# Research, Learning, and Explain it (Node.js)

1. const http = require('http');
2. const url = require('url');
3. const fs = require('fs');
4. const path = require('path');
6. const server = http.createServer((req, res) => {
7. const parsedUrl = url.parse(req.url, true);
8. const pathname = parsedUrl.pathname;
10. let filePath = path.join(\_\_dirname, 'public', pathname === '/' ? 'index.html' : pathname + '.html');
12. fs.readFile(filePath, (err, data) => {
13. if (err) {
14. if (err.code === 'ENOENT') {
15. fs.readFile(path.join(\_\_dirname, 'public', '404.html'), (err, data) => {
16. res.writeHead(404, { 'Content-Type': 'text/html' });
17. res.end(data, 'utf8');
18. });
19. } else {
20. res.writeHead(500);
21. res.end(`Server Error: ${err.code}`);
22. }
23. } else {
24. res.writeHead(200, { 'Content-Type': 'text/html' });
25. res.end(data, 'utf8');
26. }
27. });
28. });
30. const PORT = 5000;
31. server.listen(PORT, () => {
32. console.log(`Server running on port ${PORT}`);
33. });

# Write your summary for designated lines below

|  |  |
| --- | --- |
| **Lines 1-4** | Equivalent to import statements. Bring the necessary modules into the file. |
| **Line 6** | Instantiates server object |
| **Line 7** | Uses the url library to parse the req.url, enabling data passed in the URL to be accessed like that of an object through parsedUrl |
| **Line 8** | Extracts the pathname property from the parsedUrl object, which was created by parsing the URL with the url library. Used to determine which page the user is attempting to access. |
| **Line 10** | Uses the path library to construct the filepath that the user intended to access. |
| **Line 12** | Uses the fs library readfile method to pull the file for the page that the user is attempting to access. The filepath argument determines the page. |
| **Line 13** | If statement that handles the case when the readfile method fails to retrieve the file for the page that the user is trying to access, for whatever reason |
| **Line 14** | If statement that handles the case where the error in question is of type “error no entry”. This means that the page specified by the user does not have a corresponding file or the user mistyped the page they were trying to access. |
| **Line 15** | Retrieves the file for the “404 Not Found” page, corresponding to the error case of “error no entry”. Lines 16-17 run as a callback function with arguments err and data. Data contains the contents of the 404 error page, in this case. |
| **Line 16** | Defines the content type (html) and error code (404) to be passed in the response header. |
| **Line 17** | Formally ends the response and sends it back to the client with utf-8 encoding and the contents of the 404 file contained in the “data” object. |
| **Line 18** | The “}” ends the callback function; the “)” marks the closing parenthesis of the fs.readfile method; the “;” ends the “line” of code that starts with fs.readfile. The if statement is closed with a } in the following line. |
| **Line 20-21** | We are still within the block that considers the case when the file specified by the user cannot be accessed and an error is encountered. Here, however, is the error case that is NOT “error no entry”. The program assumes that any non-404 error is a server error, so the response code is set as 500 in line 20 and returned to the user as an exception in line 21. |
| **Line 23-25** | Block contains the case where NO error is encountered, and the user’s requested page can be returned properly. Line 23 is the else statement corresponding with the original if(err) statement on line 13. Line 24 sets the response code as 200 (successful) and includes in the header that the returned file is of type html. Line 25 formally ends the response and passes the retrieved page data in the “data” object, with utf-8 encoding. |
| **Line 30** | Defines the kernel-level access point that enables the application to use the networking stack present on the operating system. So, when an HTTP request is sent to the machine, my machine can properly route the request to the running Node application. |
| **Line 31** | Asks the operating system to “reserve” the specified port for the application’s traffic. So, when a request is sent, we know for sure that data routed through that port will be passed to the Node application. |
| **Line 32** | Simply prints a logging message in the console notifying the administrator that the Node application is running and able to accept requests, on the specified port. |
| **Additional thoughts** |  |