CIS 371 Web Application Programming

Fetch, Axios & Web Services

Lecturer: Dr. Yong Zhuang

Based on the original version by Professor Hans Dulimarta.

Which one?

Axios (npm library)

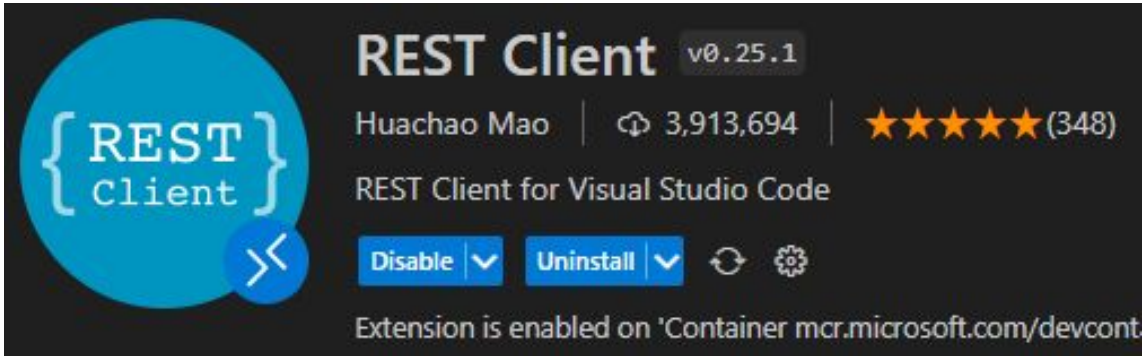



Fetch (browser built-in)

node-fetch (npm lib)

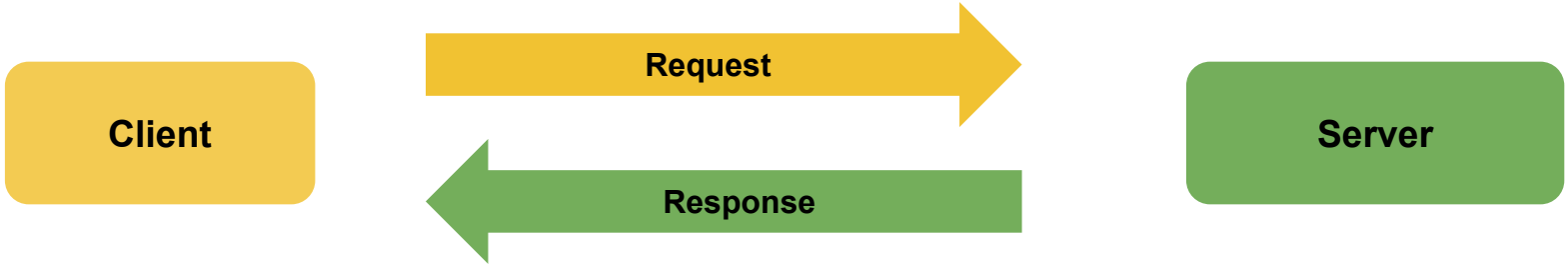
Topics

- Browser fetch() function
- NodeJS axios library
- Sending HTTP GET Requests
- Handling HTTP Responses
- Sending HTTP POST Request


REST Client
v0.25.1
Huachao Mao
3,913,694
(348)
REST Client for Visual Studio Code
Disable
Uninstall
Extension is enabled on 'Container mcr.microsoft.com/devcont


JSON Formatter
callumlocke.com
Featured
Makes JSON easy to read. Open source.
1,794
Developer Tools

