Web Application Servers

Spring 2024

Introduction to web servers

A web server listens to requests on a port

- every network-aware app listens on a port
- Web servers listen on port 80 (HTTP) or 443 (HTTPS) by default

Browsers assume those ports when not supplied explicitly

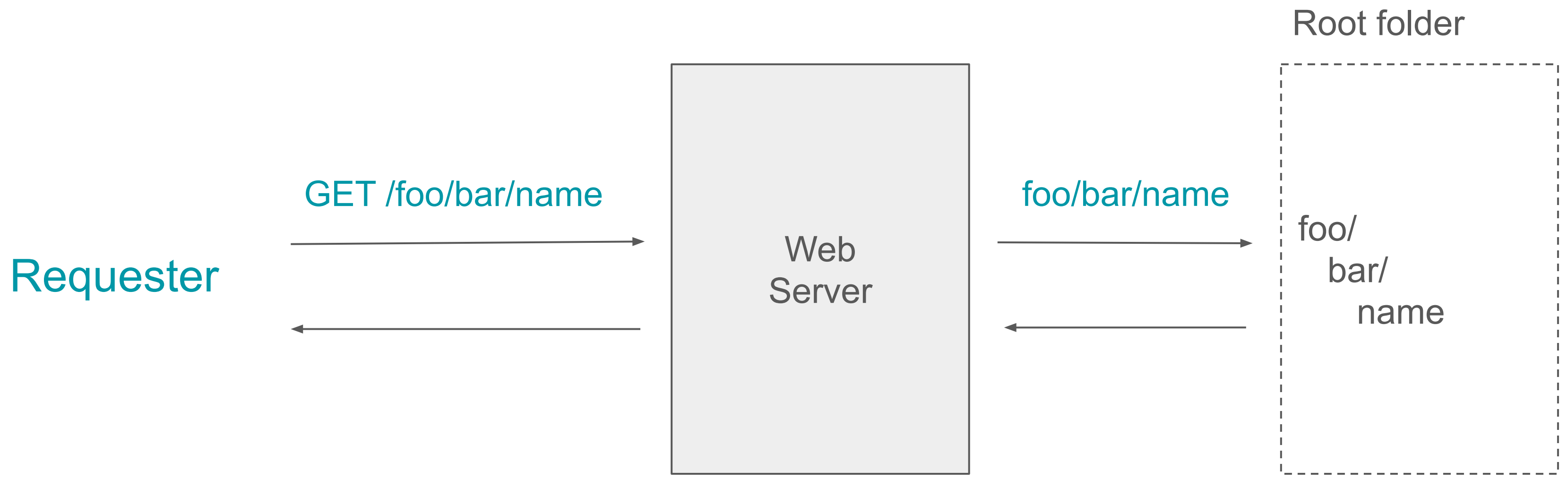
URL: **<protocol>://<hostname>:<port>/<route>**

- Protocol usually http or https
- hostname is the name of the machine running the server
- port is the port (can be dropped)
- route is the "location" of the file on the server

Classic Web Server

Respond to HTTP GET requests on a route:

