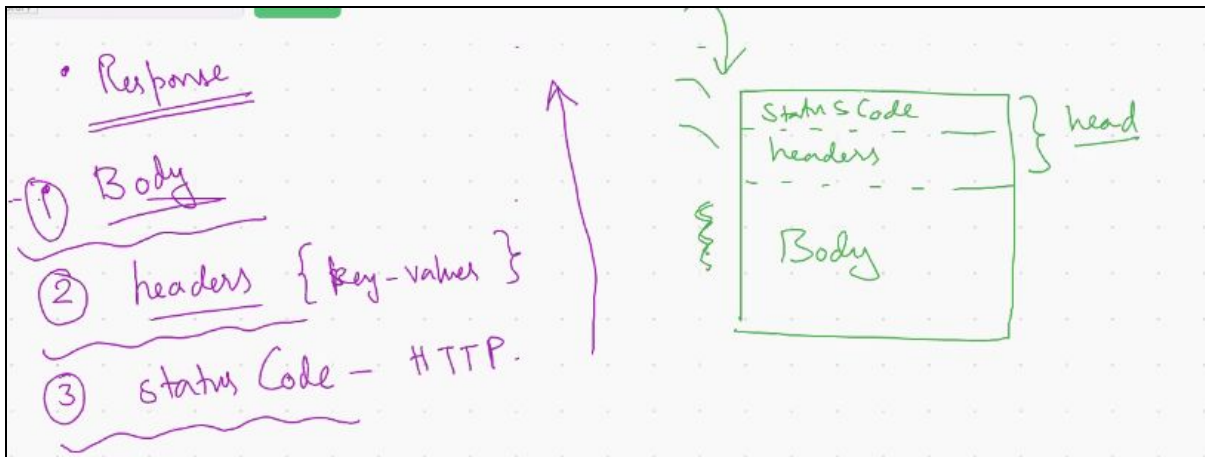


Title: Rest apis

Topics covered:

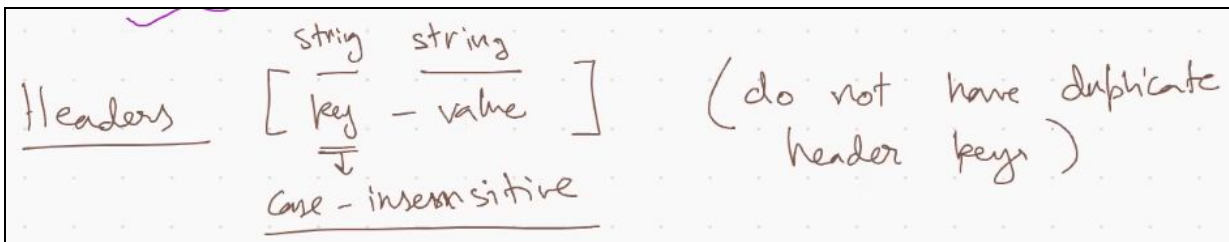
- **Response apis**
 - **Body**
 - **Headers**
 - **Status Code-http**
- **1st Assessment question on backend node server**
- **Rest api**

