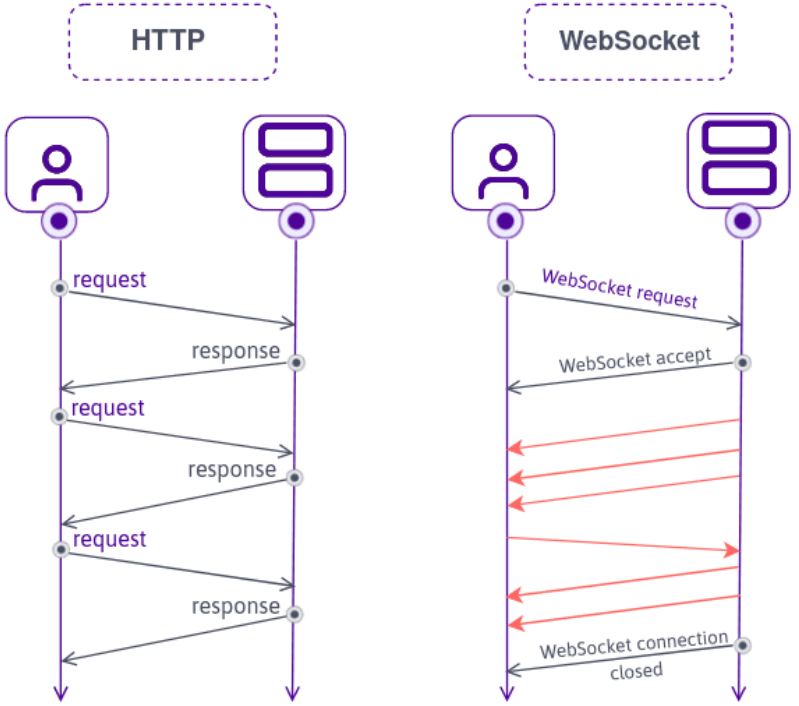
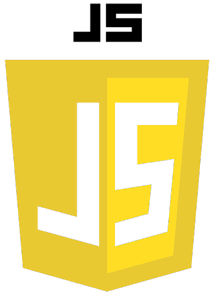
WEB SOCKET

Une WebSocket est un protocole de communication bidirectionnel en temps réel entre un client et un serveur. Contrairement aux protocoles de communication traditionnels tels que HTTP, qui sont de nature unidirectionnelle, WebSocket permet une communication en temps réel en permettant une connexion persistante entre un client et un serveur.



Les connexions WebSocket sont initiées par une demande d'ouverture de connexion, généralement envoyée par le client. Si le serveur accepte la demande, une connexion persistante est établie entre le client et le serveur, permettant aux deux parties de s'échanger des données en temps réel.

L'utilisation de WebSocket permet une expérience utilisateur plus réactive et plus interactive, en permettant des mises à jour en temps réel sans avoir à recharger constamment la page Web.



SOCKET.IO

Socket.IO est une bibliothèque JavaScript pour la création d'applications Web en temps réel. Elle offre une couche d'abstraction au-dessus du protocole WebSocket natif en fournissant des fonctionnalités supplémentaires telles que la gestion des connexions, la gestion des salons de discussion, la diffusion de messages et la prise en charge de la rétrocompatibilité avec les navigateurs plus anciens.

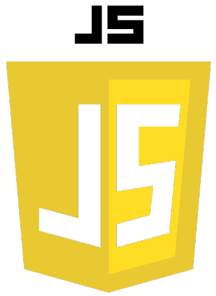
```
const server = require('http').createServer();
const io = require('socket.io')(server);

io.on('connection', (socket) => {
  console.log('Un client est connecté');
});

server.listen(3000, () => {
  console.log('Le serveur est en cours
d\'exécution sur le port 3000');
});
```

CÔTÉ SERVEUR

La fonction io.on('connection') écoute les connexions entrantes et exécute une fonction de rappel chaque fois qu'un client se connecte.



SOCKET.IO

Dans votre fichier HTML, vous devez inclure la bibliothèque Socket.IO en ajoutant la balise <script> suivante :

```
<script src="/socket.io/socket.io.js"></script>
```

```
const socket = io();
```

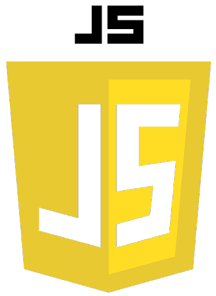
Cette commande crée un objet de socket qui permet la communication bidirectionnelle entre le client et le serveur. En utilisant la méthode socket.on() pour écouter un événement nommé "nouveau-message" :

```
socket.on('nouveau-message', (message) => {
  console.log('Nouveau message reçu : ', message);
});
```

CÔTÉ CLIENT

Étape 5: Émettre des événements côté client Vous pouvez émettre des événements côté client en utilisant la méthode socket.emit(). Par exemple, pour envoyer un message au serveur, vous pouvez utiliser le code suivant :

```
socket.emit('envoyer-message', 'Bonjour, comment ça va ?');
```



SOCKET.IO

Voici un exemple de code Arduino utilisant la bibliothèque "Socket.IO Client for Arduino" pour se connecter à un serveur Socket.IO (nodeJs) et envoyer un message :

```
#include <SPI.h>
#include <Ethernet.h>
#include <SocketIOClient.h>

EthernetClient ethClient;
SocketIOClient socketIO;

void setup() {
  Ethernet.begin(mac);  // Configure l'interface Ethernet
  socketIO.begin("http://exemple.com:3000");  // Configure le serveur Socket.IO
  socketIO.on("connect", []() {  // Écoute l'événement "connect"
    Serial.println("Connecté à Socket.IO !");
    socketIO.emit("message", "Hello, Socket.IO !");  // Envoie un message au serveur
  });
}

void loop() {
  socketIO.loop();  // Gère les événements Socket.IO
}
```

CÔTÉ CLIENT ARDUINO