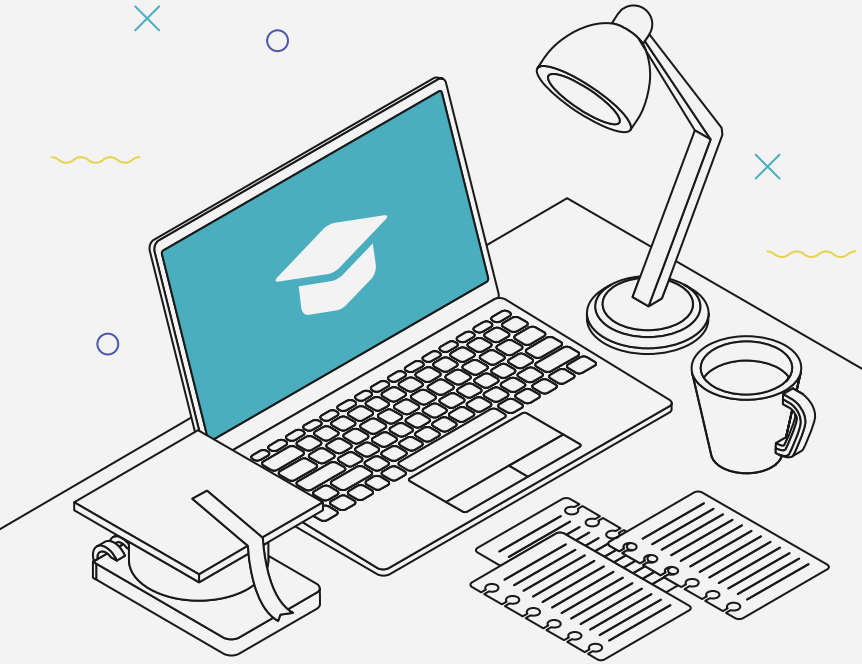
Github

https://github.com/PersoSirEduard/HackMcGill-Backend-Workshop

Backend Workshop

HackMcGill



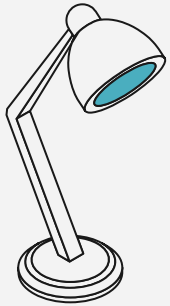
Eduard Anton

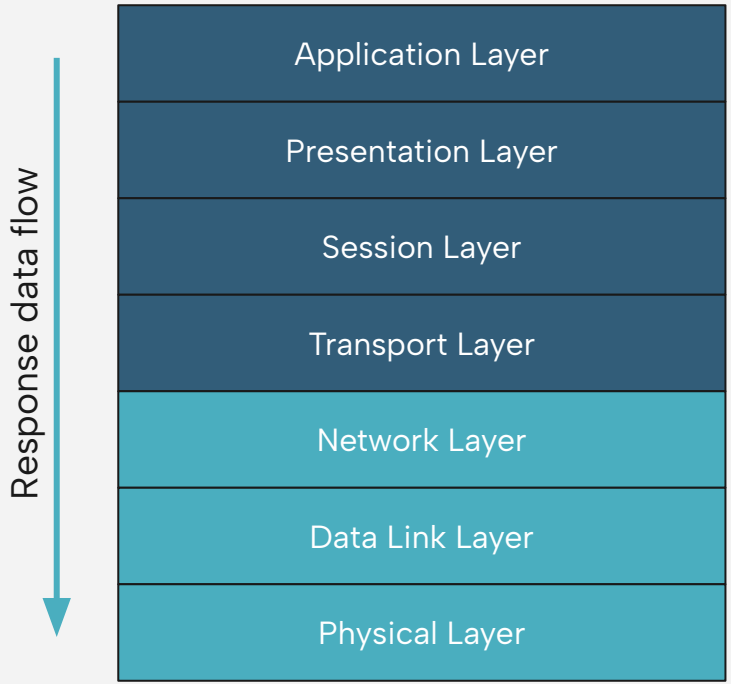
Application Programming Interface (API)

- Allows software to communicate
- Specify standards (interface)
- Software design involved
- **APIs are not necessarily provided by a remote server! (e.g. Windows API)**