- treat route as a path on the filesystem

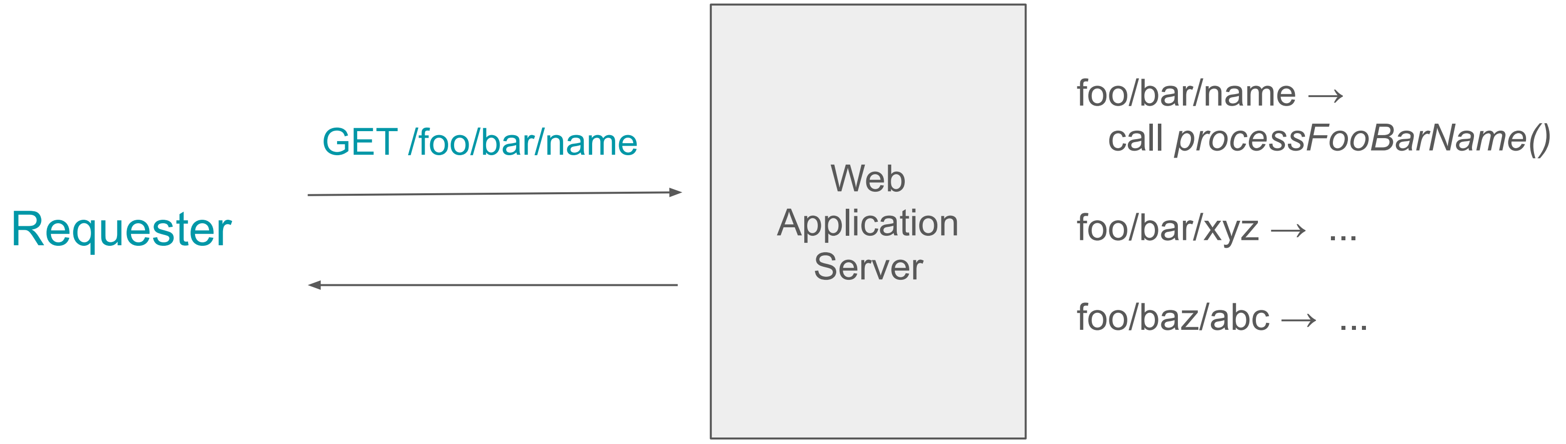


Demo — classic web server

Web Application Server

Respond to HTTP GET requests on a route:

- execute some code associated with the route that creates a response



Flask

How do you program Web Application Servers?

- Any programming language with a networking / HTTP library can be used
- Web Frameworks are libraries dedicated to creating Web Application Servers
- Different levels of scalability and "provide-X-out-of-the-box"
- Java → Spring Boot
  Node.js → Express
  Python → Django, **Flask**, …
  PHP → Symfony

We're going to use Flask: lightweight and fairly transparent

Flask

Tasks:

- create the server and run it on a port
- associate functions with routes

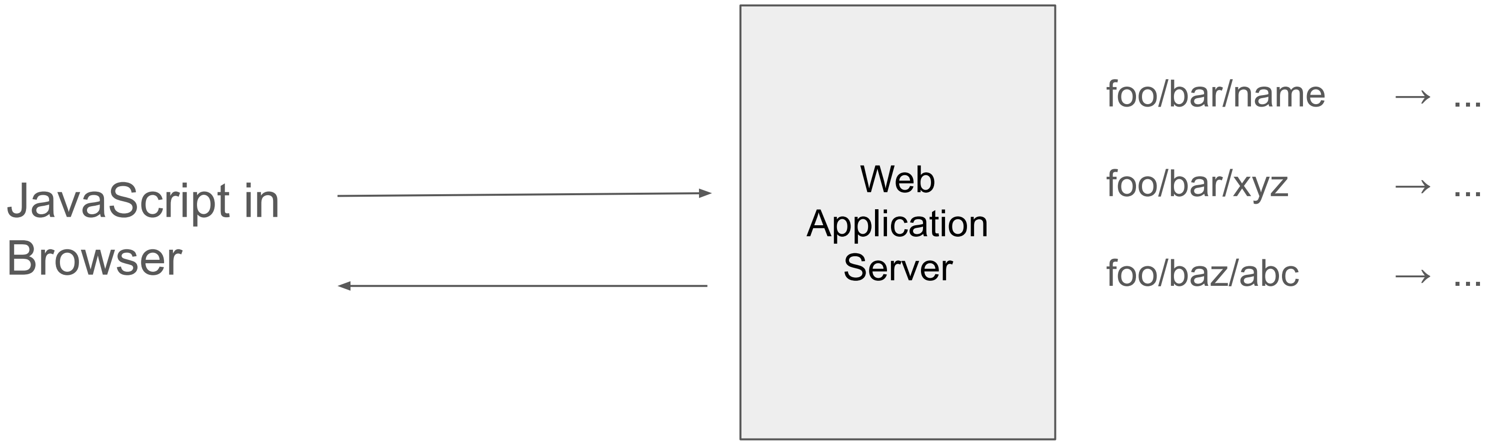
```
import flask

app = flask.Flask(__name__)
app.run(port=8080)
```

```
@app.route('/foo/bar/name')
def processFooBarName():
    return '<html>...</html>'
```

Demo — Flask

Web Application Server



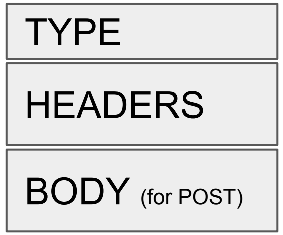
You can think of it as JavaScript making **function calls** to the server

- more expensive than normal function calls though
- why? server has more storage, shared between clients, survives refreshes...