HTTP Request & Response



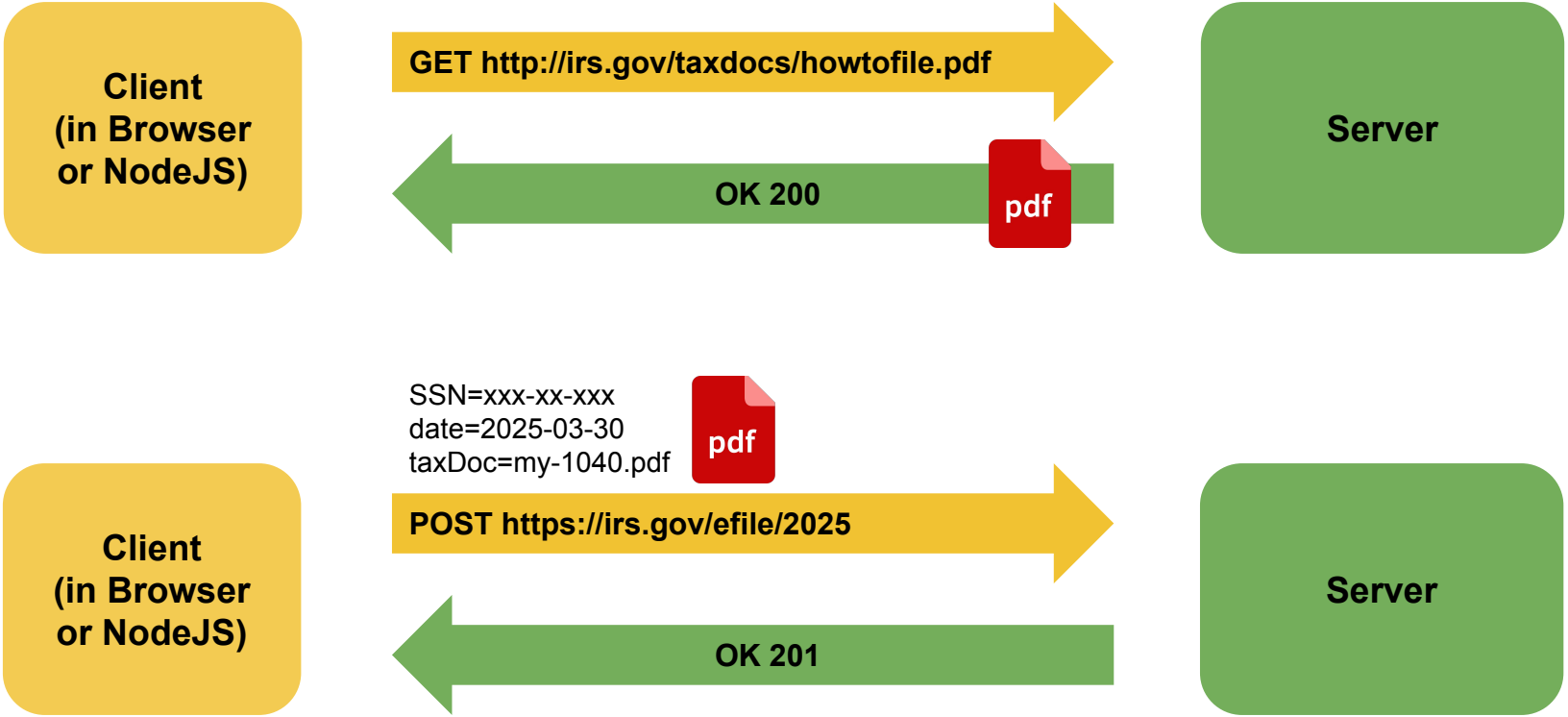
HyperText: *payload in response can be contents in* ***any*** *format (CSS, HTML, JPG, JS, JSON, PDF, PNG, ZIP, ….)*

Basic Use

```
fetch("http://some.website.org")
  .then((r: Response) => {
    //
    // Process the response
    // from the server here
    //
  })
  .catch((e: any) => {
    //
    // Handle potential
    // errors here
    //
  });
```

```
import axios, { AxiosResponse } from "axios";
axios
  .get("http://some.website.org")
  .then((r: AxiosResponse) => {
    //
    // Process the response
    // from the server here
    //
  })
  .catch((e: any) => {
    // Handle potential
    // errors here
  });
```

HTTP Request & Response



URL components

https://blog.logrocket.com/wp-content/plugins/assets/rocket-logo.png

resource name

Host name *Path*

query parameters

https://px.linkedin.com/api/collect?pid=14682&fmt=png&allowSave=true

```
{
    pid: 14682,
    fmt: "png",
    allowSave: true
};
```

Part A: Sending HTTP GET Requests

Using axios

```
npm install axios

# Add Axios type declaration file
npm install -D @types/axios
```

```
import axios, { AxiosResponse } from "axios";
axios
  .get("http://info.cern.ch")
  .then((resp: AxiosResponse) => {
    console.log(resp.headers);
    return resp.data;
  })
  .then((whatsInIt: any) => {
    console.log(whatsInIt);
  });
```

Example of Web Services

- Random Users
  - Documentation: https://randomuser.me/documentation
  - Service Endpoint: https://randomuser.me/api
- Random Quotes
  - Documentation: https://github.com/lukePeavey/quotable
  - Service Endpoint: https://api.quotables.io
- A gazillion more Web Services: https://github.com/public-apis/public-apis
  - Pick ones that allow CORS

Example #1: Random User

Browser https://randomuser.me/api

```
type RandUserData = {
  results: Array<RandomUser>;
  info: any;
};
type RandomUser = {
  name: {
    title: string;
    first: string;
    last: string;
  };
  email: string;
};
```

Important: *define these types to match the JSON structure of the API response.*

```
axios
  .request({
    method: "GET",
    url: "https://randomuser.me/api",
  })
  .then((resp: AxiosResponse) => resp.data)
  .then((incoming: RandUserData) => {
    console.log(incoming.info);
    for (let k = 0; k < incoming.results.length; k++) {
      console.log(incoming.results[k]);
    }
  });
```

