



06

# Free Online Course

## Web Application Programming **101**

# Basic



HTML



CSS



JS

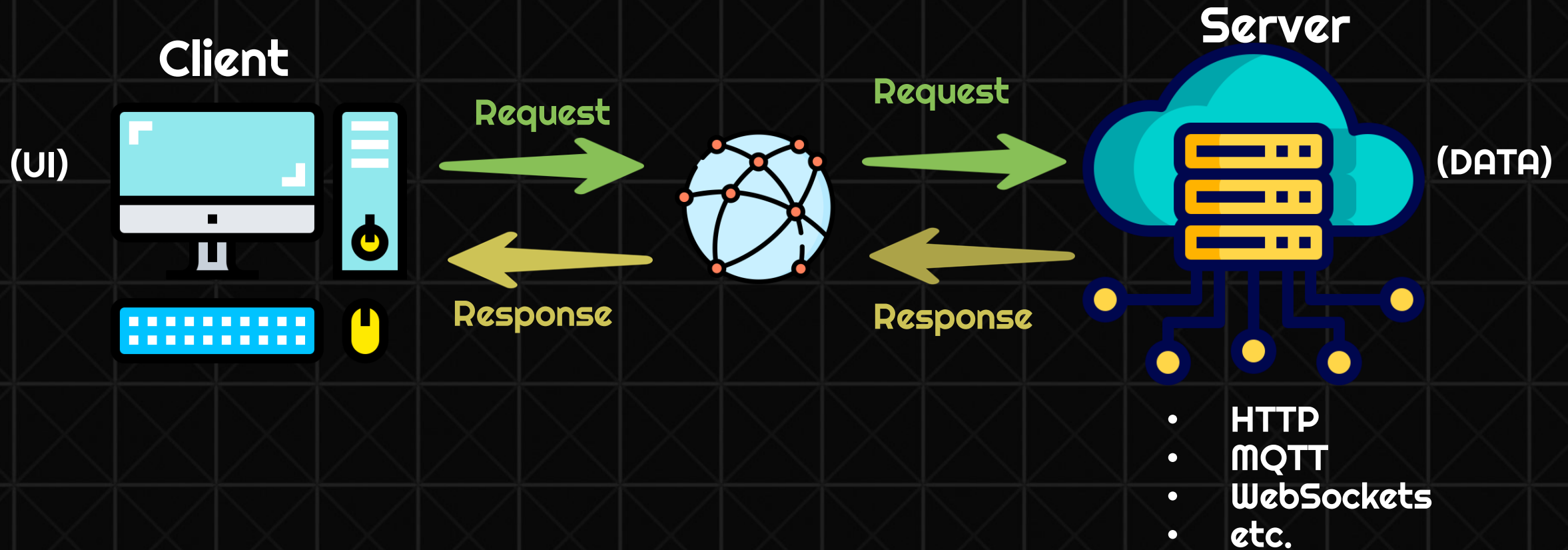




1. Client and Sever Operations
2. HTTP, MQTT, and WebSocket Protocols/Servers
3. Node.js Installation
4. WebSockets Server Development
5. WebSocket Client Development
6. UI and Real-time Data Exchange

Programming Practice

# 1. Client and Server Operations





## 2. Protocols (HTTP, MQTT, WebSockets)



### HTTP

The Hypertext Transfer Protocol (HTTP) is the foundation of the World Wide Web, and is used to load web pages using hypertext links. HTTP is an application layer protocol designed to transfer information between networked devices and runs on top of other layers of the network protocol stack. A typical flow over HTTP involves a client machine making a request to a server, which then sends a response message.



MQTT

### MQTT

MQTT (MQ Telemetry Transport) is a lightweight open messaging protocol that provides resource-constrained network clients with a simple way to distribute telemetry information in low-bandwidth environments. The protocol, which employs a publish/subscribe communication pattern, is used for machine-to-machine (M2M) communication.



### WebSockets

WebSocket is a bidirectional communication protocol that can send the data from the client to the server or from the server to the client by reusing the established connection channel. The connection is kept alive until terminated by either the client or the server. Almost all the real-time applications like (trading, monitoring, notification) services use WebSocket to receive the data on a single communication channel.

