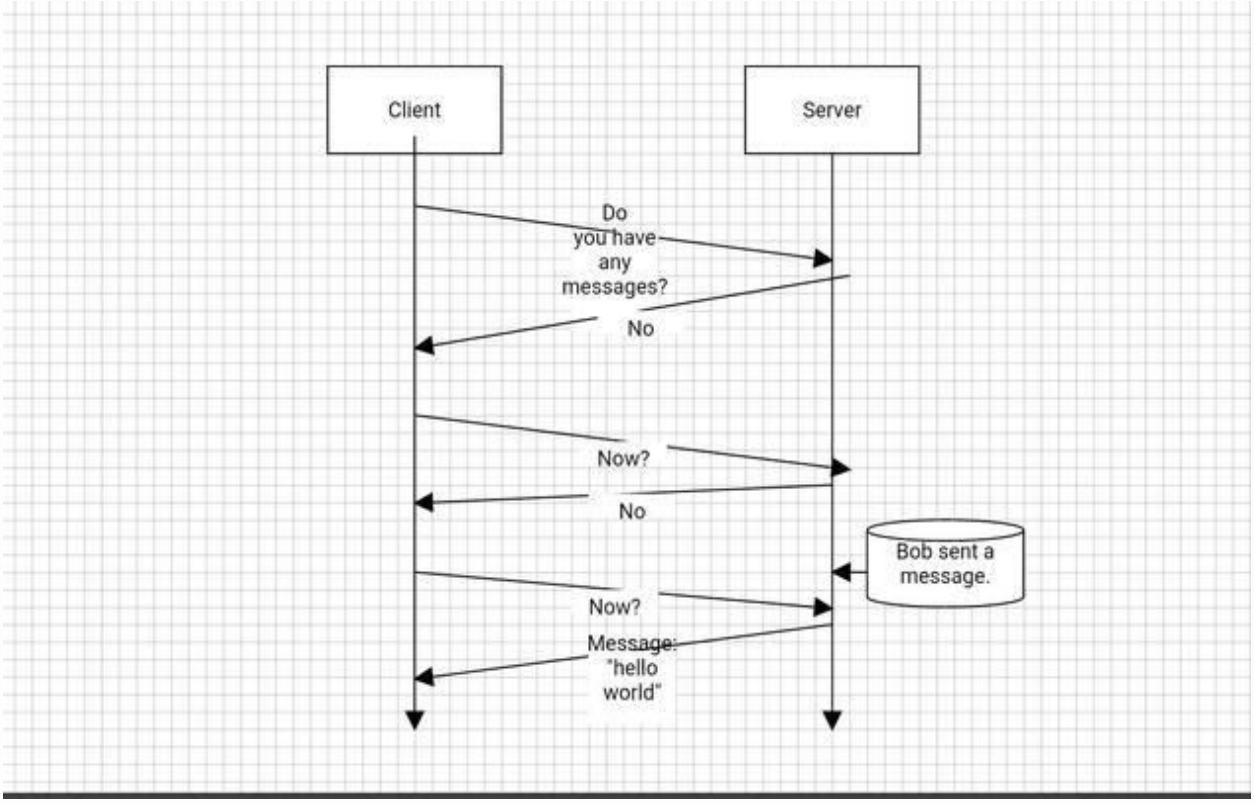
WebSocket

“WebSockets” is an advanced technology that allows real-time interactive communication between the client browser and a server. It uses a completely different protocol that allows **bidirectional data flow,** making it unique against HTTP.

WebSockets allow both the server and the client to push messages at any time without any relation to a previous request. One notable advantage in using websockets is, almost every browser support websockets.

