

HTTP Protocol and Fetch API

CS445 Modern Asynchronous Programming

Maharishi University of Management

Department of Computer Science

Teaching Faculty: Assistant Professor Umur Inan

Prepared by: Associate Professor Asaad Saad

The Internet

A connection of computer networks using the Internet Protocol (IP)

layers of communication protocols:

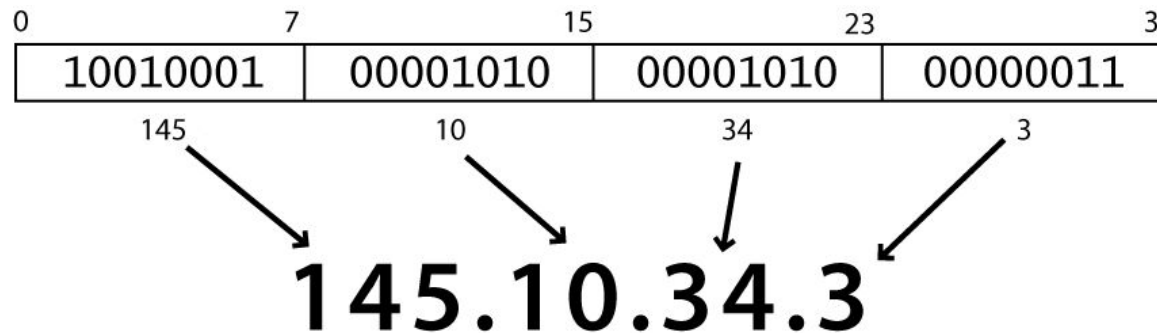
1. IP
2. TCP/UDP
3. HTTP/FTP/POP/SMTP/SSH...

Internet Protocol (IP)

A simple protocol for attempting to send data between two computers

Each device has a 32-bit IP address written as four 8-bit numbers (0-255)

There are two types of IP addresses, servers used to have **static IP** address while users usually get a **dynamic IP** address from their **ISP**.



IPv6 addresses are 128-bit IP address written in hexadecimal and separated by colons. An example IPv6 address could be written like this: 3ffe:1900:4545:3:200:f8ff:fe21:67cf

Transmission Control Protocol (TCP)

TCP (**Transmission Control Protocol**) is a standard that defines how to establish and maintain a network conversation via which application programs can exchange data. TCP works with the Internet Protocol (IP)

Multiplexing: multiple programs using the same IP address

- **port:** a number given to each program or service
- **port 80:** web browser (port 443 for secure browsing)
- **port 21:** ftp
- **port 22:** ssh

Domain Name System (DNS)

DNS servers map written names to IP addresses

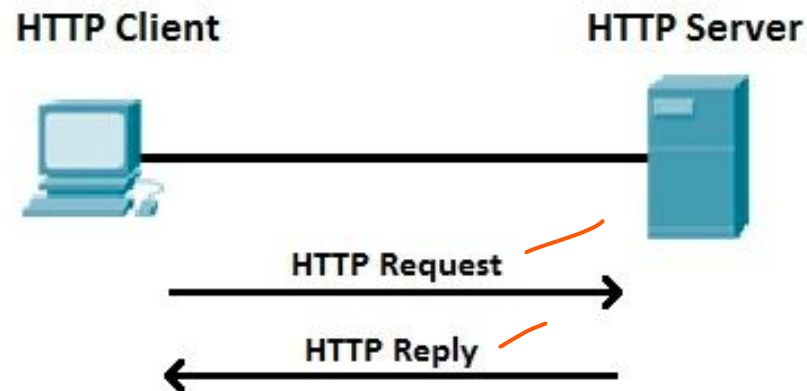
www.cs.miu.edu → 69.18.50.54

Many systems maintain a local DNS cache called a **host file**:

- Windows: C:\Windows\system32\drivers\etc\hosts
- Mac: /private/etc/hosts
- Linux: /etc/hosts

Hypertext Transport Protocol (HTTP)

HTTP is the underlying protocol used by the **World Wide Web** and this protocol defines **how messages are formatted**, and what actions Web servers and browsers should take in response to various **commands**.



