🔥 NEXT (Tell me ONE)

Reply with **any one number**:

1️⃣ Event Loop PDF (same underline style)
2️⃣ Promises + Async/Await PDF
3️⃣ HTTP Module PDF
4️⃣ Full Node.js Notes (ALL topics combined)

I'll generate it **in the exact same premium style**, bro 💯

Generate HTTP MODULE notes Completely bro

🔥 NODE.JS HTTP MODULE — COMPLETE NOTES (BEGINNER → ADVANCED)

(Industry-level • Clean theory + practical understanding • Interview ready)

🧠 WHAT IS THE `http` MODULE?

The `http` **module** is a **core Node.js module** used to:

- Create **web servers**
- Handle **HTTP requests**
- Send **HTTP responses**
- Build APIs **without any framework**

📌 Frameworks like **Express, Fastify, NestJS** are all built **on top of** `http`.

```js
const http = require("http");
```

Copy code

🌐 HOW HTTP WORKS (FOUNDATION)

Client–Server Architecture

```yaml
```

```
Client (Browser / App)
        |
        |  HTTP Request
        ↓
Node.js Server (http)
        |
        |  HTTP Response
        ↓
Client
```

HTTP Characteristics

- Stateless
- Request–Response based
- Text-based protocol
- Runs over TCP

🚀 CREATING A BASIC HTTP SERVER

```js
const http = require("http");

const server = http.createServer((req, res) => {
  res.end("Hello from HTTP server");
});

server.listen(3000, () => {
  console.log("Server running on port 3000");
});
```

🧠 `createServer()` internally creates a **TCP server** and listens for requests.

📥 REQUEST OBJECT (`req`) — COMPLETE

The `req` object represents the **incoming request** from client.

Common Properties

```js
```

```
req.method    // GET, POST, PUT, DELETE
req.url       // /api/users
req.headers   // request headers
```

Example:

```js
console.log(req.method, req.url);
```

🔥 READING REQUEST BODY (POST / PUT)

Request body comes as **streams (chunks)**.

```js
let body = "";

req.on("data", chunk => {
  body += chunk;
});

req.on("end", () => {
  console.log(body);
});
```

📌 This is how **body-parser** works internally.

📤 RESPONSE OBJECT (`res`) — COMPLETE

The `res` object is used to **send data back** to client.

✅ Setting Status Code

```js
res.statusCode = 200;
```

or

```js
```

```
res.writeHead(200);
```

Copy code

✅ Setting Headers

js

```js
res.setHeader("Content-Type", "text/plain");
```

Copy code

Common headers:

- Content-Type
- Content-Length
- Authorization
- Set-Cookie

✅ Sending Response

js

```js
res.write("Hello");
res.end(" World");
```

Copy code

📌 `res.end()` **must be called**, otherwise request hangs.

📦 SENDING JSON (API RESPONSE)

js

```js
res.writeHead(200, {
  "Content-Type": "application/json"
});

res.end(JSON.stringify({
  success: true,
  message: "API working"
}));
```

📌 This is the base of REST APIs.

Copy code

🛣️ ROUTING USING HTTP MODULE

Manual routing using `req.url` + `req.method`.

```js
if (req.url === "/" && req.method === "GET") {
  res.end("Home Page");
}
else if (req.url === "/about") {
  res.end("About Page");
}
else {
  res.writeHead(404);
  res.end("Not Found");
}
```

🧠 Express automates this logic.

Copy code

🔍 QUERY PARAMETERS

URL:

```bash
/search?item=pizza&price=200
```

Copy code

Parse query params:

```js
const url = new URL(req.url, `http://${req.headers.host}`);

console.log(url.searchParams.get("item"));
```

Copy code

📄 SERVING HTML FILES

```js
const fs = require("fs");

const server = http.createServer((req, res) => {
  const html = fs.readFileSync("index.html");
  res.writeHead(200, { "Content-Type": "text/html" });
  res.end(html);
});
```

Copy code

🍪 COOKIES & HEADERS

Set Cookie

js

```js
res.setHeader("Set-Cookie", "token=abc123");
```

Copy code

Read Cookie

js

```js
req.headers.cookie;
```

Copy code

🔐 HTTP METHODS (INTERVIEW MUST)

Method	Purpose
GET	Fetch data
POST	Create data
PUT	Replace resource
PATCH	Partial update
DELETE	Remove resource

🚦 HTTP STATUS CODES (IMPORTANT)

Code	Meaning
200	OK
201	Created

Code	Meaning
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not Found
500	Server Error

⚙️ STREAMS IN HTTP (ADVANCED)

HTTP uses **streams**, not full data loading.

```js
req.pipe(res);
```

Copy code

Used for:

- File downloads
- Video streaming
- Large data transfer

⏱️ ERROR HANDLING & TIMEOUTS

```js
server.on("clientError", (err, socket) => {
  socket.end("HTTP/1.1 400 Bad Request");
});
```

Copy code

🔐 HTTP VS HTTPS

HTTP	HTTPS
Plain text	Encrypted
Less secure	Secure

HTTP	HTTPS
Port 80	Port 443

HTTPS uses:

```js
const https = require("https");
```

Copy code

🏗️ MINI API USING PURE HTTP

```js
const http = require("http");

http.createServer((req, res) => {
  if (req.url === "/api" && req.method === "GET") {
    res.writeHead(200, { "Content-Type": "application/json" });
    res.end(JSON.stringify({ status: "OK" }));
  }
}).listen(3000);
```

Copy code

🧠 INTERNAL FLOW (ADVANCED)

1. TCP connection created
2. HTTP parser reads request
3. `request` event emitted
4. Callback executed
5. Response streamed
6. Connection closed / reused

⚠️ LIMITATIONS OF `http` MODULE

- Manual routing
- Manual parsing
- No middleware
- Boilerplate code