Response is sent in this order: **Status Code -> Headers -> Body**



First status code is sent then the headers then the body

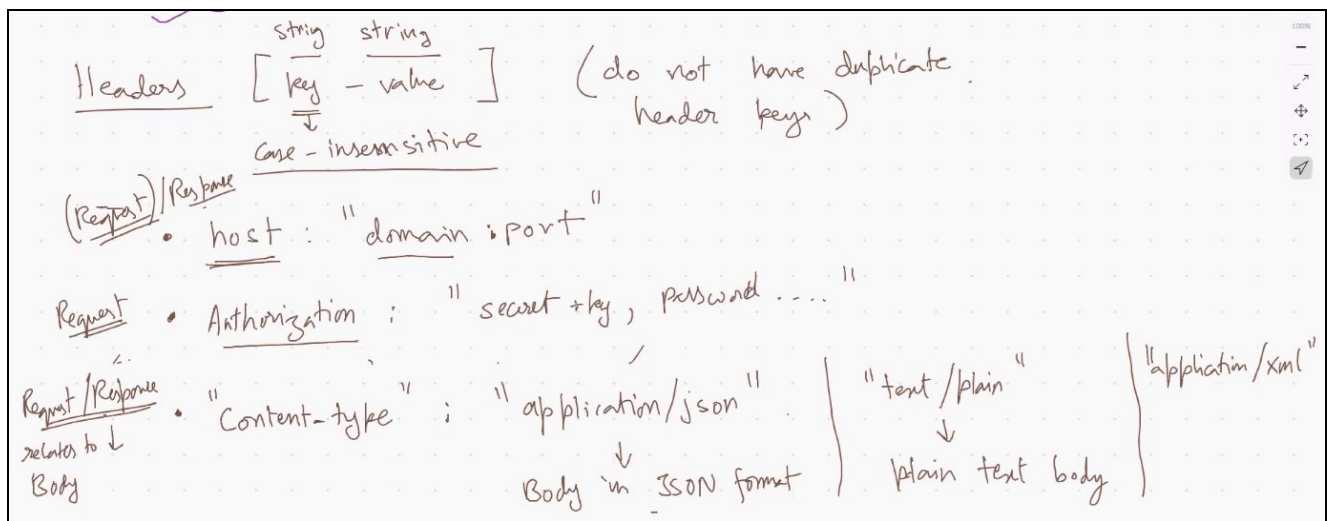
Headers



Headers should not have duplicate keys

Certain headers

1. Host header: conventionally it is present in the request
2. Authorization header, it has a secret key or password used by server to detect whether you authenticated it or not.
3. Content-type header: It is used to tell what type of content we are sending with the response, whether it is json or text or xml or something else.

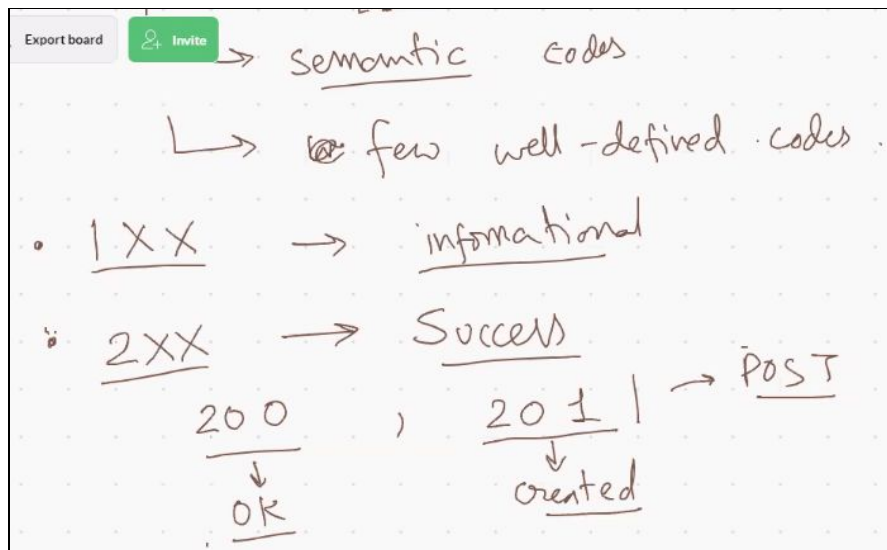


Status Codes:

<https://www.restapitutorial.com/httpstatuscodes.html>

Most of the time 200 is used for success response "OK".

201: when something is created on Database / Server (POST method uses it).