Example

Request

Header {
GET /index.html HTTP/1.1
Host: www.miu.edu
User-Agent: Mozilla/5.0
Connection: keep-alive
Accept: text/html
If-None-Match: fd87e6789

Body {
No body with GET requests



Response

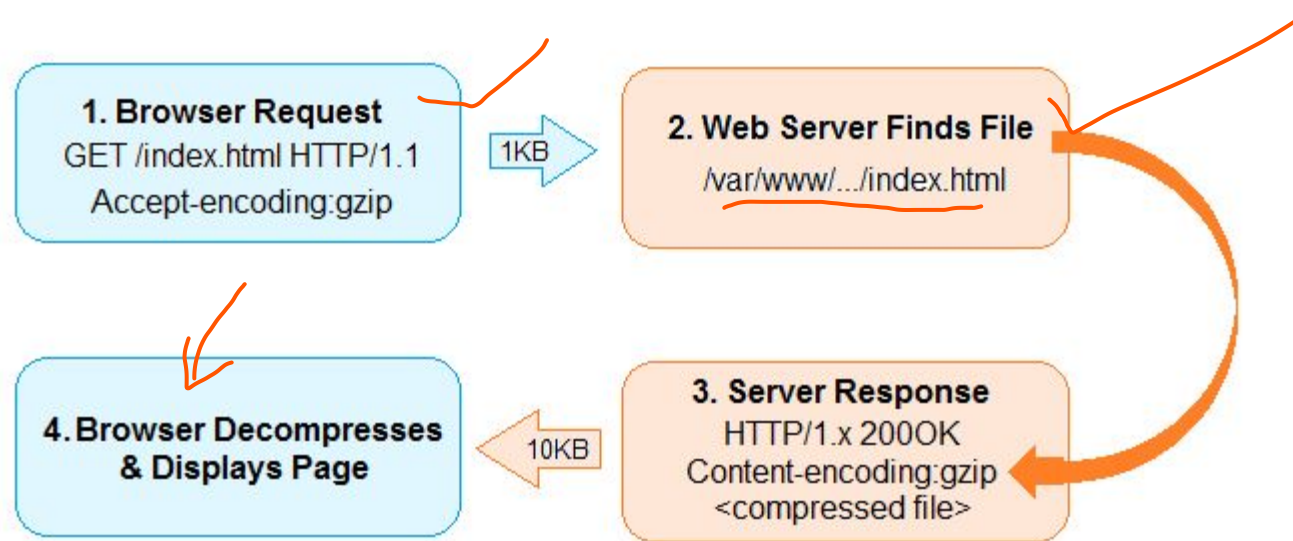
Header {
HTTP/1.1 200 OK
Content-Length: 16824
Server: Apache
Content-Type: text/html
Date: Monday 8 Dec 2020
Etag: h64w175

Body {
<!DOCTYPE html>
<html>
...

Demo: Inspect req/res in Chrome devtools.

HTTP Request

1. Your **browser** is going to send an HTTP request to the **server**
2. The server will prepare and send a response back
3. The browser will parse the response based on the **content-type**. If **text/html**, the **DOM** is created, JS runs and wait for user events.