# 3. Node.js Installation



<https://nodejs.org/>

The screenshot shows the Node.js website with the following elements:

- Navigation Bar:** Includes the Node.js logo, a navigation menu with links to HOME, ABOUT, DOWNLOADS, DOCS, GET INVOLVED, SECURITY, CERTIFICATION, and NEWS, and a dark theme toggle.
- Instructions:** Two yellow boxes highlight the steps: "1) Download and Install Node.js" and "2) Check version of Node.js".
- Text:** "Node.js® is an open-source, cross-platform JavaScript runtime environment."
- Download Section:** Titled "Download for Windows (x64)", it features two green buttons:
  - 16.17.1 LTS:** Labeled "Recommended For Most Users".
  - 18.10.0 Current:** Labeled "Latest Features".
- Links:** Below the buttons are links for "Other Downloads", "Changelog", and "API Docs" for both versions.
- Information:** A line of text states: "For information about supported releases, see the [release schedule](#)."
- Footer:** A copyright notice for the OpenJS Foundation and Node.js contributors.

# 4. WebSockets Server Development



**Install WebSockets dependency**



**Develop/Write WebSockets Server**



**Run/Start WebSockets Server**



# 4. WebSockets Server Development

ws\_server.js (1/3)



```
const WebSocketServer = require('websocket').server;
const http = require('http');

const server = http.createServer(function (request, response) {
  console.log((new Date()) + ' Received request for ' + request.url);
  response.writeHead(404);
  response.end();
});
server.listen(9099, function () {
  console.log((new Date()) + ' Server is listening on port 9099');
});

let wsServer = new WebSocketServer({
  httpServer: server,
  autoAcceptConnections: false
});

function originIsAllowed(origin) {
  return true;
}
```

# 4. WebSockets Server Development

ws\_server.js (2/3)



```
wsServer.on('request', function (request) {
    if (!originIsAllowed(request.origin)) {
        request.reject();
        console.log((new Date()) + ' Connection from origin ' + request.origin + ' rejected.');
```

```
        return;
    }

    let connection = request.accept('realtime', request.origin);

    console.log((new Date()) + ' Connection accepted.');
```

```
    connection.on('message', function (message) {
        if (message.type === 'utf8') {
            processAndRespond(connection, message.utf8Data);
        }
        else if (message.type === 'binary') {
            console.log('Received Binary Message of ' + message.binaryData.length + ' bytes');
```

```
            connection.sendBytes(message.binaryData);
        }
    });
    connection.on('close', function (reasonCode, description) {
        console.log((new Date()) + ' Peer ' + connection.remoteAddress + ' disconnected.');
```

```
    });
});
```



# 4. WebSockets Server Development

ws\_server.js (3/3)



```
const processAndRespond = (con, msg) => {
  const generate = (type) => {
    let x = Math.random();
    if (type === 1) {
      return x > 0.5;
    }
    else if (type === 2) {
      return (Math.random()).toFixed(3);
    }
    else if (type === 3) {
      return (Math.random() * 100).toFixed(3);
    }
  }
  const response = (conn, data) => {
    console.log(`Response: ${data}`);
    conn.sendUTF(data);
  }
  const generateAndResponse = (conn, type) => {
    response(conn, generate(type));
  }
  console.log(`Request: ${msg}`);
  if (msg.indexOf("get-sensor-") === 0) {
    try {
      generateAndResponse(con, Number(msg.substring(msg.length - 1)));
    } catch (ex) {
      console.error(ex);
    }
  }
}
```

# 5. WebSocket Client Development



```
<html>
<head>
  <title>WebSocket Client</title>
</head>
<body>
  <script>
    const ws = new WebSocket("ws://127.0.0.1:9099", "realtime");
    ws.onopen = function (event) {
      console.log("Connected")
    }

    ws.onmessage = function (message, x) {
      console.log(message.data);
    }

    /**
     * CODE CODE CODE
     */
  </script>
</body>
</html>
```



# 6. UI and Real-time Data Exchange

[Index.html](#)





## Let's Code