Example #2: Random Quote

Browser http://api.quotable.io/random

```
type Quote = {
  tags: Array<string>;
  content: string;
  author: string;
  length: number;
};
```

```
axios
  .request({
    method: "GET",
    url: "http://api.quotable.io/random",
  })
  .then((resp: AxiosResponse) => resp.data)
  .then((q: Quote) => {
    console.log(`${q.content} [${q.author}]`);
  });
```

Example #3: Random User with Query Params

Browser https://randomuser.me/api?results=5&nat=gb,fr&inc=name,email,picture

```
type RandUserData = {
  results: Array<RandomUser>;
  info: any;
};
type RandomUser = {
  name: {
    title: string;
    first: string;
    last: string;
  };
  email: string;
};
```

```
axios
  .request({
    method: "GET",
    url: "https://randomuser.me/api",
    params: {
      results: 5,
      nat: "gb,fr",
      inc: "name,email,picture",
    },
  })
  .then((resp: AxiosResponse) => resp.data)
  .then((incoming: RandUserData) => {
    console.log(incoming.info);
    for (let k = 0; k < incoming.results.length; k++) {
      console.log(incoming.results[k]);
    }
  });
```

Example #4: Random Quotes with Query params

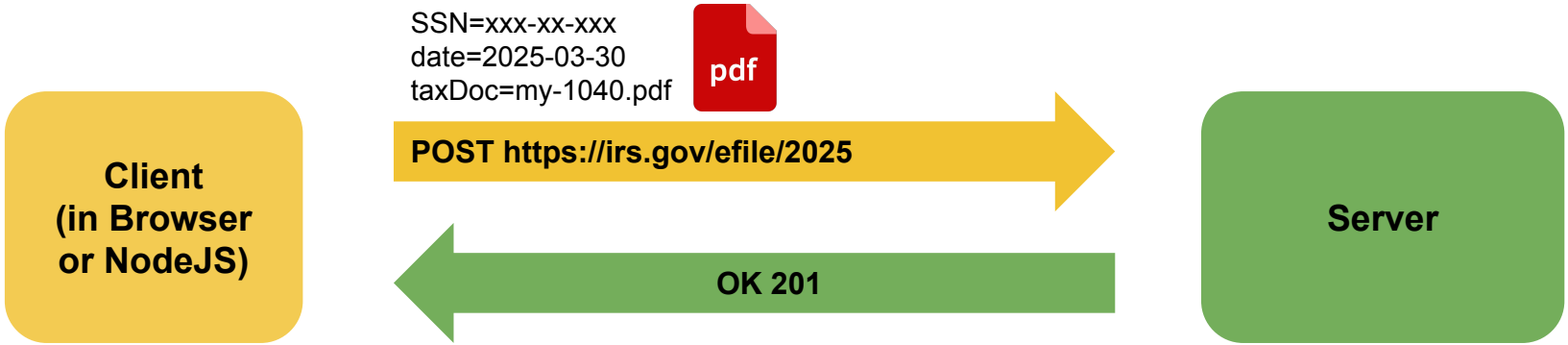
Browser http://api.quotable.io/quotes?limit=3

```
type QuoteResponse = {
  count: number;
  results: Array<Quote>;
};
type Quote = {
  tags: Array<string>;
  content: string;
  author: string;
  length: number;
};
```

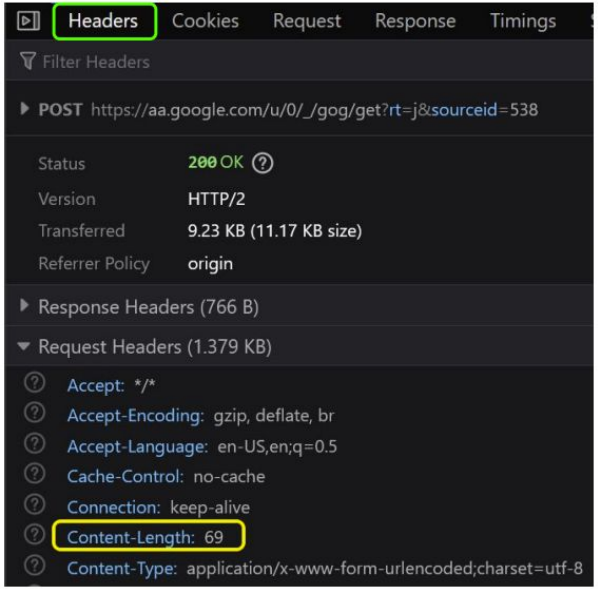
```
axios
  .request({
    method: "GET",
    url: "http://api.quotable.io/quotes",
    params: {
      limit: 3,
    },
  })
  .then((resp: AxiosResponse) => resp.data)
  .then((qr: QuoteResponse) => {
    console.log(qr.count);
    for (let q of qr.results) {
      console.log(q);
    }
  });
```