HTTP Verbs

GET Retrieves data from the server

HEAD Same as GET, but response comes without the body

POST Submits data to the server

PUT Replace data on the server

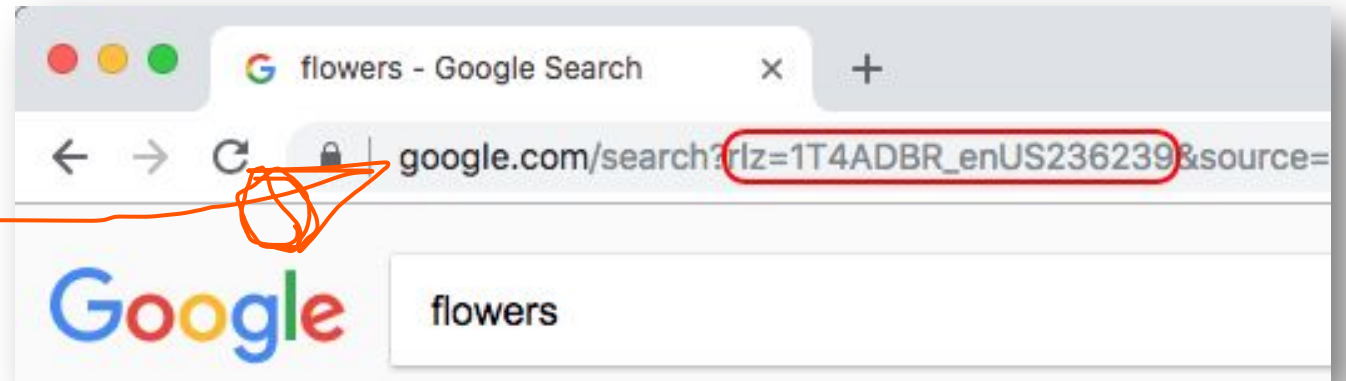
PATCH Partially update a certain data on the server

DELETE Delete data from the server

OPTIONS Handshaking and retrieves the capabilities of the server

How to send a Request?

URL (**GET** only)



Form element (**GET** and **POST** only)

A screenshot of a login form. It contains two input fields: 'Email Address' and 'Password'. Below these fields is a checkbox labeled 'Remember me'. At the bottom of the form is a blue button labeled 'Login'. An arrow points from the text 'Form element (GET and POST only)' to the 'Email Address' input field.

How to send a Request?

JavaScript (AJAX) with Fetch API (supports all verbs)

```
fetch('https://dog.ceo/api/breeds/image/random')  
.then(response => {  
  console.log(response.headers.get('Content-Type'))  
  console.log(response.headers.get('Date'))  
})
```

< ▶ *Promise {<pending>}*

application/json

Sun, 22 Apr 2018 05:05:00 GMT

Uniform Resource Locator (URL)

Anchor: jumps to a given section of a web page

`http://www.mum.edu/download/index.html#downloads` ✓

Port: for web servers on ports other than the default 80

`http://www.cs.mum.edu:8080/msd/mwp.txt` ✓

Query string: a set of parameters passed to a web program

`http://www.google.com/search?q=mum&start=10` ✓

A typical URL would have a protocol, a hostname, and a file name.

A Web Page Story

Once upon a time, users had dynamic IP, and servers had static IP and Domain Name.

Domain names and IP addresses are registered at global DNS Server.

When the user opens a browser window and ask for www.test.com, the browser will check the local DNS (host file) for the IP address of that domain, if not found, it will connect to ISP and ask for it.

Once retrieved, the browser will establish a TCP connection and send a request to that server via HTTP or HTTPS/TLS protocol.

The server will send the browser a response with body contains HTML code.

The browser will parse the HTML code line by line and start building the DOM.

For every resource not found in the browser cache, the browser will send a new request to the server again asking for that resource and so on.

Encryption vs. Hashing

Encryption is a **two-way function**, what is encrypted can be decrypted with the proper key. Usually a key and salt are used.

Example: base64 (atob, btoa), RSA, Crypto, AES

Hashing is a **one-way function** that scrambles plain text to produce a **unique message digest** (most of time with a **fixed size**). There is no way to reverse the **hashing process** to reveal the original plain text.

Example: MD5, SHA1/256/512, PBKDF2, BCrypt

How do we save passwords in your DB server?



SHA (Secure Hash Algorithm)

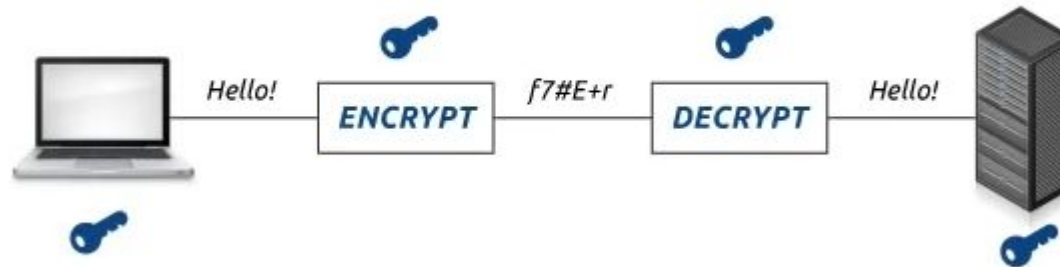
The SHA (Secure Hash Algorithm) is a **cryptographic hash function**. You may think of it as a signature for text or data file.