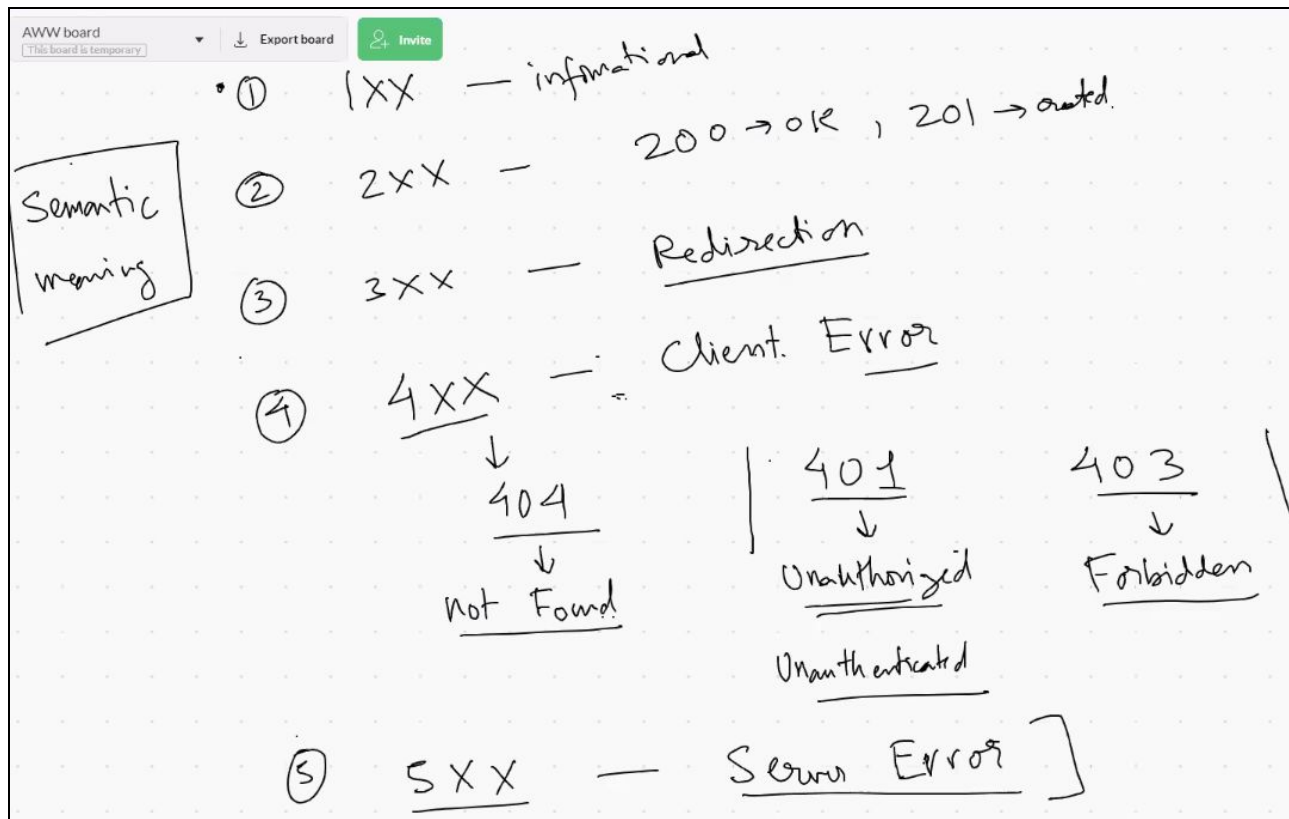


401 is when you are not logged in, but still trying to access the site.

403 is server can figure out who you are but you don't have permission

404 - page not found

5xx: Server not able to connect to Database.



Three response apis

```
17 response.writeHead(statusCode, responseHeaders);
18 response.write(responseBody);
19 response.end();
```

```
JS server.js  {} package.json  JS utils.js
src > JS server.js > http.createServer() callback
10
11 http.createServer((request, response) => {
12   console.log('Request to Host: ', request.headers.host);
13   console.log('Request Verb', request.method);
14   console.log('Request headers', request.headers);
15   console.log('Request Path/query params', request.url);
16
17   response.writeHead(statusCode, responseHeaders);
18   response.write(responseBody);
19   response.end();
20
21 }).listen(9999);
```

Response headers are key value pair

```
17 response.writeHead(200, {
18   'content-type': 'text/plain'
19 });
20 response.write('Hello Newton School');
```

```

src > JS server.js > http.createServer() callback
10
11 http.createServer((request, response) => {
12     console.log('Request to Host: ', request.headers.host);
13     console.log('Request Verb', request.method);
14     console.log('Request headers', request.headers);
15     console.log('Request Path/query params', request.url);
16
17     response.writeHead(200, {
18         'content-type': 'text/plain'
19     });
20     response.write('Hello Newton School');
21     response.end();
22
23 }).listen(9999);