Part B: Sending HTTP POST requests ("Email" with attachments)

HTTP Post Request



Example: HTTP POST Request + Data Payload



Important options for POST

```
const options = {
  method: "POST",
  url: "https://aa.google.com/u/0/_____",
  data: "discussed-in-the-next-few-slides",
  headers: {
    "Content-Type": "discussed-in-the-next-few-slides",
  },
};
axios.request(options).then((r: Response) => {
  /* your code here */
});
```

HTTP Post Data Payload

Content-Type	Format of Data Payload	When to Use
application/x-www-form-urlencoded	Key-value pair, special characters are encoded using ASCII Hex	Multiple textual data items of relatively small size
multipart/form-data	Multiple “documents”, delimited by special lines. Binary data are encoded to hex	Multiple text or binary data items of larger size (images, PDF, ….), each item becomes one attachment of your “email”
application/json	JSON (converted to text)	Structured textual data
text/plain	Any plain text	Unstructured textual data

- *Request must include “Content-Type” header to inform server how to parse/unpack the data payload*
- *JSON is the easiest to use*
- *Plain text is rarely used*

HTTP POST: Plain Text attachment

```
const myMessage: string = "Hello World\n" +
"Do you copy?";
axios
  .request({
    method: "POST",
    url: "https://pizza4.me/order",
    headers: {
      "Content-Type": "text/plain",
    },
    data: myMessage,
  })
  .then((r: Response) => {
    /* code here */
  });
```

Rarely used in practice

Actual HTTP Request

```
POST /order HTTP 1.1
Host: https://pizza4.me
Content-Type: text/plain
Content-Length: 24
Hello World
Do you copy?
```

HTTP POST: JSON data attachment

```
const pizzaOrder = {
  size: 8,
  crust: "thin",
  toppings: ["cheese", "green pepper", "ham"],
  extraCheese: true,
};

axios
  .request({
    method: "POST",
    url: "https://pizza4.me/order",
    headers: {
      "Content-Type": "application/json",
    },
    data: JSON.stringify(pizzaOrder),
  })
  .then((r: Response) => {
    /* code here */
  });
```

- *Use JSON.stringify() to convert a JS/TS object to its string representation*
- *The data is delivered as plain/text*

Actual HTTP Request

```
POST /order HTTP/1.1
Host: https://pizza4.me
Content-Type: application/json
Content-Length: 89

{"size":8,"crust":"thin","toppings"
:["cheese","green pepper","ham"],
"extraCheese":true}
```

HTTP POST: application/x-www-form-urlencoded

```
const payload = new URLSearchParams();
payload.append("size", "8"); // must be a string
payload.append("crust", "thin");
payload.append("extraCheese", "true");
const toppings = ["cheese", "green pepper", "ham"];
for (let k = 0; k < toppings.length; k++) {
  payload.append("toppings[]", toppings[k]);
}

axios
  .request({
    method: "POST",
    url: "https://pizza4.me/order",
    headers: {
      "Content-Type": "application/x-www-form-urlencoded",
    },
    data: payload,
  })
  .then((r: AxiosResponse) => {
    /* code here */
  });
```

- *Key name for arrays ends with [] (empty brackets)*

Actual HTTP Request

```
POST /order HTTP/1.1
Host: https://pizza4.me
Content-Type: application/x-www-form-urlencoded
Content-Length: 93

size=8&crust=thin&extraCheese=true&toppings[]=ch
eese&toppings[]=green%20pepper&toppings[]=ham
```

HTTP POST: multipart/form-data

```
import FormData from "form-data";

const payload = new FormData();
payload.append("size", 8);
payload.append("crust", "thin");
const toppings = ["cheese", "green pepper", "ham"];
toppings.forEach((t) => {
  payload.append("toppings[]", t);
});

axios
  .request({
    method: "POST",
    url: "https://pizza4.me/order",
    headers: payload.getHeaders(),
    data: payload,
  })
  .then((r: AxiosResponse) => {
    /* code here */
  });
```

- *Use this for sending multiple data which can be expressed as key-value pairs of significantly large size*
- *Each data item can be text or binary*
- *The HTTP protocol enforces a limit on the maximum message size. A huge payload must be split into smaller chunks*