SHA-256 algorithm generates an **almost-unique**, fixed size 256-bit (32-byte) hash. Hash is a one-way function – it cannot be decrypted back.

GitHub uses SHA algorithm to generate a hash for every commit.

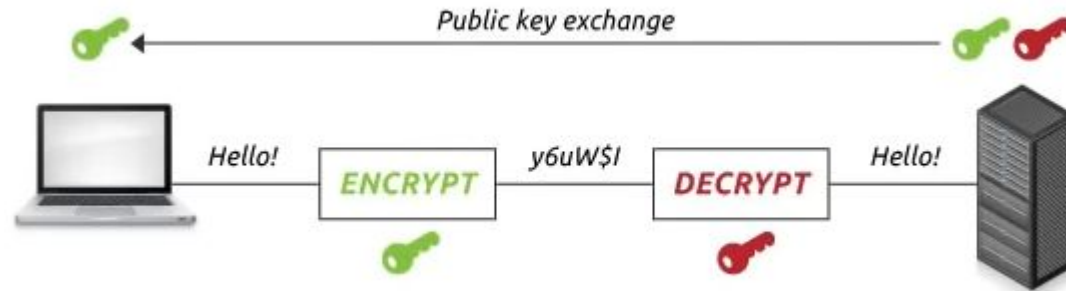
Symmetric Encryption

Symmetric cryptography is a type of encryption where the same key (**secret key**) is used to both encrypt and decrypt the data. Usually it's very fast because it uses 256 bits key.



Asymmetric Encryption

Asymmetric cryptography (public key cryptography), uses public and private keys to encrypt and decrypt data. **Public key** is used to encrypt the data, and the **Private key** is used to decrypt the data. While this is more secure because it uses 2048 key, it is slow.



Digital Certificate

A certificate is an electronic credentials used to assert the online identities of individuals, applications and other entities on a network. They are similar to an ID card (passport, driving license, diploma). A certificate is signed by (Certificate Authority CA).

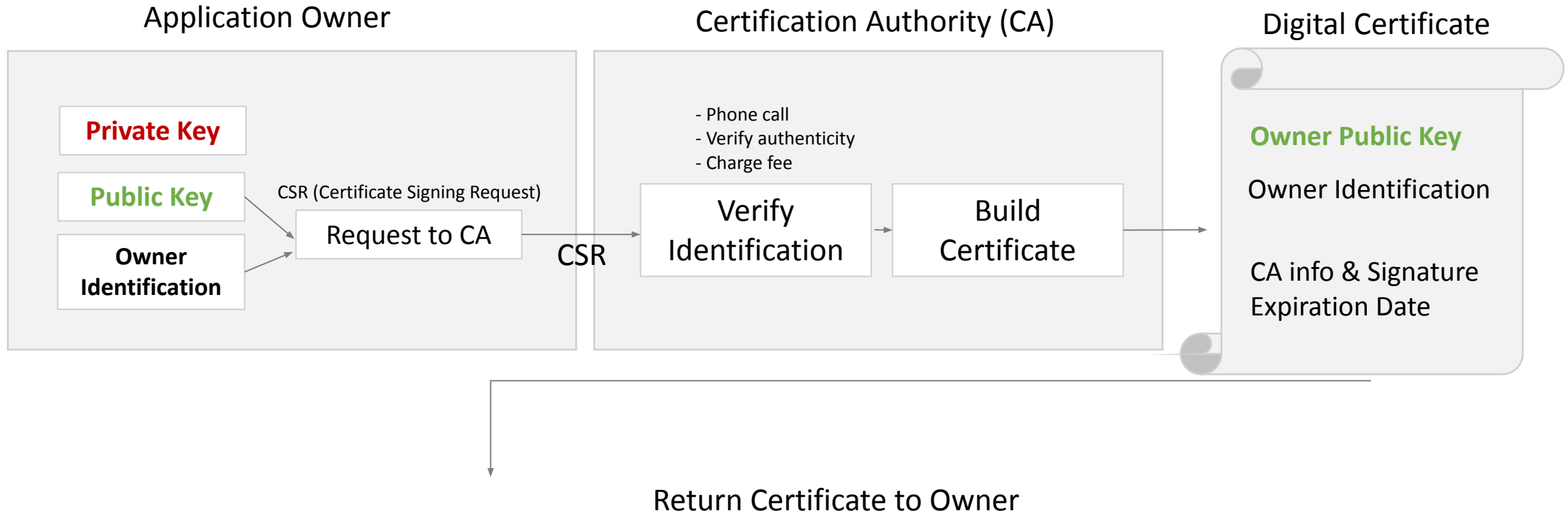
A certificate is a file contains:

