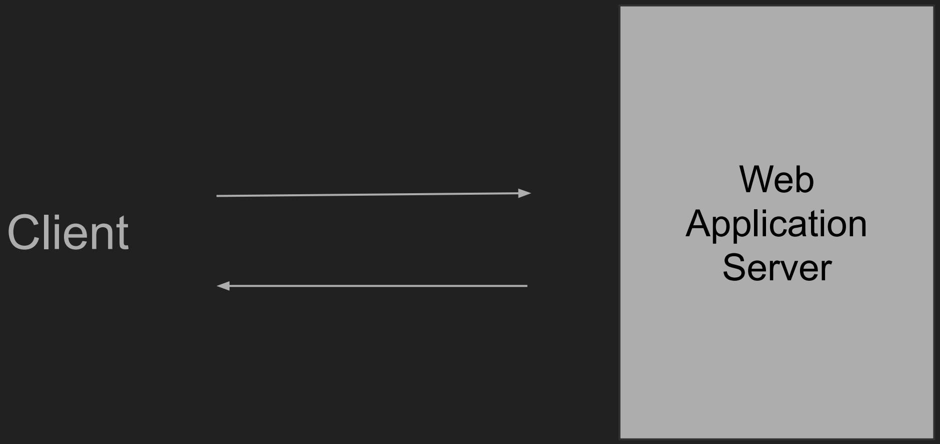
Forms and Distractions

Web Dev, Spring 2021

Summary — Web Application Server



ROUTES

`foo/bar/name`	→	...
`foo/bar/xyz`	→	...
`foo/baz/abc`	→	...

GET requests → don't change the state of the server, return a result
POST requests → can change the state of the server, can return a result

Distraction 1: Forms

A way to issue a POST request from an HTML document without using Javascript

```
<form action="http://localhost:8080/add-picture" method="post">
    <input id="input-title" type="text" name="title">
    <input id="input-url" type="text" name="url">
    <input type="submit" value="Submit URL">
</form>
```

Distraction 1: Forms

A way to issue a POST request from an HTML document without using Javascript

```
<form action="http://localhost:8080/add-picture" method="post">
    <input id="input-title" type="text" name="title">
    <input id="input-url" type="text" name="url">
    <input type="submit" value="Submit URL">
</form>
```

A form wraps a bunch of input elements

Distraction 1: Forms

A way to issue a POST request from an HTML document without using Javascript

```
<form action="http://localhost:8080/add-picture" method="post">
    <input id="input-title" type="text" name="title">
    <input id="input-url" type="text" name="url">
    <input type="submit" value="Submit URL">
</form>
```

This is rendered as a button — when clicked the form is "submitted"

Distraction 1: Forms

A way to issue a POST request from an HTML document without using Javascript

```
<form action="http://localhost:8080/add-picture" method="post">
    <input id="input-title" type="text" name="title">
    <input id="input-url" type="text" name="url">
    <input type="submit" value="Submit URL">
</form>
```

This defines the kind of HTTP request to use to "submit"

Distraction 1: Forms

A way to issue a POST request from an HTML document without using Javascript

```
<form action="http://localhost:8080/add-picture" method="post">
    <input id="input-title" type="text" name="title">
    <input id="input-url" type="text" name="url">
    <input type="submit" value="Submit URL">
</form>
```

Values of the input fields are sent with the request — keyed by name

Distraction 1: Forms

A way to issue a POST request from an HTML document without using Javascript

```
<form action="http://localhost:8080/add-picture" method="post">
    <input id="input-title" type="text" name="title">
    <input id="input-url" type="text" name="url">
    <input type="submit" value="Submit URL">
</form>
```

For a POST, values in the body are in *www-form-urlencoded* format:

title=*something*&url=*somethingelse*

Processing www-form-urlencoded bodies in Flask

How do you deal with this kind of body on the endpoint?

```python
@app.route('/add-picture', methods=['POST'])
def add_picture_route():
   title = request.form['title']
   url = request.form['url']
   # do something with title and url
```

Using www-form-urlencoded in fetch

You can also use x-www-form-urlencoded in a fetch instead of JSON:

```
fetch(url, {
  method: 'POST',
  headers: {
    'Content-Type': 'application/x-www-form-urlencoded'
  },
  body: 'foo=10&bar=20'
})
```

Some old APIs still require this

Distraction 2: Query parameters

We know we can send arguments to the server in a POST via a request body

- JSON object
- application/x-www-form-urlencoded
- multipart/form-data (for file uploads)

What about sending arguments in a GET?