HTTP POST: multipart/form-data

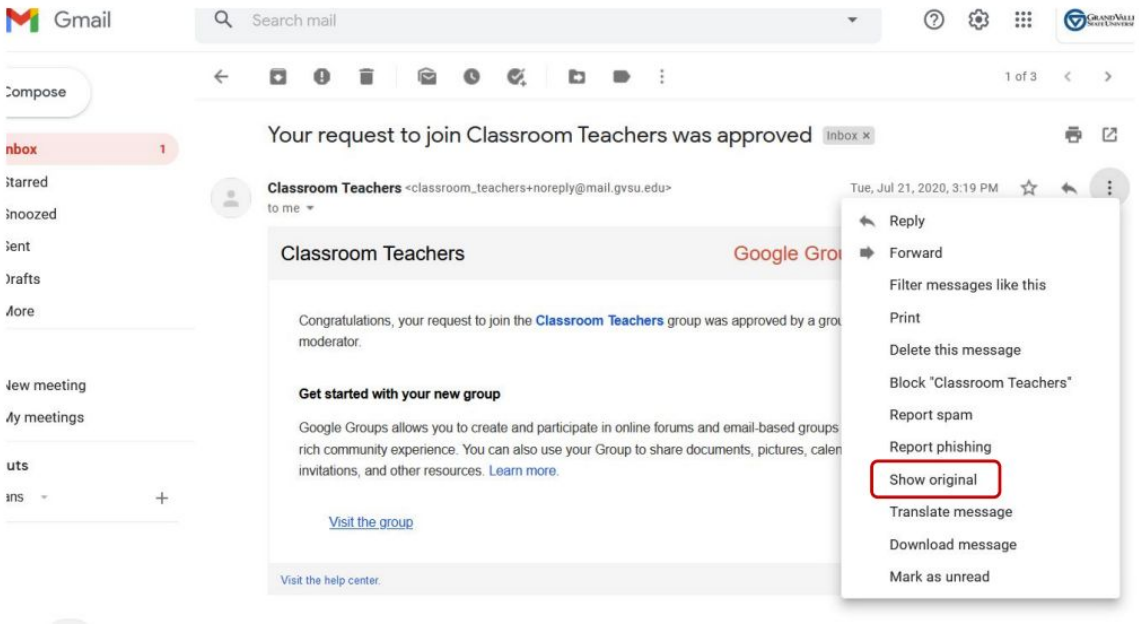
Actual HTTP Request

```
POST /order HTTP/1.1
Host: https://pizza4.me
Content-Type: multipart/form-data; boundary=letseatpizza
Content-Length: xxxx

--letseatpizza
Content-Disposition: form-data; name="size"
Content-type: text/plain

8
--letseatpizza
Content-Disposition: form-data; name="toppings[]"

cheese
--letseatpizza
Content-Disposition: form-data; name="toppings[]"

green pepper
--letseatpizza
Content-Disposition: form-data; name="extraCheese"

true
--letseatpizza--
```

- *The boundary text ("letseatpizza" in example) will be auto generated by the utility you use, axios*
- *Each part may include more headers.*
- *Parts with binary data will include Content-Type header with proper value (such as "image/jpeg") to inform the server how to unpack each part*

HTTP POST: multipart/form-data

Try this: For a real example of a multipart message, open a message that has attachments in GMail and select “Show Original”