- Identity of owner
- Public Key of the owner
- Signature of the certificate issuer
- Expiration date

A signature is always made with the private key of issuer and can be validated with the public key of the issuer.

Anyone with access to the server's public key, can verify that a signature could only have been created by the server's private key.

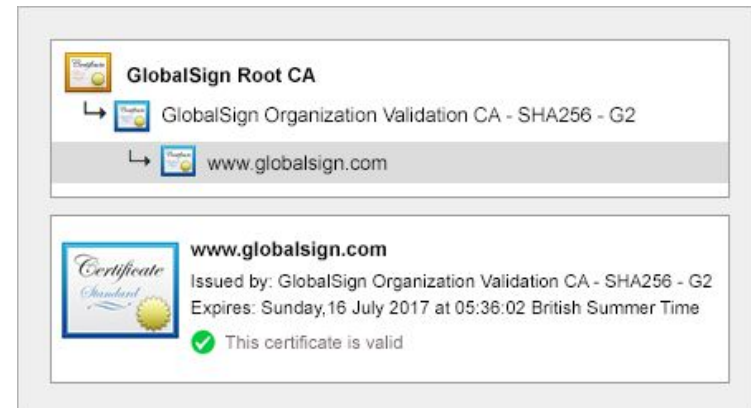
How Digital Certificates Are Created?



Using Digital Certificates

When browsers contact an application owner via TLS, the first thing they receive is the digital certificate. Browsers check the validity of this certificate (if it was issued to the claimed app owner, not expired, the CA signature).

Usually a **chain of certificates** are sent to the browser (the owner certificate signed by ICA, and ICA certificate signed by Root CA), since the browser has the public key for all Root CAs, it validates the ICA certificate first and make sure it is a certified entity, and then use it's public key in that certificate to validate the owner certificate.



TLS Handshake

Browser: I want to connect to `https://miu.edu`

MIU Server: Sure, here's my certificate containing my asymmetric public key. It was signed by Comodo CA.

Browser: validates the signature of the certificate from the saved public keys of trusted issuers, if valid, it creates a new symmetric secret key and encrypts it with the asymmetric public key of MUM found in the certificate.

Server decrypts the asymmetric public key with its asymmetric private key to get the symmetric secret key

All future communication is encrypted and decrypted with the symmetric secret key.

If the browser was to connect to the same server the next day, a **new secret key** would be created.

Web Communication

We generally use the **asymmetric system** to exchange the secret key and then **symmetric system** to encrypt and decrypt messages.

The server sends its public key to the client

The client generates a secret key and encrypt it with the server's public key and sends it to the server

The server with its private key will decrypt and read the secret key

All future communication will be done using the symmetric encryption/decryption using the secret key (faster).

JavaScript Object Notation (JSON)

JavaScript Object Notation (JSON): Data format that represents data as a set of JavaScript objects

- Natively supported by all modern browsers
- Replaced XML (Extensible Markup Language)

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<note private="true">
  <from>Alice Smith (alice@example.com)</from>
  <to>Robert Jones (roberto@example.com)</to>
  <to>Charles Dodd (cdodd@example.com)</to>
  <subject>Tomorrow's "Birthday Bash"
event!</subject>
  <message language="english">
    Hey guys, don't forget to call me this
weekend!
  </message>
</note>
```

JSON