```

The screenshot shows the VS Code interface with a REST client extension. At the top, there's a toolbar with buttons for 'NEW', 'Runner', 'Import', and 'Builder'. Below this, a notification bar states 'Chrome apps are being deprecated. Download our free native apps for continued support and better performance. [Learn more](#)'. The main area displays a REST client request configuration. The request method is 'GET' and the URL is 'localhost:9999/anypath2?abc=123&qac=45&flag=true'. The 'Params' tab is selected, and a 'Sending...' button is visible. Below the request configuration, there are tabs for 'Authorization', 'Headers (3)', 'Body', 'Pre-request Script', and 'Tests'. The 'Body' tab is selected, and it shows a 'Loading...' status with a 'Cancel Request' button. At the bottom, the 'Body' tab is selected, and it shows the response body as 'Hello Newton School' in a 'Text' format.

Response body should always be string

Status: 200 OK

Body Cookies Headers (4) Test Results

connection → keep-alive

content-type → text/plain

date → Fri, 27 Nov 2020 16:33:48 GMT

transfer-encoding → chunked

localhost:9999/anypat

No Environment

GET

localhost:9999/anypath2?abc=123&qac=45&flag=true

Params

Send

Authorization

Headers (3)

Body

Pre-request Script

Tests

Type

No Auth

Body Cookies Headers (4) Test Results

Status: 200 OK

connection → keep-alive

content-type → text/plain

date → Fri, 27 Nov 2020 16:33:48 GMT

transfer-encoding → chunked

```
17 response.writeHead(255, {  
18   'content-type': 'text/plain'  
19 });
```

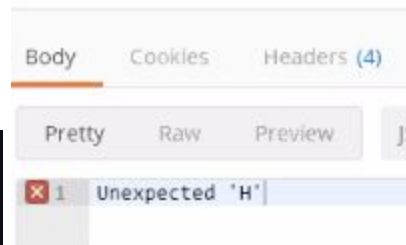
Status: 255 unknown

Unknown because did not recognize the code, because its not a standard code

```

17 response.writeHead(255, {
18   'content-type': 'application/json'
19 });

```



Because written json was not correct

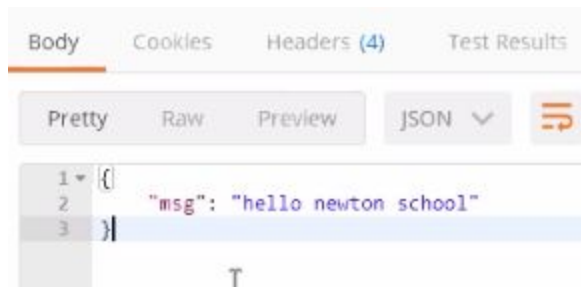
```

20 response.write('{ "msg" : "hello newton school" }');

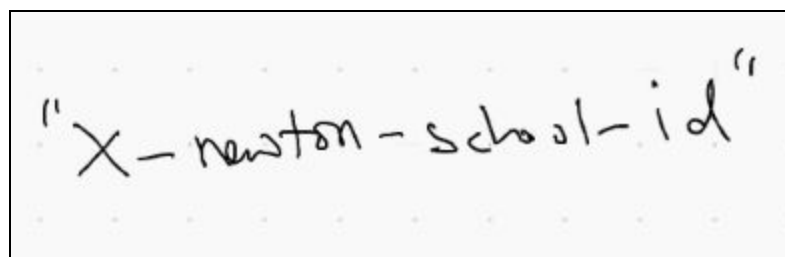
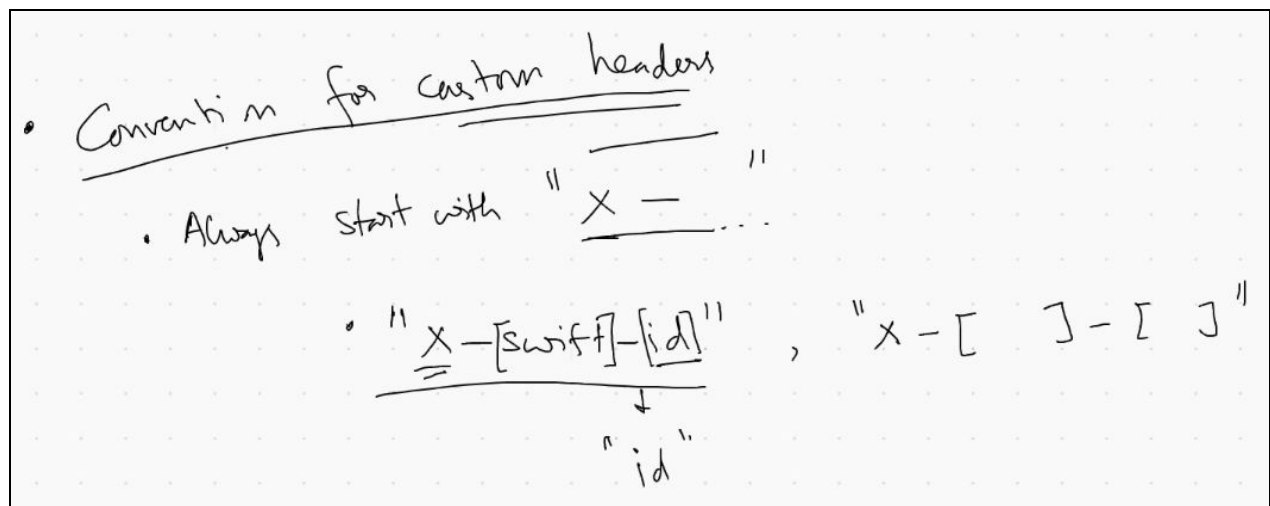
```