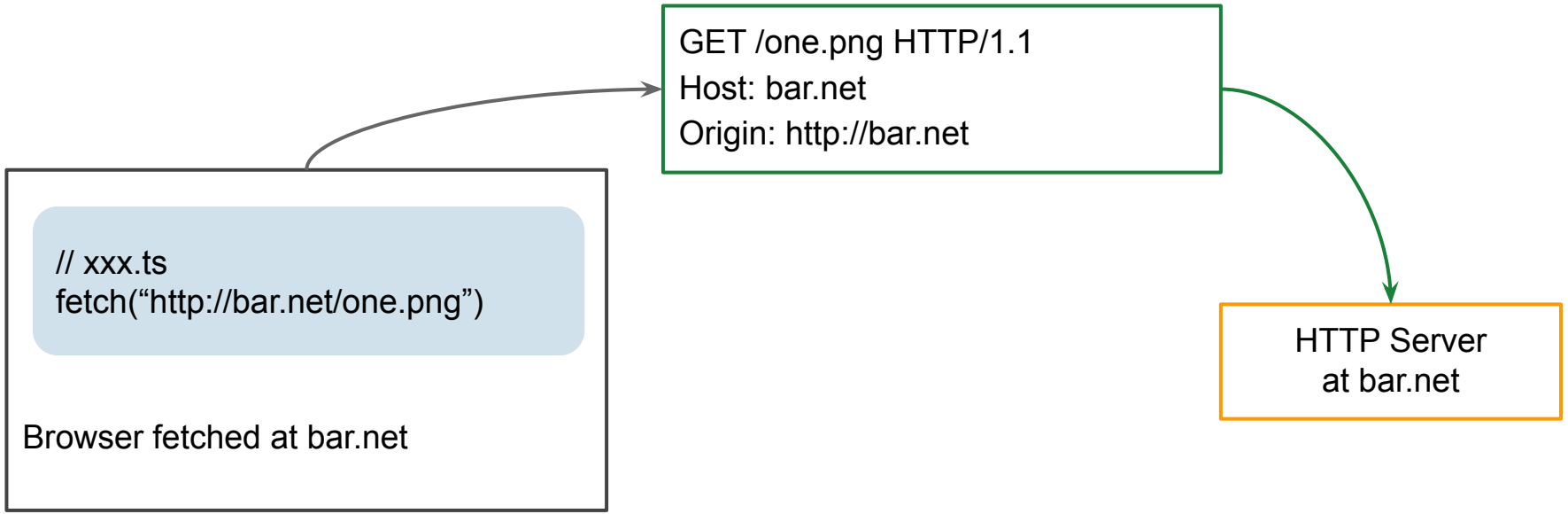


Security Issues

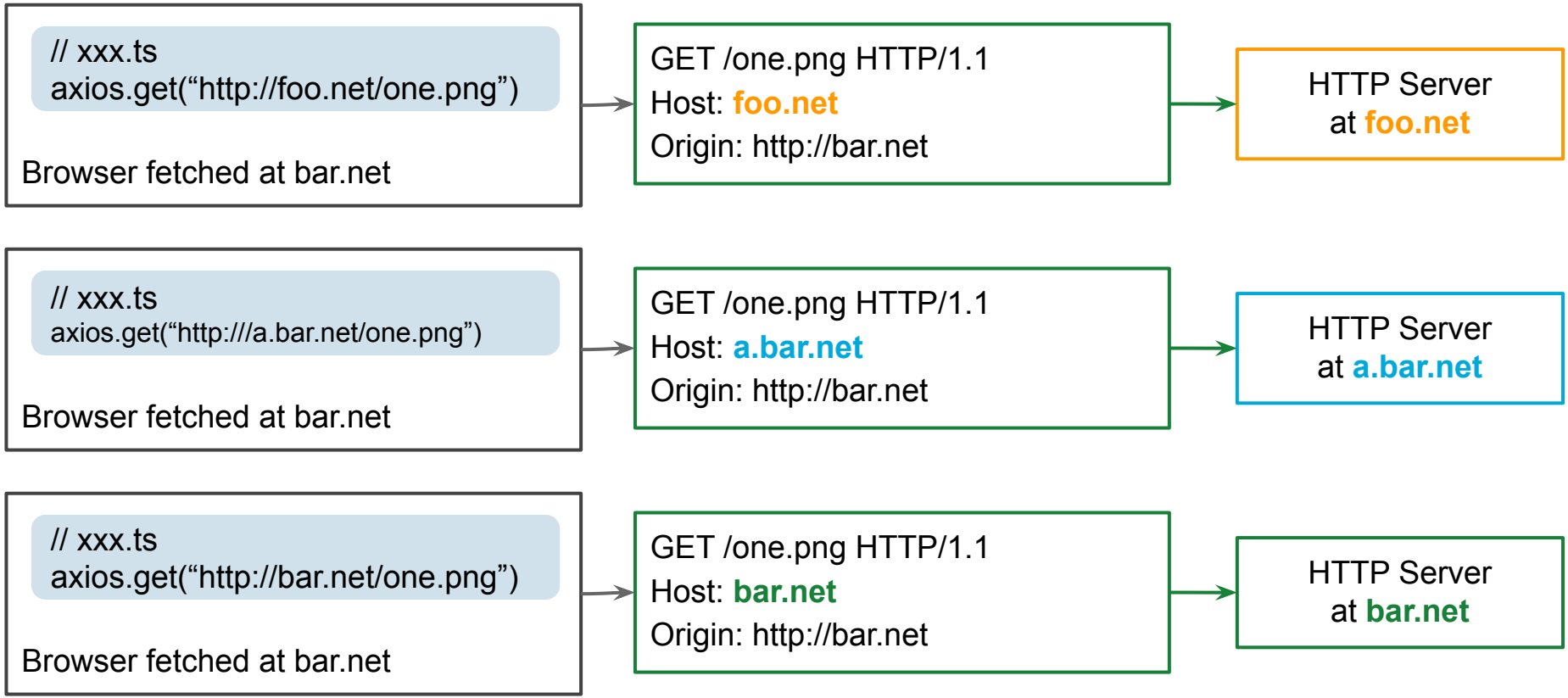
Browser Same-Origin Policy

- Scripts loaded from http://some.domain.net are allowed to fetch resources from http://some.domain.net
- Scripts loaded from **http://some.domain.net** are NOT automatically allowed to fetch from (cross-origin)
  - **https**://some.domain.net (different protocol)
  - http://some.domain**.org** (different domain)
  - http://some.**other-name**.net (different domain)
  - http://some.domain.net**:9000** (different port)
- Cross-origin requests require special handling by the server!

