Node.js basics

**Table content**:

* How does the Web Work?
* Creating Node.js server
* Using node core modules
* Working with request & responses (basic)
* Async code & event loop

1. **How the Web Works**a. user visiting a web page by entering URL of the web page.  
   b. when he enter, browser reaches domain name servers & browser request that server with that IP address.  
   c. we(developer) write code which run on that server. We write that code which handle in-coming request and do some operation on that & send response (some html code, html text, file, json, xml). Here node.js come into the picture (php, asp.net, ….) in this code contain input validation, communicating with DB, authentication and many more.  
   d. Request and response transmission is done by some protocol (a standard way to communicate).   
   HTTP = a protocol for transferring data which is understood by browser & server. HTTPS = HTTP + data encryption (during Transmission)
2. **Creating Node.js server**a. app.js / server.js is the root file of node.js application.  
   b. core modulus: http (launch a server, send a request), https (launch a SSL server), fs (file), path, OS, ….  
   c. create a server:  
    A. importing a http module:  
    const http = require(“http”);  
    import our files should start with / or ./ .if don’t add that it will not load that local file (whose name is http.js in our project folder) intend it will look for global module.  
    B. creating a server:  
    const server = http.createServer((req, res)=>{});  
    createServer() take requestListener as parameter, which is function which will execute for every incoming request. requestListener also take 2 parameters: request (of type IncomingMessage) has data which we pass during calling, response (of type ServerResponse) which we can use to return data.  
    C. run a server:  
    server.listen(3000); // 3000 is port number  
    due to this nodejs will not exist execution immediately intend it will listen incoming request.   
   listen() take couple of parameter : port no. (default port is 80), hostname (default host is localhost)
3. **Node lifecycle & Event loop**  
   when we run node app.js: it start executing script, parse code, register variable & functions. We never exit the code, this is event-loop  
   it is loop process manage by node.js which keeps on running as long as event listener is registered. Our Node application is manage by Event Loop.  
   a. Nodejs is running on a single thread. (Node us event driven approach)  
   b. nodejs has on going event loop as long as there are listeners and create server, has a listener that never stops.  
   c. we can unregister listener by process.exit() (we don’t do this reason this will quit our server & client cannot be able to reach our webpage). Process.exit basically hard exited our event loop & therefore the program shuts-down because there were no work to do.
4. **Understanding requests**a. url = evert thing after base url  
   b. method = get/post  
   c. header = meta information added to request
5. **Sending response**we send data back using response obj.  
   a. setHeader() = to add header of response such as code, datatype,…  
   b. write() = to add data to the response  
   c. end() = to end response and send back s response (after this we cannot use write() it will give error).
6. [**HTTP headers**](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers)   
    let the client and the server pass additional information with an HTTP request or response. An Http header consist of its case-insensitive name followed by a colon(:), then by its value. (for-more go to link in title)
7. **Routing request**returning data based on urls.
8. **Redirecting request**
9. **Parsing Data (Stream & Buffer)**a. we parse data to get data form the request.  
   b. incoming data is passed as STREAM of data.  
   **example**: consider an incoming request, (stream is an ongoing process) nodejs read request in chucks. Consider a file upload case, in this we a write file in chucks to the disk(server) because upload at once will take more time to parse entire file.   
   c. Node handle all requests in chucks because it don’t know how complex and big the data is. So we can start working on the data early.  
   d. The problem is we can not arbitrarily try to work with these chunks. Instead we organized them we use BUFFER.  
   e. Buffer is like a bus-stop. Buffer hold multiple chucks and work with them before they are released once you done. We work with buffer.  
    f. to read chucks,  
    const body = []  
    request.on(“data”,(chunk)=>{body.push(chunk);});   
    adding listen using 0n(), which take 2 parameters : event (in this case is data) & listener function. Above listener is use to read data.  
     
     
     
     
    request.on(“end”,()=>{  
    const parseBody = Buffer.concat(body).toString();  
    });  
    now we got all the chucks , now we need to work with them. Buffer.concat() we concat our chunks. And we got new buffer. toString() as we know string will come. For other case we need to do another operation