```
{
  "private": "true",
  "from": "Alice Smith (alice@example.com)",
  "to": [
    "Robert Jones (roberto@example.com)",
    "Charles Dodd (cdodd@example.com)"
  ],
  "subject": "Tomorrow's \"Birthday Bash\" event!",
  "message": {
    "language": "english",
    "text": "Hey guys, don't forget to call me this weekend!"
  }
}
```


JavaScript Object Notation (JSON)

JSON is a syntax similar to JS Objects for storing and exchanging data and an efficient alternative to XML

```
{ "students": [  
  { "firstName": "Ashim", "lastName": "Ghimire" },  
  { "firstName": "Mohamed", "lastName": "Hassan" },  
  { "firstName": "Leul", "lastName": "Necha" },  
  { "firstName": "Shawn", "lastName": "Daudi" },  
]
```

A name/value pair consists of a field name **in double quotes**, followed by a colon, followed by a value. Values can be any JS valid type except functions.

Browser JSON Methods

Method	Description
JSON.parse(<i>string</i>)	Converts the given string of JSON data into an equivalent JavaScript object and returns it
JSON.stringify(<i>object</i>)	Converts the given object into a string of JSON data (the opposite of JSON.parse)

JSON Exercise

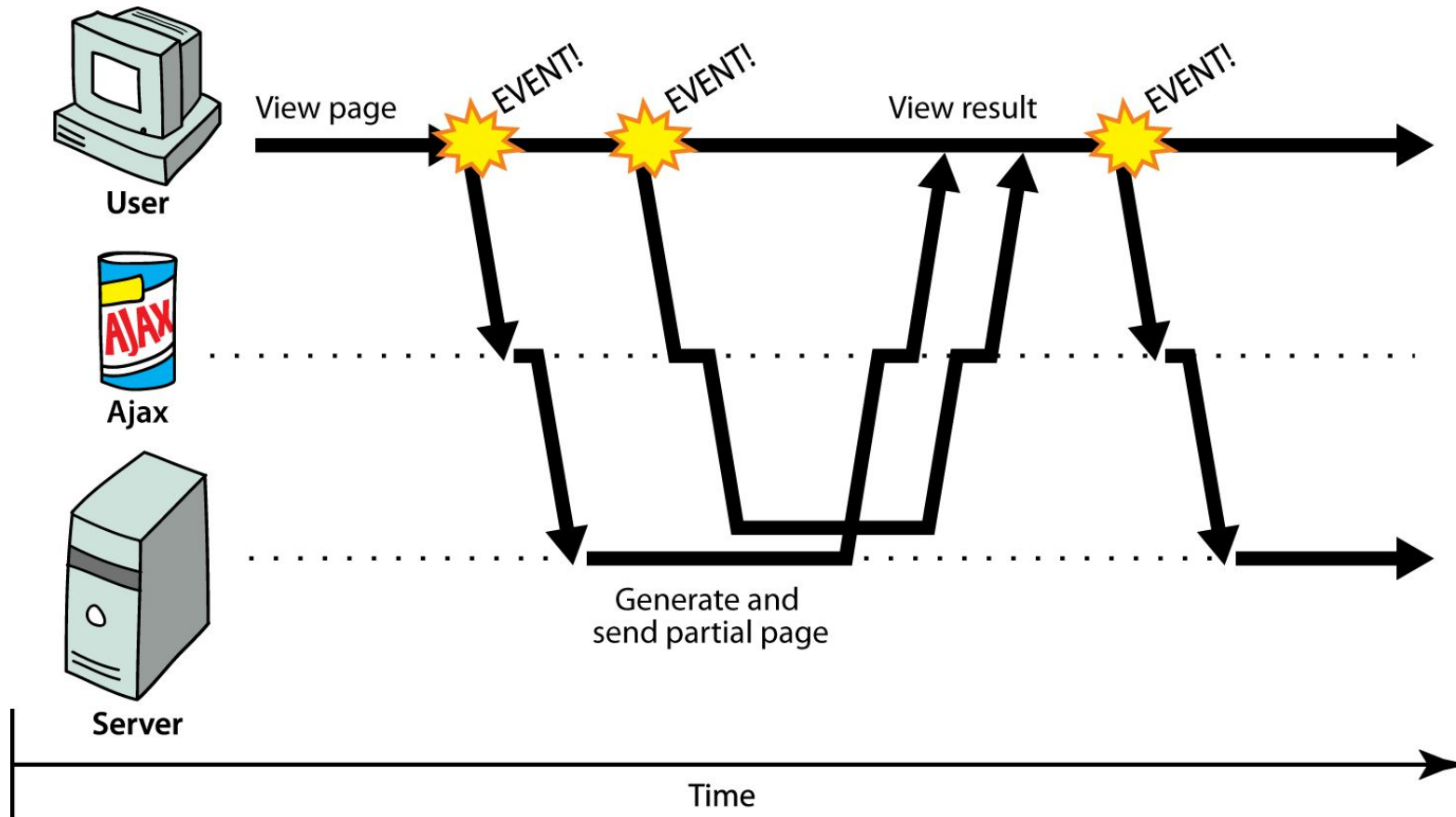
```
const jsonString = '{
  "window": {
    "title": "Sample Widget",
    "width": 500,
    "height": 500
  },
  "image": {