Corrected

Output



Convention for custom headers



Another example

"x-swift-id"
↓ ↓
no spaces "id"

Key value pair this way

```
{  
  "x-swift-id" : "23",  
  "x-vs-id" : "240"  
}
```

Solving in call assignment

```
C:\Users\Debanshu\Documents\Newton School\Backend-Assignments---Node-Server-Dominos-needs-your-help>npm install nodemon  
[ ] \ finalize:diff: sill finalize C:\Users\Debanshu\Documents\Newt
```

```
8 | "start": "nodemon src/index.js"
```

This will restart the server automatically whenever there is a change in the code in index.js

```
package.json
{
  "scripts": {
    "test": "mocha '___tests___/*.js' --reporter mochawesome --recursive",
    "start": "nodemon src/index.js"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "mochawesome": "6.1.1",
    "nodemon": "^2.0.6"
  },
  "devDependencies": {
    "chai": "^4.2.0",
    "chai-http": "^4.3.0",
    "mocha": "^8.2.1"
  }
}
```

Npm start

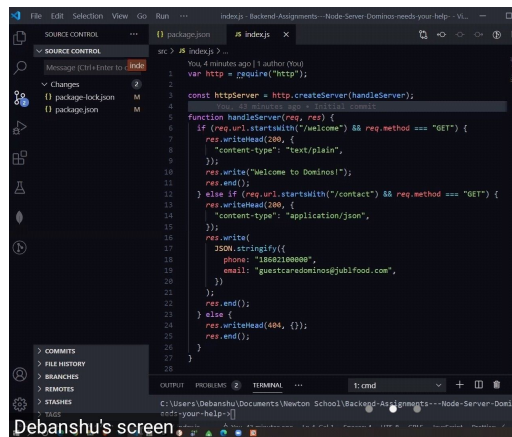
```
> node-server-testing@1.0.0 start C:\Users\Debanshu\Documents\Newton School\Backend-A
ssignments---Node-Server-Dominos-needs-your-help-
> nodemon src/index.js

[nodemon] 2.0.6
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node src/index.js`
[nodemon] clean exit - waiting for changes before restart
[nodemon] restarting due to changes...
[nodemon] starting `node src/index.js`
[nodemon] clean exit - waiting for changes before restart
```