Open Systems Interconnection (OSI)

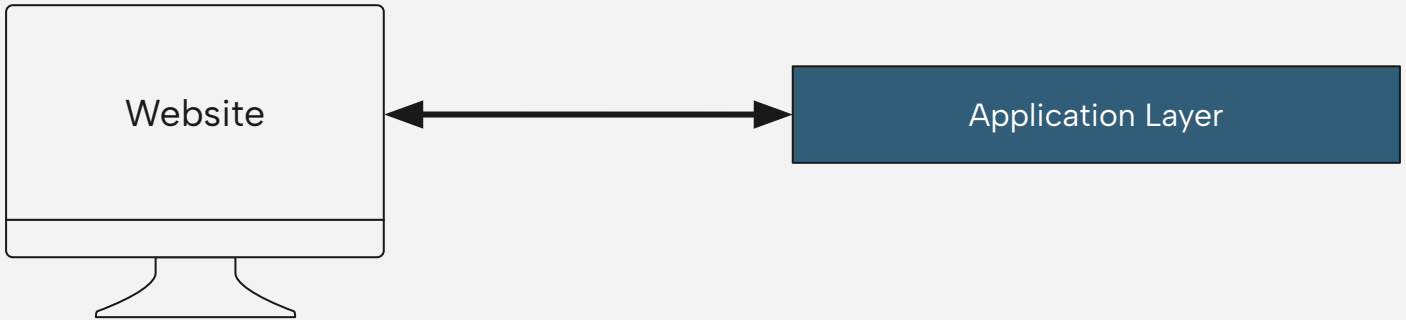
- Standardization of communication (AKA protocols)
- Communication over the network
- **Layered abstraction**





Application Layer

- Interacts with data from the user
- E.g. HTTP, SMTP, SIP, SSH, etc.