    "src": "images/logo.png",
    "coords": [250, 150, 350, 400],
    "alignment": "center"
  },
  "messages": [
    {"text": "Save", "offset": [10, 30]},
    {"text": "Help", "offset": [ 0, 50]},
    {"text": "Quit", "offset": [30, 10]},
  ],
  "debug": "true"
}';
const data = JSON.parse(jsonString);
```

Given the JSON data, how can we access:

- The window's title?
- The image's third coordinate?
- The number of messages?
- The y-offset of the last message?

Asynchronous Web Communication

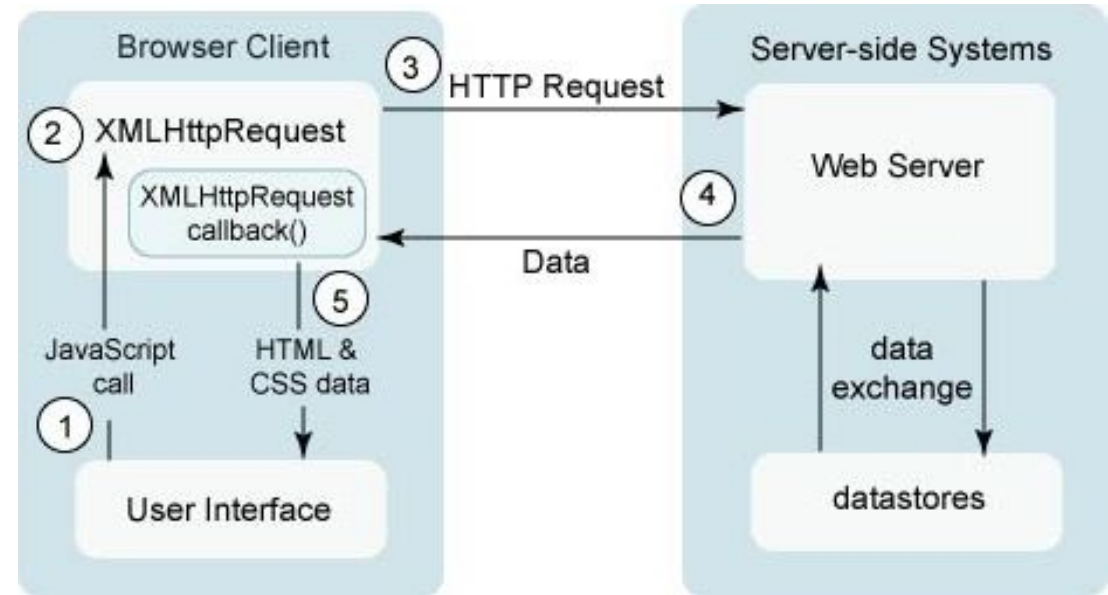
Users keep interacting with page while data loads



AJAX Calls

The browser will send an asynchronous request to the server and register a callback function to handle the response.

When servers send the response, it will be passed to the callback function and pushed into the Macrotask queue.



Fetch API

The Fetch API provides a JavaScript interface for accessing and manipulating requests and responses. It provides a global **fetch()** method that provides an easy, logical way to fetch resources asynchronously across the network.

This kind of functionality was previously achieved with the deprecated **XMLHttpRequest API**.

Using Fetch

The `fetch()` method takes one mandatory argument, the path to the resource you want to fetch. It returns a Promise that resolves to the Response to that request.