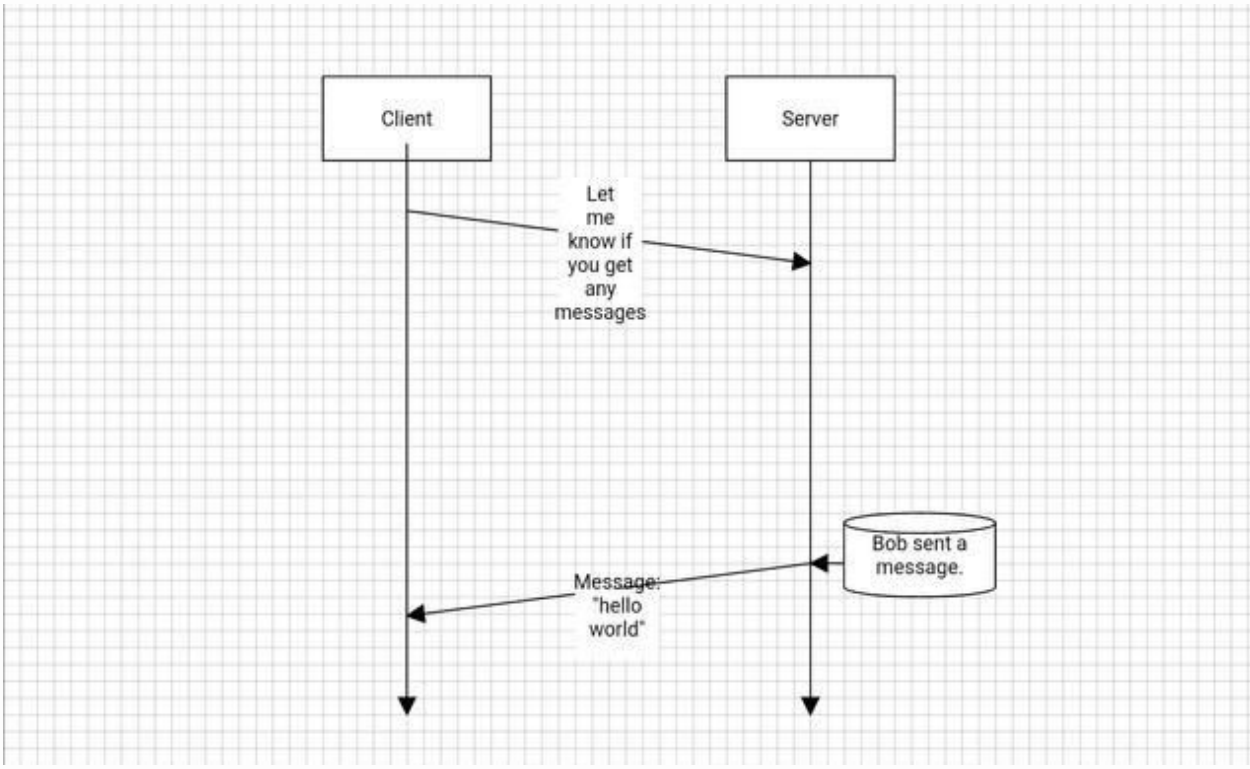
How does HTTP works?



If you are a developer, you probably know what HTTP **(or HTTPS)** is. It is seen every day, in your browser. A request is needed to respond; **you to constantly ask the server if there are new messages in order to receive them.**

You should also know that HTTP allows you to have different types of requests such as post, get or put, each with a different purpose.

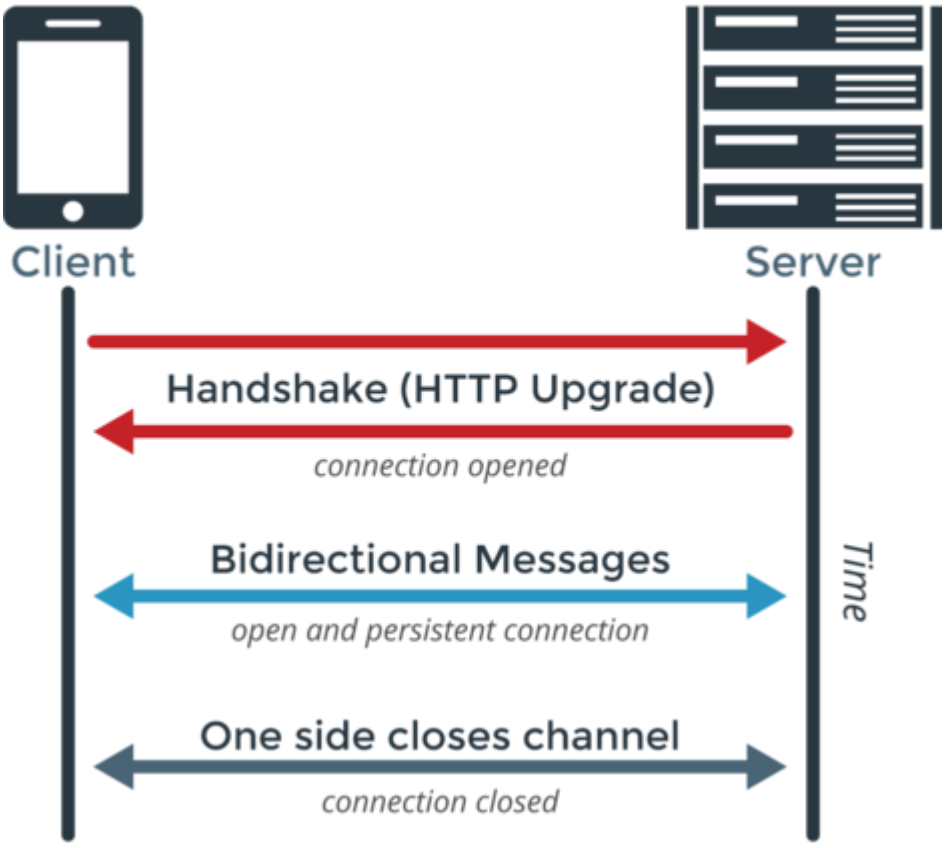
Websocket:



