Web Request-Response

Web Clients **send** a **web request**

Web Servers **listen** for **web requests**

Web Servers **respond** with **web responses**

Web Clients **receive web responses**

Basic HTTP is 1:1 request:response

- non-HTTP options exist for more
    - websockets, notifications
- Beyond this class

Request/Response Structure

- Request line (request only)
- Status line (response only)
- **Headers**
- Body

We can examine the request/response

- `curl -v URL` will show request/response
- **Browser DevTools** will let us see headers

Request line

The request begins with an **HTTP METHOD**

- Ex: `GET`, `POST`, etc

It then has the **path**

- Ex: `/search?q=smug+cats`, `/students/grades.html`
- Minimum `/`
- Plus any query parameters (part after `?`)
- Not the fragment (after any `#`)

It ends with the **protocol version**

- Ex: `HTTP/1.1`, `HTTP/2`, etc

Request line method

HTTP requests have one of a set list of "**methods**"

- Common data methods: GET, POST
- Additional methods for data
    - PUT, DELETE, PATCH
- Additional "management" methods
    - OPTIONS, TRACE, HEAD

Request Method Details

- We'll learn more about HTTP Methods soon
- For now, we'll look at GET and POST
    - Most common methods used

The HTTP "GET" Method

HTTP Method: GET

- Most basic request for info
- Following a link sends a GET request
- CAN send data in URL
- Data sent should not "change" data on server
    - Oversimplification, but a good starting point
- "idempotent"
    - Repeating the request is "safe"
- Has no "body" data in request!

HTTP "POST" Method

The HTTP Method: "POST"

- Can send additional data not in URL
    - Any such data is sent in body of request
- May change data on server
    - Example: making social media post

What does the Method *mean*?

- Server decides how to use functionality
- Different conventions exist
    - Later we will do REST
- At most basic level
    - GET used for getting pages
        - Links or manual typing will only GET
        - Cannot change data
        - Can provide info about desired data
    - POST used to create/update data
        - Front end requires HTML Form or JS
    - Any method can be sent by a program

Request line path

The **path** of the **request** line

- Includes any **query parameters**
  - Ex: `?foo=bar&baz=2`
- Does NOT include any **hash fragment**
  - Ex: `#foo`
  - Fragments are used by the client only

What response is server going to make?

Webserver decides what **response** to give

- Based on **method** + **path**
    - Define what to do
        - And what records to do it to
- **Parameters** usually considered later
    - Params are usually specific details

Examples:

- Do a search: `GET /search?q=smug+cats`
- Upload an image: `POST /profile/avatar`

Response line protocol version

Most requests are `HTTP/1.1`

- Most of the web only 1 small change from original
- PATCH was added in 2010; no change required

HTTP/2 also very common

- Mostly HTTP/1.1 + efficiency changes
- Binary, not text

HTTP/3 exists and is being worked on

- More efficiency/performance
- Same underlying concepts

A Vital Web Concept

Don't Break The Web

- Sounds easy
- Very hard if you also change
- HTTP, HTML, CSS, and JS
    - Decades of improvements
    - Across many vendors
    - Across countless web site providers
- Old stuff still works!
    - What's your oldest working non-web tech?

Request Headers

Headers are text key/value pairs, one per line

```
some-header-name: some-header-value
```

Headers are info ABOUT the request

- Date and time
- Format of any body content
- Preferred format for a response
- Any authorization info (more later)

Can be seen in your browser DevTools-Network Command Line example: `curl -v -s examplecat.com`

Request Body

The contents of the body can be....anything

Decided by any headers that define what to expect

Common options:

- URL-encoded key-value pairs
    - Ex: `foo=bar&baz=my%20cat`
- Structured text data
    - Ex: JSON, XML, etc
- Binary data
    - images, sound, etc

Request Complete!

Now the server has everything it needs to give a response

- We'll talk about **static** vs **dynamic** responses soon
- For now, we'll focus on sending the response back

Like you've asked a question over the phone

- They are going to make a response

Response Status Line

A web response starts with a line of 3 parts:

- Protocol version (just like end of request line)
- Numeric status code
- Text Status
    - Human readable text for numeric status code

HTTP Status Codes - 3 digit number

First digit: "category" of response/result

- 2xx - Success
- 3xx - Redirect
- 4xx - Client did something wrong
- 5xx - Server had a problem

https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

Example Status Codes via http.cat/

Specific number - better classification

4xx - Client Error examples:

- You need to login **401**
- Something doesn't exist **404**

5xx - Server Error examples:

- Server is down for maintenance **503**
- Server needs a system that is down **504**

These are still sweeping categories

- We'll discuss more when we write services

Response Text Status - For Humans

- Non-nerds don't memorize what 204 means

204

No Content

Response Status Line

- First line of Response
    - Protocol
    - Status Code
    - Status Text
- Leading with "is it good news or bad news"

Response Headers - Just like Request

```
some-header-name: some-header-value
next-header-name: that-header-value
```

- Headers are ABOUT the response
- Not the response itself
- Size
- Kind/Format of content

Headers tell us if the response is an image file or an HTML file

Response Body

- The actual content of response
- Can be ANYTHING
- Headers tell client what to expect
- HTML/CSS/JS are all text formats
- Also XML/JSON/CSV/YAML/etc
- Images/PDF/video/music are binary formats
    - Except SVGs

Web has two parts in conversation

- Request w/bad data will not get a good response
- Changing a good request won't fix a bad server

Any time a response sent data is wrong:

- Examine your requests/responses in the browser
- See which side is sending the wrong thing
- Even if you think it's correct
    - You're wrong somewhere or it'd be working!
    - Confirm, don't guess

Don't waste time solving a non-problem

Summary

- HTTP is a 1:1 series of
  - Client sending request
  - Server getting request
  - Server responding with response
  - Client getting response
- Request:
  - **method**, **path+query**, **headers**, **body** (maybe)
- Response:
  - **status**, **headers**, and **body** (usually)
- Can see requests/responses in browser DevTools