Same Origin


GET /one.png HTTP/1.1
Host: bar.net
Origin: http://bar.net
// xxx.ts
fetch("http://bar.net/one.png")
Browser fetched at bar.net
HTTP Server
at bar.net

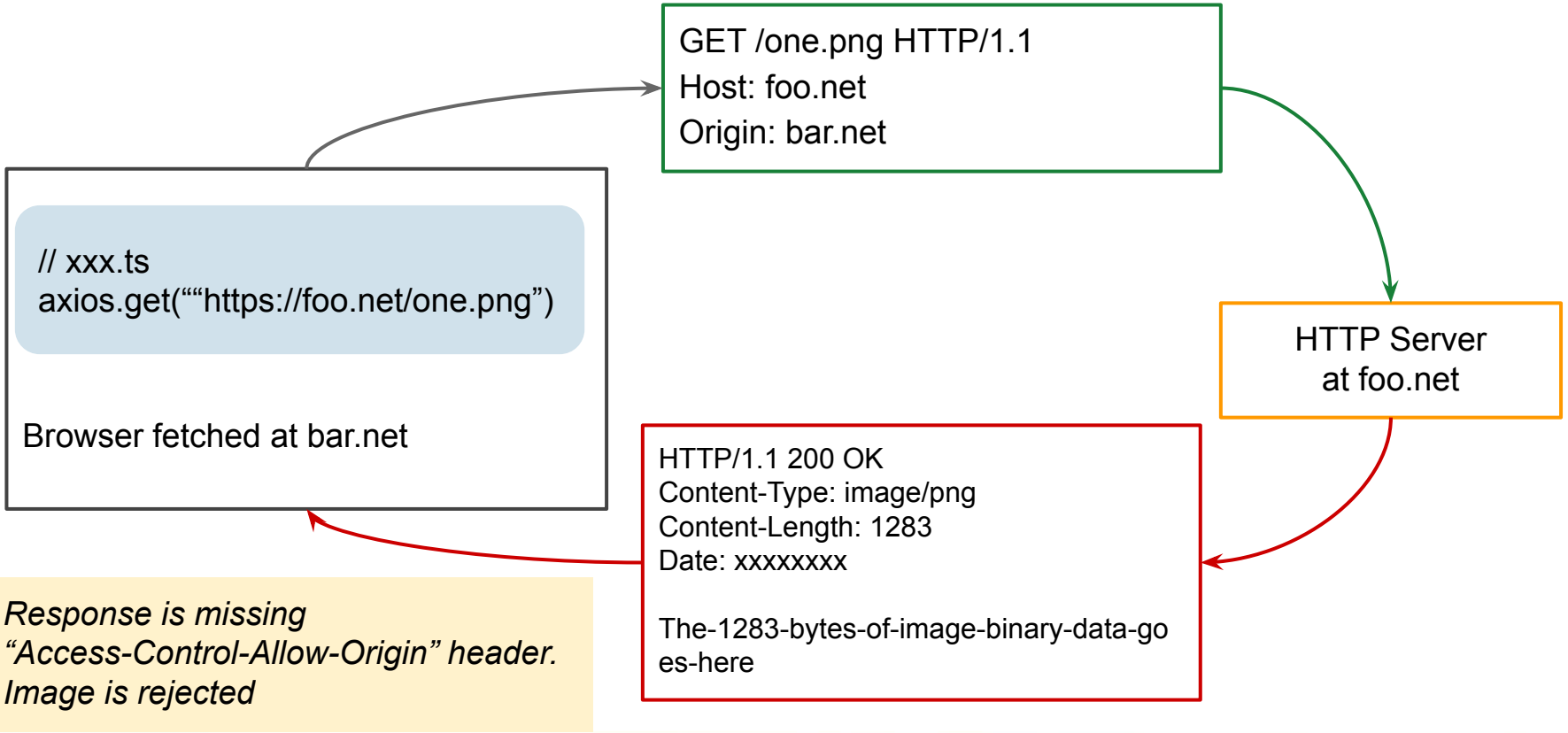
Different Origin (Cross Origin)



CORS (Cross-Origin Resource Sharing)