```
fetch('http://www.mum.edu/students.json')
  .then(response => response.json())
  .then(myJson => console.log(myJson));

fetch('http://www.mum.edu/students',
  { method: 'POST', // GET, POST, PUT, DELETE, etc.
    headers: {'Content-Type': 'application/json'},
    body: JSON.stringify({student: "Asaad Saad"})
  })
  .then(response => response.json());
  .catch(error => console.error(error));
```

Working with Response Body

Response is usually a stream and we need to read it to completion, it returns a promise that resolves with the results of parsing the body:

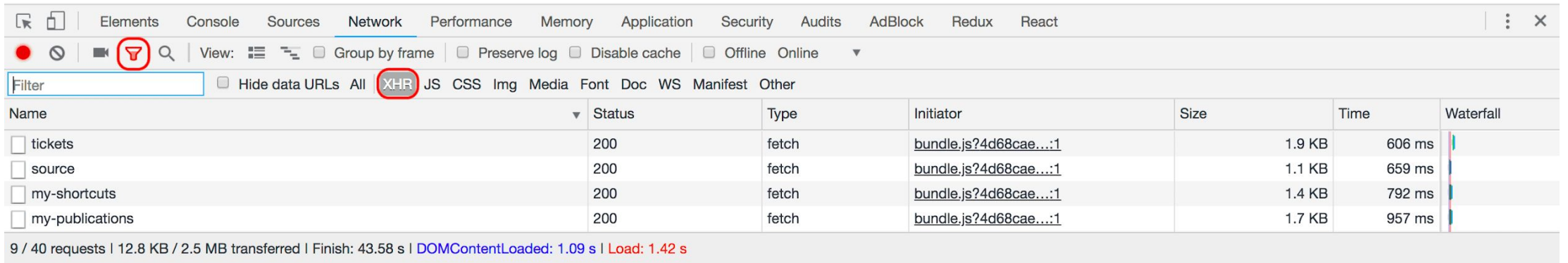
```
// when response is JSON object
await response.json()
// when response is text
await response.text()
```


Fetch API with Async/Await

```
async function fetchDataJSON() {  
  const response = await fetch('/data');  
  const data = await response.json();  
  return data;  
}
```

```
// Remember to use try/catch in case of network problems
```

Debugging AJAX Requests



The screenshot shows the Chrome DevTools Network tab. The 'Network' tab is selected in the top bar. In the toolbar, the 'Filter' icon (a funnel) is circled in red. Below the toolbar, the 'Filter' input field is active, and the 'XHR' filter is selected, also circled in red. The table below displays four filtered requests, all with a status of 200 and type of 'fetch'. The bottom status bar shows 9 / 40 requests, 12.8 KB / 2.5 MB transferred, and a finish time of 43.58 s.

Name	Status	Type	Initiator	Size	Time	Waterfall
<input type="checkbox"/> tickets	200	fetch	bundle.js?4d68cae....:1	1.9 KB	606 ms	
<input type="checkbox"/> source	200	fetch	bundle.js?4d68cae....:1	1.1 KB	659 ms	
<input type="checkbox"/> my-shortcuts	200	fetch	bundle.js?4d68cae....:1	1.4 KB	792 ms	
<input type="checkbox"/> my-publications	200	fetch	bundle.js?4d68cae....:1	1.7 KB	957 ms	

9 / 40 requests | 12.8 KB / 2.5 MB transferred | Finish: 43.58 s | DOMContentLoaded: 1.09 s | Load: 1.42 s

Single Page Applications (SPA)

Your browser sends a request to the server and receives a response with a single HTML page and JavaScript client-side application.

HTML file has a single outlet element, where All DOM creation, and upcoming requests are handled by JavaScript in the browser (including Routes). The browser sends AJAX calls behind the scene to get/post data from the server when needed.