Presentation Layer

01

Translation

From text structure to bytes

02

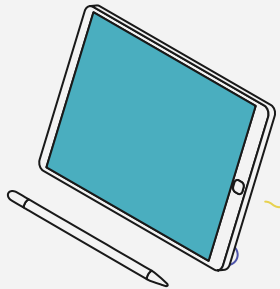
Encryption

Security

03

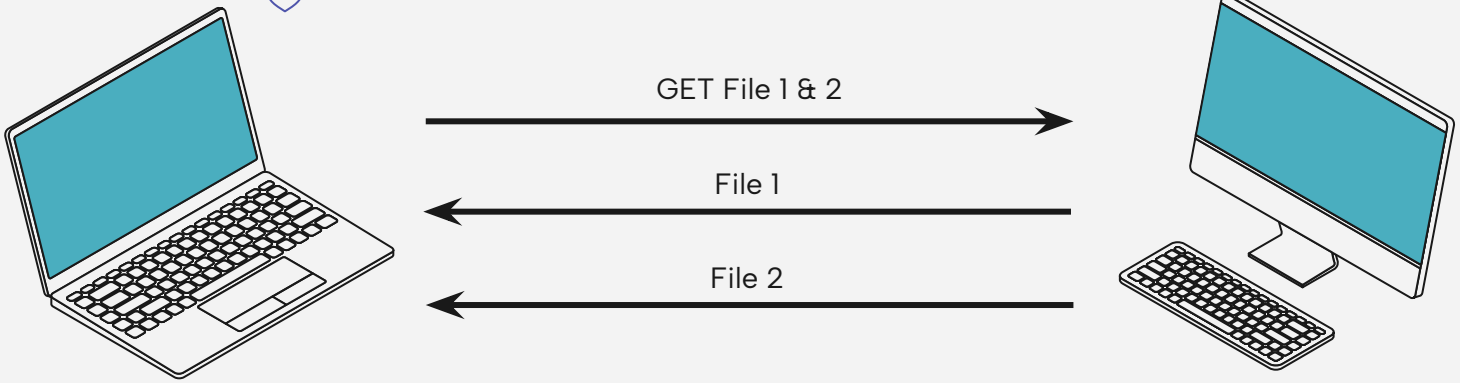
Compression

Speed and Efficiency



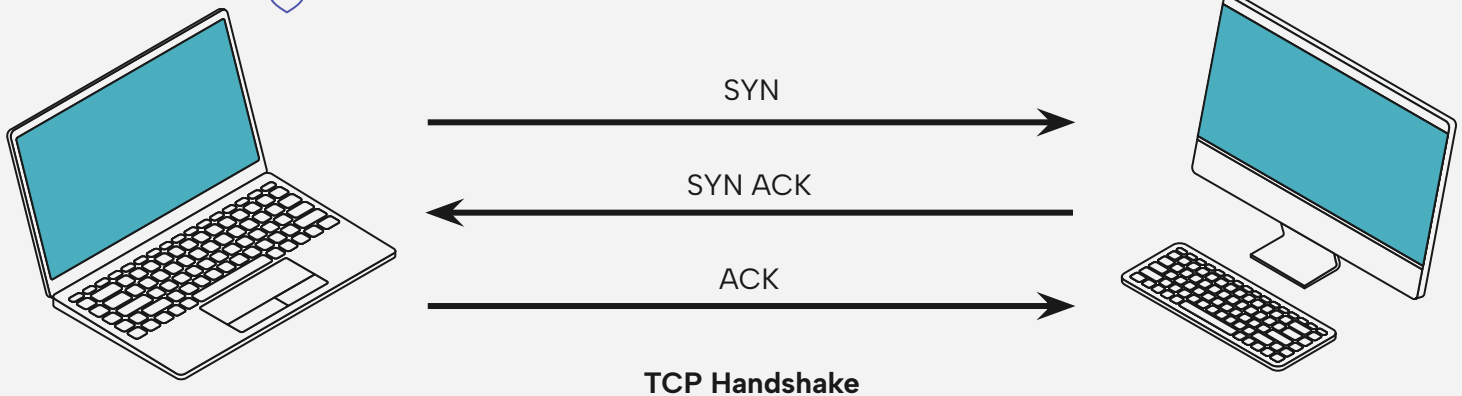
Session Layer

- Starting and closing communication
- Resource management
- Synchronizes data with checkpoints (e.g. downloading)



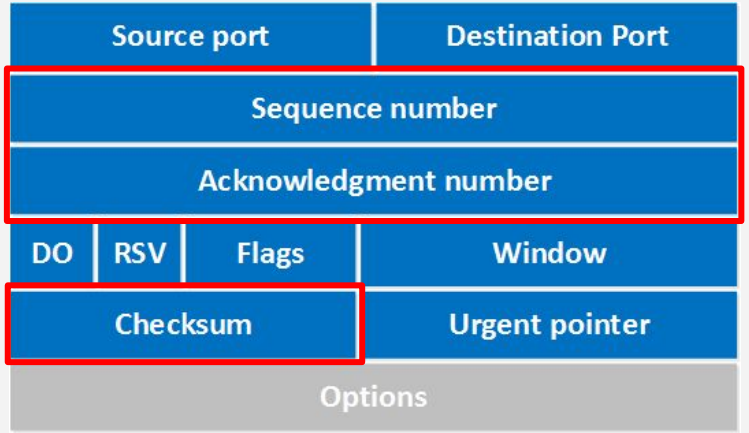
Transport Layer

- End-to-end communication
- Breaking up/reconstructing data into/from **segments**
- Flow control and error control (TCP)
- Transmission Control Protocol (TCP)
- User Datagram Protocol (UDP)



