**Code for simple examples**

**Some modules whose exported objects/methods we will need to use:**

* **http module** <https://nodejs.org/api/http.html>
	+ **http.createServer(function(req, res) { }) and properties of req, res**
		- **req.method**
		- **req.on()** an event handler
		- **res.writeHead()**
		- **res.end()**
	+ **listen(portnumber) or listen(portnumber, url)**
	+ **You will often see code like:** const PORT = process.env.PORT || 3000;
	This allows you to use the environment's port if it is specified, and then, obviously, you would use this as the parameter for portnumber in listen().
* **url module**
	+ **url.parse() method – *sadly this is now "legacy & you should be using the search and searchParams methods*** [**https://nodejs.org/api/url.html#url\_class\_url**](https://nodejs.org/api/url.html#url_class_url)
* **querystringmodule – *this is also now legacy and has been replaced by URLSearchParams*** [**https://nodejs.org/docs/latest/api/url.html#url\_class\_urlsearchparams**](https://nodejs.org/docs/latest/api/url.html#url_class_urlsearchparams)
* **fs module**
	+ **readFileSync()**
	+ **The replace() method used here is JavaScript and not from Node**
* **Also for JSON objects the stringify and parse methods.**
	+ **JSON.stringify(some\_JavaScript\_object) is a JSON string**
	+ **JSON.stringify(some\_JSON\_String) is a JavaScript object**

**NOTE: There is a global JSON object in JavaScript, so the two lines above are using the methods which come with that object and are not using Node.**

**First some comments about paths, query strings etc.**

For a GET the key-value pairs have been appended to the request URL --- so we will extract them from there.

For a POST it is in the body --- and typically it would be the data returned when there was a *data* event.
We would then store it in a variable (customarily called body), use querystring.parse(body). And store that in a variable myDataFromForm. Now we can access the values which the form posted to us as myDataFromForm.name, etc.
You can find examples of this kind of code at <https://stackoverflow.com/questions/4295782/how-to-process-post-data-in-node-js> (scroll past the express examples) **and** [**https://codezup.com/handle-process-http-post-request-data-node-js-tutorial/**](https://codezup.com/handle-process-http-post-request-data-node-js-tutorial/) **(about 2/3 of the way down) (let's look at this now – I sent you a copy of this tutorial *please notice that the body variable holds an object with the data which was POSTed by the form. Of course, in your callback function you may make use of this data!* )** and <https://www.edureka.co/community/74955/how-to-process-post-data-in-node-js> As mentioned above, instead of using the querystring module we need to use URLSearchParams.

Alternatively, you can put all the data together and then split it on the = sign. This is shown in <https://www.tutorialspoint.com/parsing-request-body-in-node> Or possibly use JSON.parse on that data. See <https://flaviocopes.com/nodejs-parse-json/> Another example of readin data in chunks is at <https://itnext.io/how-to-handle-the-post-request-body-in-node-js-without-using-a-framework-cd2038b93190>

From the Brad Dayley book on Node etc 2nd edition <https://learning.oreilly.com/library/view/nodejs-mongodb-and/9780134655642/ch07.xhtml>

Remember that in Node.js we always have \_\_dirname and \_\_filename



And we can extract all of these usin the **path**  module, which has a **parse** method.

<https://nodejs.org/api/path.html#path_path_parse_path>

As time allows – brief look at Chapter 7 of the Dayley book.

Note that url.parse() has been replaced by URLSearchParams -see <https://nodejs.org/docs/latest/api/url.html#url_class_urlsearchparams> or the **origin proprty on req.url --- see diagram at** [**https://nodejs.org/docs/latest/api/path.html#path\_path\_parse\_path**](https://nodejs.org/docs/latest/api/path.html#path_path_parse_path)As pointed out at [**Node.js URLsearchParams API - GeeksforGeeks**](https://www.geeksforgeeks.org/node-js-urlsearchparams-api/) we can access the .get method of a URLSearchParams object to get a specific value.

