-5.1	10.1	3.2-4.1			
6-9		² 6-10	15	² 6-6.1	¹ 11.5	4.2-5.1		2.1-4.3	¹ 12
10	12-79	11-73	16-79	7-12.1	12.1-65	6-13.2		4.4-4.4.4	12.1
11	80	74	80	13	66	13.3	all	80	46
		75-76	81-83	13.1-TP		13.4			



Real time



Bidirectional



Efficient

Literature



- <https://javascript.info/websocket>
- <https://javascript.info/long-polling>
- <https://github.com/heroku-python/flask-sockets>
- <https://www.ably.io/blog/websockets-vs-long-polling/>

Homework principles

Reading → Coding → Deploy → Code-review

<https://wad.itmo.xyz/>

- Each homework in separate repository within your GitHub account (or in itmo-wad organization if you want)
- Submit link to the homework in Google Classroom
- Each homework has **description** in README.md

Homework 1

- Register a GitHub account, create a repository
- Fill link to the account in Google Doc. Join itmo-wad GitHub organization
- Frontend: static CV (profile) page
 - Content: heading, text, image
 - It may be visible as a README of your GitHub profile (if you have nothing against it)
- Backend: using flask to serve the profile page
 - Default route to serve profile page from template (can be with or without parameters)
 - Images, CSS files are as static resources
- Readme file
 - Short description about what have been done
- Github page (*.github.io) for the frontend part

Homework 1 — checklist

1. You have <https://github.com/yourname/yourname> repository with content which is visible on your profile page
2. You have <https://github.com/yourname/yourname.github.io> repository which is available over <https://yourname.github.io>
3. You have added your GitHub name to the Google Doc
4. You have a homework1 frontend&backend solution committed to the <https://github.com/itmo-wad/yourname-hw1> or to the any personal repository

Advice

- Communicate in the chat. Ask questions. Help other student with their questions.
- Don't copy other's code. Self-practice is the key.

Questions?