TCP Handshake

TCP Header

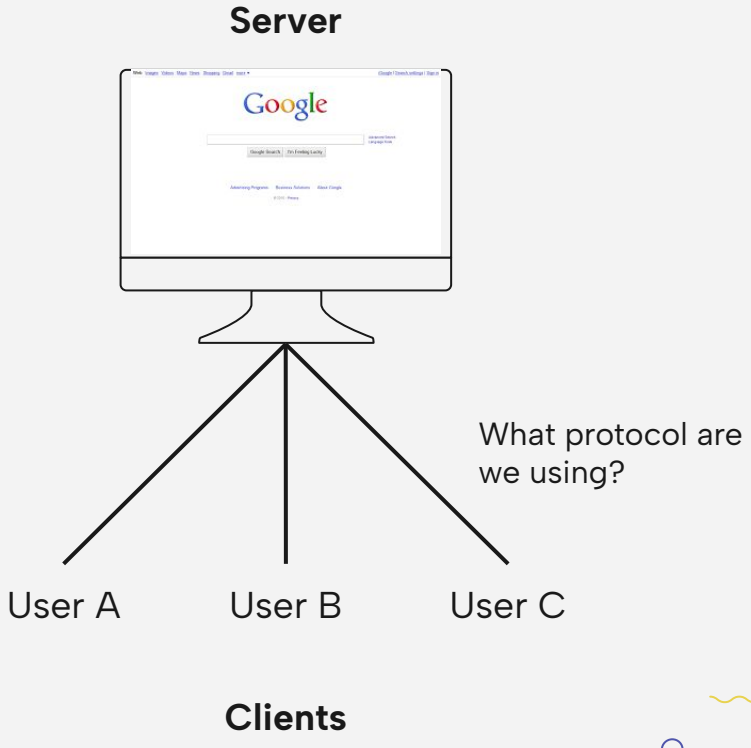


Verify order and size of segment

Error detection

Client-Server

- Client requests a “service” provided by the server
- Agreement on the API
- Centralized architecture
- Simple
- Many-to-one connections
- Careful with traffic management. There is a risk of Denial-of-service (DoS) attacks
- Other architectures: P2P



Sockets

- Communication mechanism provided by the OS

Server

sockets > socket_server.py

```python
import socket

HOST = 'localhost'
PORT = 8000

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.bind((HOST, PORT))
sock.listen(1)

conn, addr = sock.accept()
with conn:
    data = conn.recv(1024)
    if not data is None:
        print(data.decode())
        conn.sendall(b"Hello from server")
```

Client

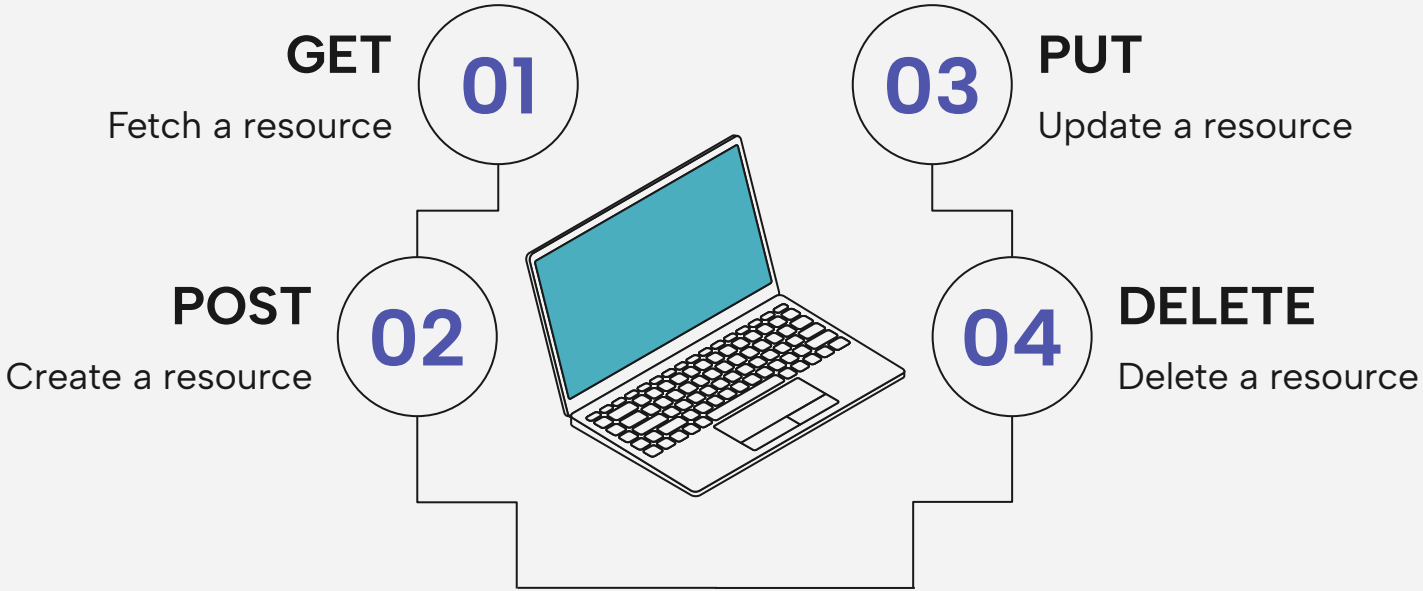
sockets > socket_client.py

```python
import socket

HOST = 'localhost'
PORT = 8000

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((HOST, PORT))

sock.send(b"Hello from client")
res = sock.recv(1024)
print(res.decode())
```