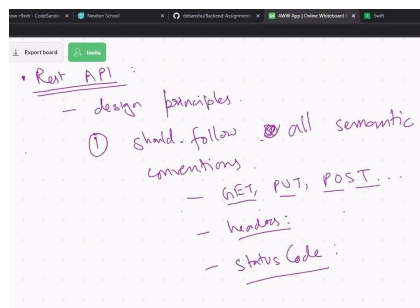


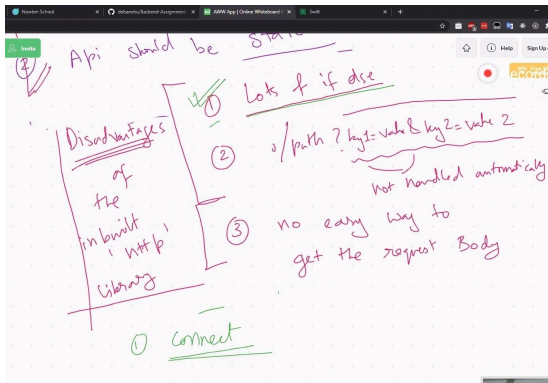
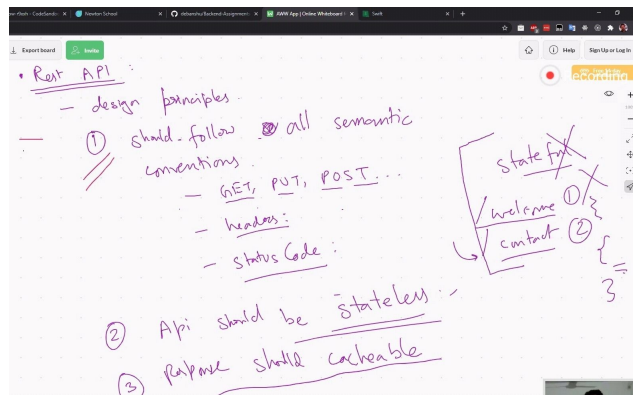
```
src > JS index.js > handleServer

...
1  var http = require("http");
2
3  const httpServer = http.createServer(handleServer);
4
5
6  function handleServer(req, res) {
7      if(req.url.startsWith("/welcome") && req.method === "GET") {
8          res.writeHead(200, {
9              'content-type': 'text/plain'
10             });
11             res.write('Welcome to Dominos!');
12             res.end();
13         }
14
15
16
17
18
19     httpServer.listen(8081);
20
21     module.exports = httpServer;
```



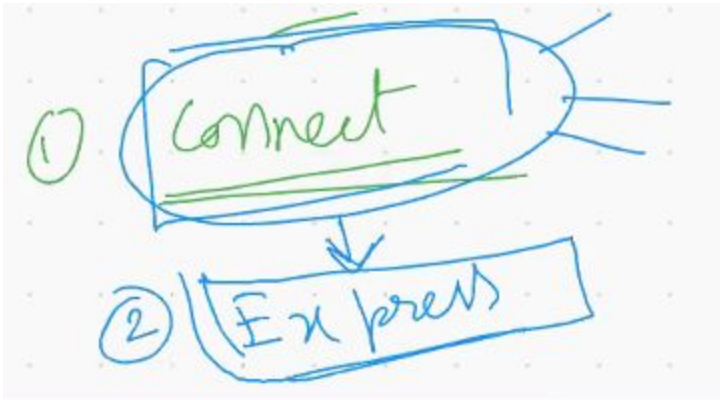
Debanshu's screen





Disadvantages of the in-built http library:

Two famous external libraries:



Note: We can use Switch instead of If-Else in http, but we will not be able to do the “startsWith” matching in that case.

A screenshot of a video lecture window. The main content is a whiteboard with handwritten notes in red and blue ink. The notes are organized into a list of disadvantages of the built-in http library, with a small diagram at the bottom. The notes include:

- ① Api should be static
- ② Lots of if else
- ③ no easy way to get the request Body
- ④ no path ? key=value & key2=value2 (Not handled automatically)

On the left side, a vertical note says: "Disadvantages of the built-in http library". At the bottom, there is a small diagram showing a rounded box labeled '① Connect' with an arrow pointing to a rectangle labeled '② Express'.