- there is no request body in a GET
- use query parameters

Query parameters

```
http://hostname/path?key1=somevalue&key2=othervalue
```

The parameters are after the path, separated by a ?

- not part of the route
- they are encoded like *www-form-urlencoded*
- they are part of the URL, so easily visible in server logs (not great for secrets)
- can be used with any request/route (including POSTs)

Query parameters in Flask

How do you deal with this kind of body on the endpoint?

```
http://hostname/picture?id=34598734
```

```
@app.route('/picture')
def picture_route():
    id = request.args['id']
    # do something with id
```

Distraction 3: Where do you get the frontend?

We've been opening HTML documents (frontend) from the file system

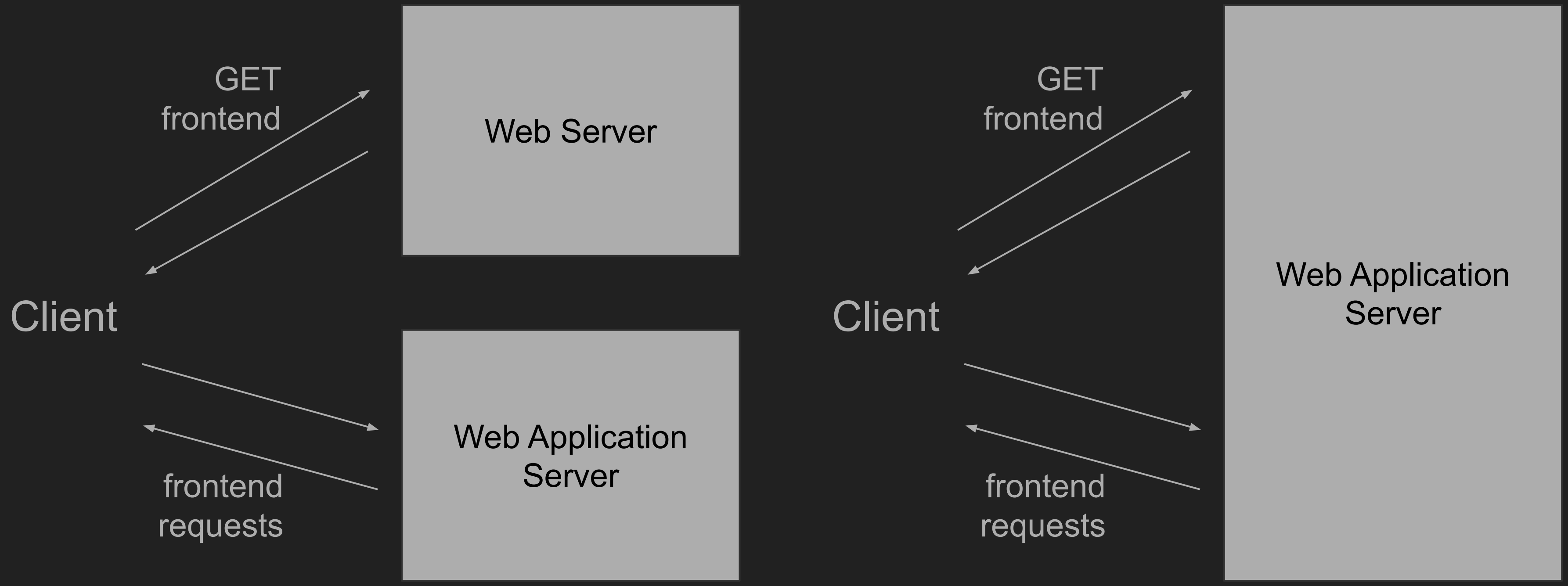
- the web application server implements endpoints that the documents can call