Demo


HTTP Method
Path
HTTP Version
GET / HTTP/1.1
Host: localhost:8000
Connection: keep-alive
sec-ch-ua: "Chromium";v="123", "Not:A-Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Linux"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/
signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9
HTTP Headers

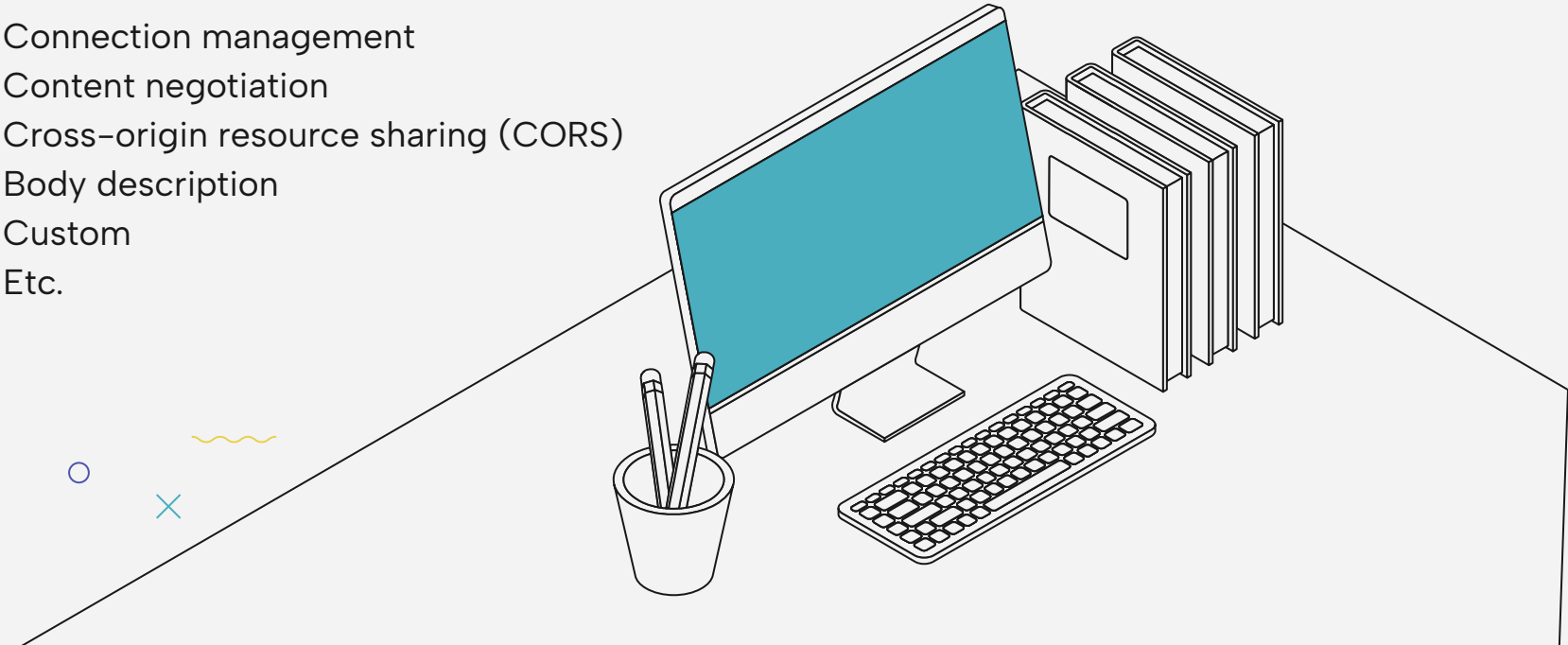
HTTP Methods



"CRUD": Create, Read, Update, Delete

HTTP Headers

- Authentication
- Caching
- Conditionals
- Connection management
- Content negotiation
- Cross-origin resource sharing (CORS)
- Body description
- Custom
- Etc.



HTTP Versions

1.0

- First version, introduced in 1996
- Stateless (no sessions)