WebSocket solves a few issues with HTTP:

- **Bi-directional** protocol—either client/server can send a message to the other party (In HTTP, the request is always initiated by client and the response is processed by server—making HTTP a uni-directional protocol)
- **Full-duplex** communication—client and server can talk to each other independently at the same time

- Single TCP connection—After upgrading the HTTP connection in the beginning, client and server communicate over that same TCP connection throughout the lifecycle of Websocket connection



Web Sockets on the other hand don't need you to send a request in order to respond. They allow bidirectional data flow so you just have to listen for any data.

You can just listen to the server and it will send you a message when it's available. Seems more practical, right? And it is…

What is the web socket good for?

- Real-time applications
- Chat apps
- IoT (internet of things)
- Online multiplayer games

It is necessary to initialize the connection to the server from client for communication between them. For initializing the connection, creation of Javascript object with the URL with the remote or local server is required.

```
var socket = new WebSocket(“ ws://echo.websocket.org ”);
```

The URL mentioned above is a public address that can be used for testing and experiments. The websocket.org server is always up and when it receives the message and sends it back to the client.

Web Sockets – Events

There are four main Web Socket API events −

- Open
- Message
- Close
- Error

Each of the events are handled by implementing the functions like onopen, onmessage, onclose and onerror functions respectively. It can also be implemented with the help of addEventListener method.

The brief overview of the events and functions are described as follows −

Open

Once the connection has been established between the client and the server, the open event is fired from Web Socket instance. It is called as the initial handshake between client and server. The event, which is raised once the connection is established, is called onopen.

Message

Message event happens usually when the server sends some data. Messages sent by the server to the client can include plain text messages, binary data or images. Whenever the data is sent, the onmessage function is fired.

Close

Close event marks the end of the communication between server and the client. Closing the connection is possible with the help of onclose event. After marking the end of communication with the help of onclose event, no messages can be further transferred between the server and the client. Closing the event can happen due to poor connectivity as well.

Error

Error marks for some mistake, which happens during the communication. It is marked with the help of onerror event. Onerror is always followed by termination of connection. The detailed description of each and every event is discussed in further chapters.

Web Sockets – Actions

Events are usually triggered when something happens. On the other hand, actions are taken when a user wants something to happen. Actions are made by explicit calls using functions by users.

The Web Socket protocol supports two main actions, namely −

- send( )
- close( )

Example:

```
import * as apis from '../middleware/api'
```

```
import {store} from '../store/store'
import * as actions from '../actions/action'

export const init = (handshakeUrl, path = '/ws') => {
 const socket = new WebSocket(handshakeUrl + path)
 // Show a connected message when the WebSocket is opened.

socket.onopen = function (event) {
  console.log('Chat Server Successfully connected')
  sessionStorage.setItem('socketInfo', socket)
  store.dispatch(actions.persistSocket(socket))
  // Dispatch required home page request
  store.dispatch(apis.initialiseHomePageRequests(socket))
 }

 // Handle messages sent by the server.
 socket.onmessage = function (event) {
  let data = event.data
  if (data === 'ping') {
    // Sending the pong message for every ping message
    socket.send('pong')
  } else {
    let wsResponse = JSON.parse(event.data)
    console.log(wsResponse)
    store.dispatch(apis.processWebSocketResponse(wsResponse))
  }
 }
```

```
 // Handle any errors that occur.
 socket.onerror = function (error) {
   console.log('WebSocket Error: ' + error)
 }
}

export const socketClose = (socket) => {
 // Show a disconnected message when the WebSocket is closed.
 socket.onclose = function (event) {
   console.log('Server is disconnected')
   store.dispatch(actions.resetSocketInfo())
 }
}

export const socketSend = (socket, requestData) => {
 socket.send(JSON.stringify(requestData))
}
```