Better to make them available as documents via URLs

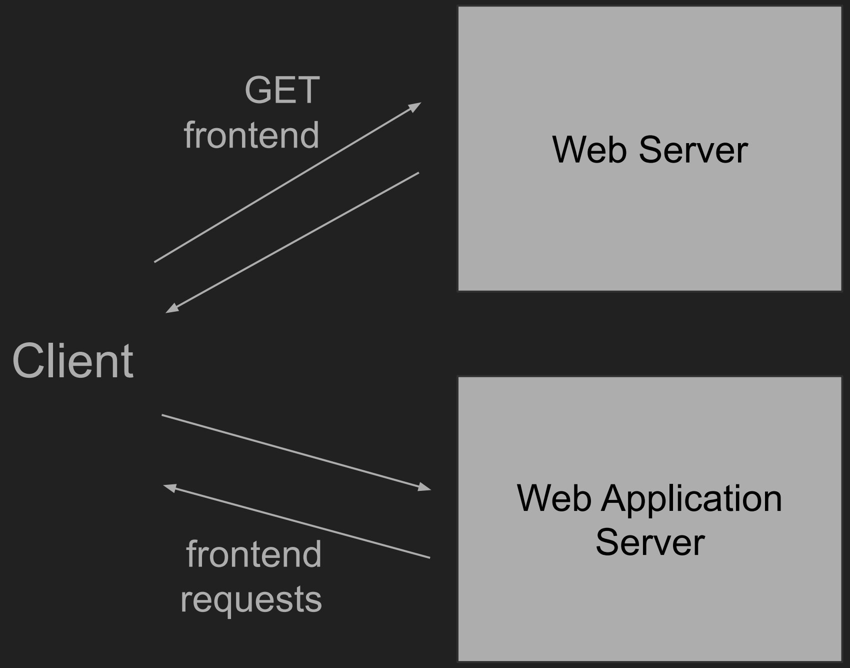
Who delivers them?

- Another web server
- The web application server itself

Deliver frontend options



Deliver frontend via a different web server



URLs in the frontend must be fully qualified

```
fetch('http://hostname/endpoint')
```

PROS: can use nginx/apache

- optimized for scalability
- reduces load on web app server

CONS: CORS

Deli

Cross-Origin Resource Sharing (CORS):

HTTP requests initiated from scripts are subject to the same-origin policy

- can only request resources from the same origin (server) the script was loaded from

To change that, the **server** needs to allow the script to make the request

GET: add header to the response:

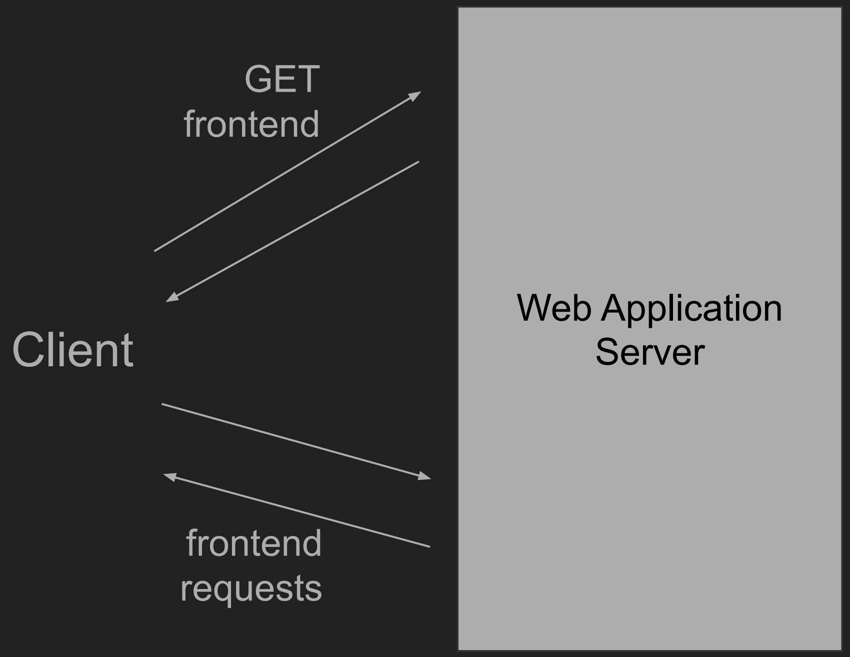
Access-Control-Allow-Origin

POST: fetch will first send an OPTION message (pre-flight request) that the server needs to respond to with headers:

Access-Control-Allow-Origin
Access-Control-Allow-Methods
Access-Control-Allow-Headers

Clie

Deliver frontend via the web application server



URLs in the frontend can be local

```
fetch('/endpoint')
```

PROS:	simpler deployment, no CORS
CONS:	load, scalability

Deliver frontend in Flask

- Create a folder `root` (name doesn't matter)
- Put the frontend code in `root`
- Define a route to send a file from `root` if no route matches

```
@app.route('/<path:p>')
def static_route(p):
    return send_from_directory('root', p)
```