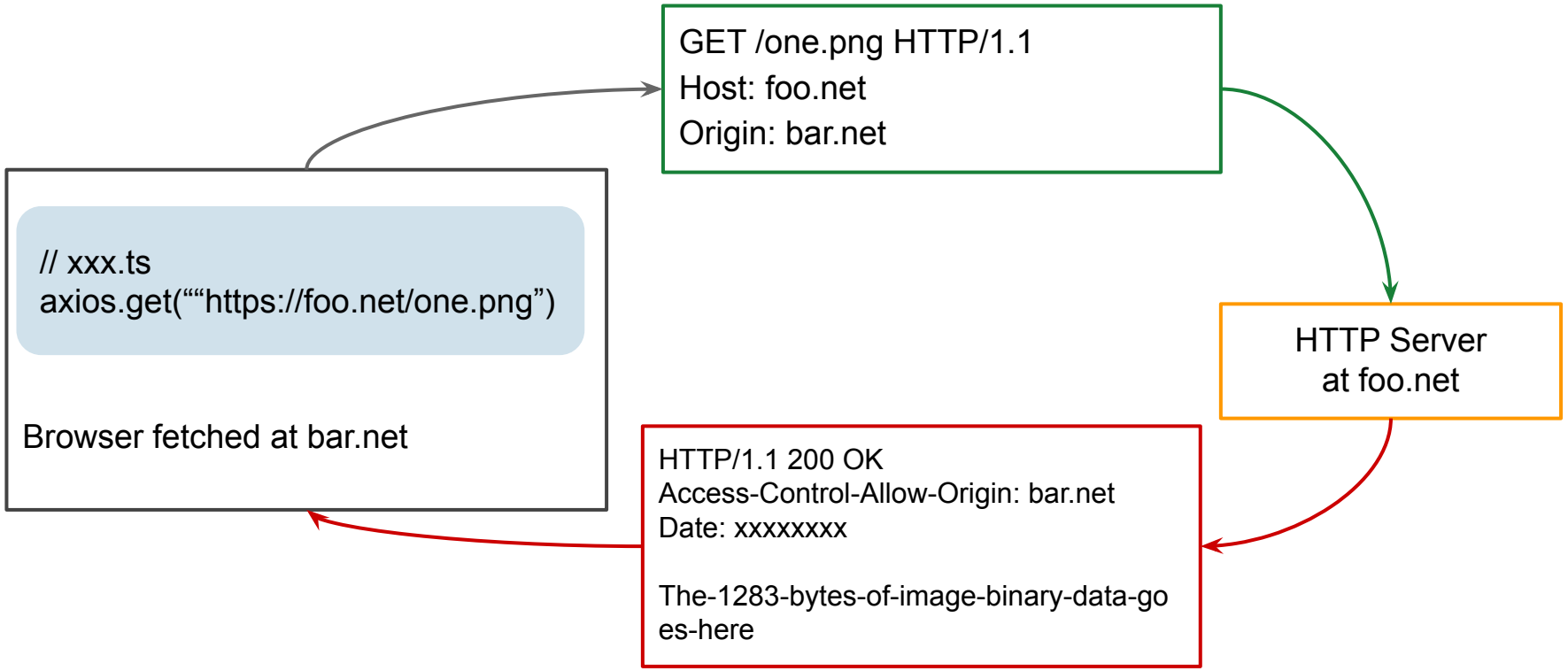
- New(er) spec to allow browsers “break” the same-origin policy
- Typical Client/Server negotiation sequence:
  - Client: send an HTTP OPTION query that include the following header lines
    - “Origin” and “Access-Control-Request-Method”
  - Server: responds with the following header lines
    - Access-Control-Allow-Origin
    - Access-Control-Allow-Methods

CORS: Rejected Response (Simplified)



Response is missing “Access-Control-Allow-Origin” header. Image is rejected

CORS: Accepted Response



GET https://randomuser.me/api?results=7&inc=name,email,picture
Status: HTTP/1.1 200

Request Headers	
Accept	text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Encoding	gzip, deflate, br
Accept-Language	en-US,en;q=0.9,id-ID;q=0.8,id;q=0.7,la;q=0.6
Cookie	__cfduid=d57b18855239032023e9c7617b85ef10c1538271818
Referer	[redacted]
Upgrade-Insecure-Requests	1
User-Agent	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.
Response Headers	
access-control-allow-origin	*
cache-control	no-cache
cf-ray	463b7ca8bd607e75-DTW
content-encoding	br
content-type	application/json; charset=utf-8
date	Wed, 03 Oct 2018 01:03:05 GMT
etag	W/"878-sPQkdJ3SNfIA02yPG/UQdA"
expect-ct	max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-cgi/beacon/expect-ct"
server	cloudflare
status	200

Solving CORS issue

- Change the server settings to enable CORS (only possible if you are the admin of that server)
- Access the service via a middleware
    - Your script (in the browser) sends the request to a middleware (running on the same host where keep the script)
    - The middleware then sends the actual request to the actual server. This strategy tricks the Browser as if the responses are coming from the middleware running on the same host
- Using a third-party Proxy server (in place of your own middleware)