**But we will take the easier way, and use Express.**

Express is a small and flexible framework, which likes to describe itself as *unopinionated.*

**There are several good reasons to do this --- it saves us a lot of low-level coding, and also it makes the management of large websites much, much easier.**

**The most popular framework for doing this is Express –** and Chapter 18 of the Brad Dayley book has a very clear introduction to it. <https://learning.oreilly.com/library/view/nodejs-mongodb-and/9780134655642/ch18.xhtml#ch18> There is also a Brad Traversy video at <https://www.youtube.com/watch?v=L72fhGm1tfE> (Postman is a Chrome extension which will simply make a get or post request) and a tutorial at [ExpressJS Tutorial - Tutorialspoint](https://www.tutorialspoint.com/expressjs/index.htm) -start at the [ExpressJS - Hello World - Tutorialspoint](https://www.tutorialspoint.com/expressjs/expressjs_hello_world.htm) Hello World page.
**There is an excllent tutorial at** [**Express/Node introduction - Learn web development | MDN (mozilla.org)**](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction) **and you know that that one (unlike many others) will be current.**

**Okay – let's use Express!!! Here are the steps:**

**install in the folder for your project**

It's easier if you do it before the npm init, but if you have already created the package.json file, then install express with the –save option.

**Or install express globally** (I'll assume you did that)

The basic format is

 **express = require('express');
 app = express();
 app.listen(3000);**

and then you handle requests with:

 for a get request from a route /someRoute
 **app.** **get(/someRoute, (**req, res) => {whateverCode} **)**;

for a post request from a route /someRoute
 **app.** **post(/someRoute, (**req, res) => {whateverCode} **)**;

**NOTE**: The callback functions (req, res) => {whateverCode} should have as their last line **next();**  which is needed to pass control to the next handler.

In express, instead of writing headers etc we use
 **res.send(**stuff**);**

In order to handle (parse) data in the body of a POST request or from the URL-encoded GET request, we first bring in some extra help. (Here are are using the express version of what we previously did by parsing the req.url or the body in vanilla Node.)

 app.use(express.urlencoded()); //Handle urlencoded data such as from a GET

 app.use(express.json()); //Handle json data such as in the body of a POST
 //request.body/someKey will have the associated valueb

Some people prefer to require the body-parser module:

In *older* code you will see the bodyParser required as a separate module, as in:

var bodyParser = require('body-parser');

//To parse URL encoded data

app.use(bodyParser.urlencoded({ extended: false }))

//To parse json data

app.use(bodyParser.json())

**But the body-parser is now part or Express and you should NOT require it separately.
So, now we code:**

app.use(express.urlencoded({extended: true}));

 app.use(express.json()) // To parse the incoming requests with JSON payloads

Other examples are discussed at <https://stackoverflow.com/questions/66525078/bodyparser-is-deprecated>

Note: When you are writing a server-side script to handle a post, you know the names of the form elements you were sent. So, if you are not using Node, you can use
 myURL = whatever you got from the request object;
 let aValue = myURL.searchParams. get('nameOfElement');
 If the element on the form may have multiple values:
 let arrayOfValues = myURL.searchParams. getAll('nameOfElement');

 See <https://nodejs.org/api/url.html#the-whatwg-url-api> for the documentation on the URL module, and the WHATWG documentation or see <https://www.geeksforgeeks.org/node-js-urlsearchparams-api/> for examples.
Please also note that the URL module is used to construct new URLs. And <https://nodejs.org/api/url.html#url-strings-and-url-objects> has a very clear picture showing the parts/methods of a URL with the legacy (deprecated) names above the URL and the new WHATWG parts/methods below. The username and password in the legacy version had security issues, which is why that API has been deprecated.

**What next?**

* **For express we might want to learn more about how to handle many routes**
* **We might want to learn how to use the data from a request and interact with a mongodb database (there are node modules mongo and mongoose)**
* **We might want to learn about a template engine (e.** **g. pug or handlebars or ejs**
* **We might want to learn about a framework like Angular for the front end.** Angular appears to be declining in popularity --- REACT has been gaining.