HTTP requests and responses

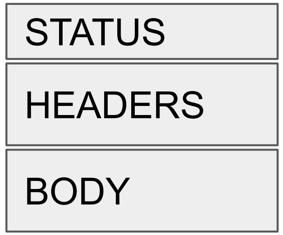
HTTP is a **text-based** protocol

- GET requests
  - used for requests that **don't change the server state**
  - minimal argument passing (query parameters in the route)
  - no body
  - what a browser sends from the browser bar
- POST requests
  - used for requests that **change the server state**
  - can pass "arbitrary" data in the body



Responses:

- status [200 = OK, 3xx / 4xx / 5xx = errors]
- response data in the body



GET requests

```
GET /index.html HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.feedbacknow.com
Connection: Keep-Alive
```

POST requests

```
POST /api/settime HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.feedbacknow.com
Content-Type: application/json
Content-Length: 34
Connection: Keep-Alive

{"DateTime":"2021-03-03 20:12:34"}
```

POST requests

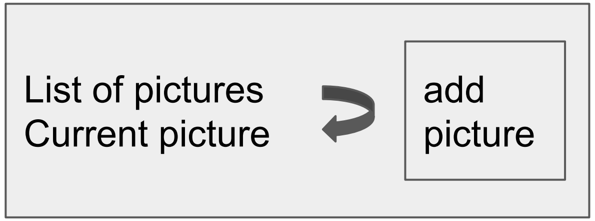
```
POST /api/settime HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.feedbacknow.com
Content-Type: application/json
Content-Length: 34
Connection: Keep-Alive

{"DateTime":"2021-03-03 20:12:34"}
```

MIME type —
describes the content of the body

Long catalog of MIME types

Our running demo



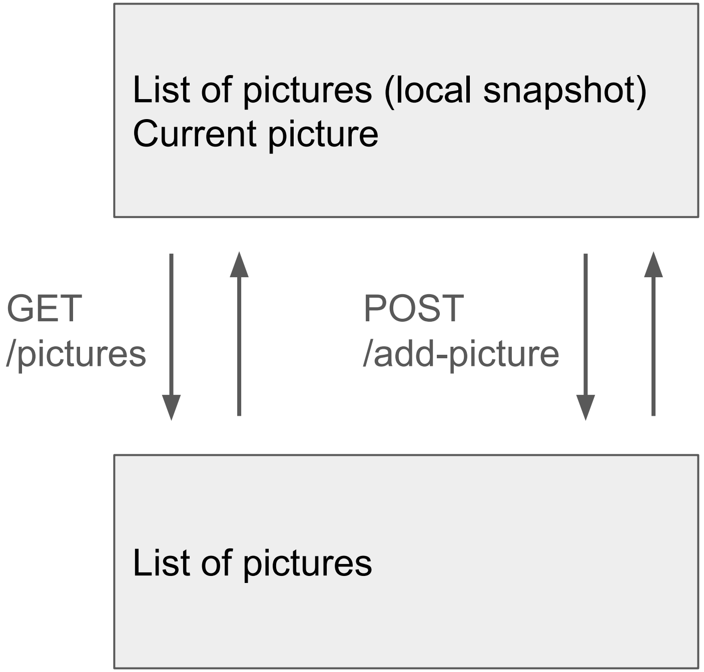
Maintain a list of pictures in the document (browser)

Can add a picture to the browser list

Any picture added is lost on browser refresh

What if we wanted to keep pictures across refreshes or over time?

Our running demo



One solution — keep the list of pictures on the server

When document loads, fetch a copy of the list from the server

When adding a picture, send it to the server

Demo

Pay attention to:

- how JavaScript gets data back from the **`/pictures`** endpoint
- how JavaScript sends a POST request to the **`/add-picture`** endpoint with a JSON body
- how Flask gets the JSON body out of the request

Distraction: where do you get the frontend?

Right now — we're opening the HTML document from the file system

Better to make it available via a URL

Who delivers it?

- Another web server
- The web application server itself