1.1

- Improved performance and security
- Persistent connection
- Caching

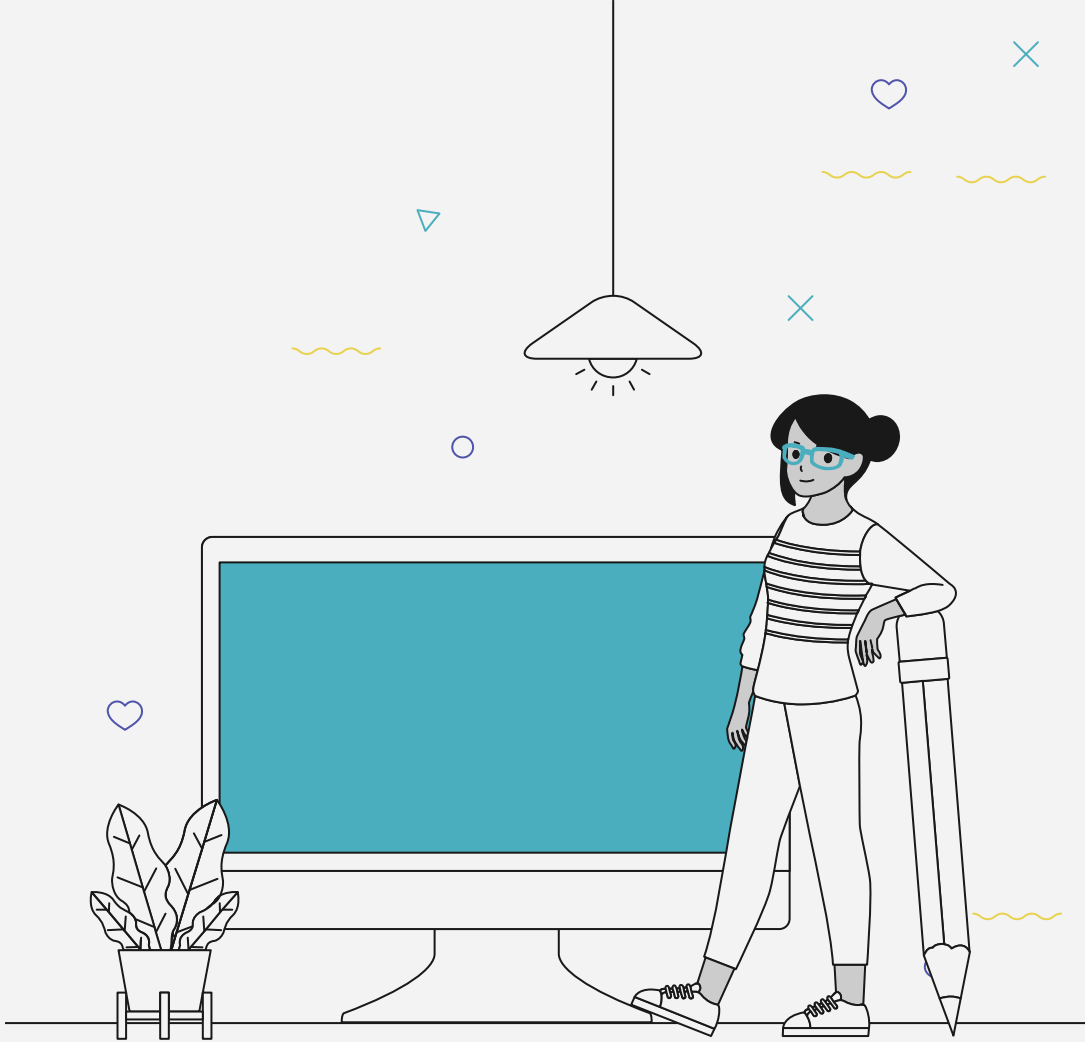
1.2

- Pipelining (send multiple requests before a response)
- Server push (proactive server)
- Header compression

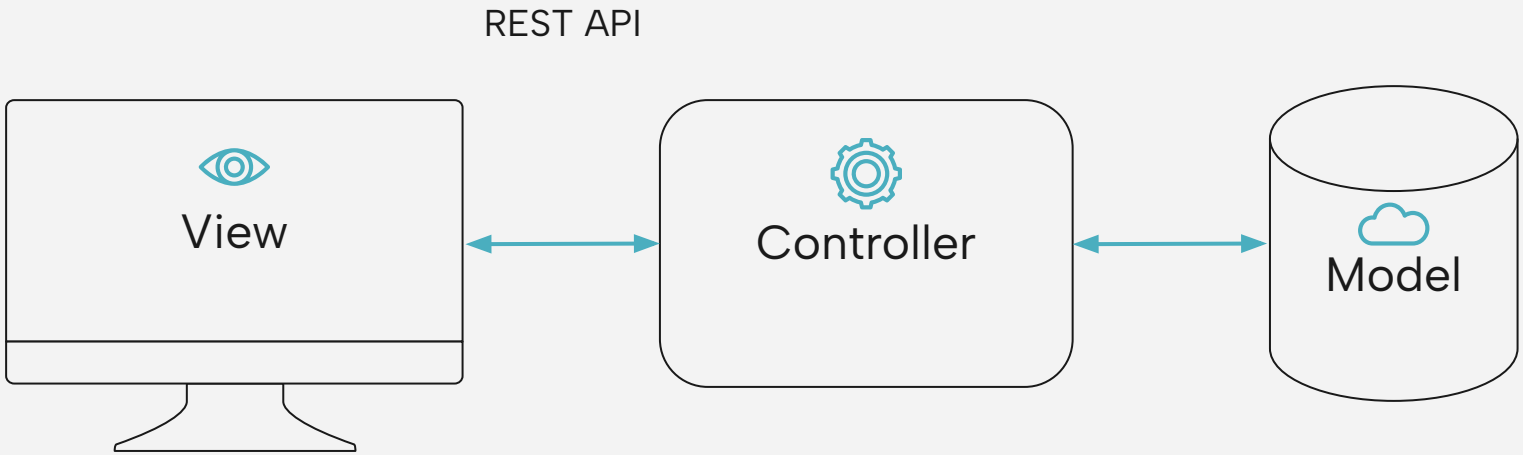
2.0

- Prioritization
- Streams
- Compression

DEMO



Model-View-Controller (MVC)



Fetch API

```
// Specify the API endpoint for user data
const apiUrl = 'https://api.example.com/users/123';

// Make a GET request using the Fetch API
fetch(apiUrl)
  .then(response => {
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    return response.json();
  })
  .then(userData => {
    // Process the retrieved user data
    console.log('User Data:', userData);
  })
  .catch(error => {
    console.error('Error:', error);
  });
```

```
// Specify the API endpoint for user data
const apiUrl = 'https://api.example.com/users/123';

try {
    // Make a GET request using the Fetch API
    const response = await fetch(apiUrl)
    const userData = await response.json()

    // Process the retrieved user data
    console.log('User Data:', userData)
} catch (error) {
    console.error('Error:', error);
}
```

Thanks!

Do you have any questions?