TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

PHÁT TRIỂN ỨNG DỤNG WEB VỚI NODEJS (502070)

LAB2 – NODEJS

TDTU

PHÁT TRIỂN ỨNG DỤNG WEB VỚI NODEJS
Lab2 – NodeJS

Documentation | Node.js (nodejs.org)

Introduction to Node.js (nodejs.dev)

Node.js Introduction (w3schools.com)

Node.js is an open-source and cross-platform JavaScript runtime environment.
Node.js runs the V8 JavaScript engine, the core of Google Chrome, outside of the browser. This allows Node.js to be very performant.

```
var http = require('http');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.end('Hello World!');
}).listen(8080);
```

Node.js HTTP Module (w3schools.com)

Node.js has a built-in module called HTTP, which allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP).

```
var http = require('http');

//create a server object:
http.createServer(function (req, res) {
  res.write('Hello World!'); //write a response to the client
  res.end(); //end the response
}).listen(8080); //the server object listens on port 8080
```

Node.js HTTP Module (w3schools.com)

If the response from the HTTP server is supposed to be displayed as HTML, you should include an HTTP header with the correct content type

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write('Hello World!');
  res.end();
}).listen(8080);
```

Node.js HTTP Module (w3schools.com)

The function passed into the http.createServer() has a req argument that represents the request from the client, as an object (http.IncomingMessage object).
This object has a property called "url" which holds the part of the url that comes after the domain name

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write(req.url);
  res.end();
}).listen(8080);
```

http://localhost:8080/summer

```
/summer
```

Node.js HTTP Module (w3schools.com)

There are built-in modules to easily split the query string into readable parts, such as the URL module

```
var http = require('http');
var url = require('url');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  var q = url.parse(req.url, true).query;
  var txt = q.year + " " + q.month;
  res.end(txt);
}).listen(8080);
```

http://localhost:8080/?year=2017&month=July

```
2017 July
```

How can I read POST data? | Node.js (nodejs.org)

```
var http = require('http');
var postHTML =
  '<html><head><title>Post Example</title></head>' +
  '<body>' +
  '<form method="post">' +
  'Input 1: <input name="input1"><br>' +
  'Input 2: <input name="input2"><br>' +
  '<input type="submit">' +
  '</form>' +
  '</body></html>';

http.createServer(function (req, res) {
  var body = "";
  req.on('data', function (chunk) {
    body += chunk;
  });
  req.on('end', function () {
    console.log('POSTed: ' + body);
    res.writeHead(200);
    res.end(postHTML);
  });
}).listen(8080);
```

How To Create a Web Server in Node.js with the HTTP Module | DigitalOcean

```
http.createServer(function(req,res){
        // normalize url by removing querystring, optional
        // trailing slash, and making it lowercase
        var path = req.url.replace(/\/?(?:\?.*)?$/, '').toLowerCase();
        switch(path) {
                case '':
                        res.writeHead(200, { 'Content-Type': 'text/plain' });
                        res.end('Homepage');
                        break;
                case '/about':
                        res.writeHead(200, { 'Content-Type': 'text/plain' });
                        res.end('About');
                        break;
                default:
                        res.writeHead(404, { 'Content-Type': 'text/plain' });
                        res.end('Not Found');
                        break;
        }
}).listen(3000);
```

The Node.js fs module (nodejs.dev)

```
var http = require('http'),
        fs = require('fs');

function serveStaticFile(res, path, contentType, responseCode) {
        if(!responseCode) responseCode = 200;
        fs.readFile(__dirname + path, function(err,data) {
                if(err) {
                        res.writeHead(500, { 'Content-Type': 'text/plain' });
                        res.end('500 - Internal Error');
                } else {
                        res.writeHead(responseCode,
                                { 'Content-Type': contentType });
                        res.end(data);
                }
        });
}
```

Lab2 – 1 – Calculator

1	Create GET endpoint (/)	- Declear HTML element inside js file - Form action=http://localhost:<port>/result method="GET" - http://localhost:<port> should render the form
2	Create GET endpoint (/result)	- Use parse/querystring of request to get params

3	Define operation logic	- 1 endpoint (/result): send all operations and operators to the endpoint. OR
		- 4 endpoints:
		  - /add
		  - /subtract
		  - /multiply
		  - /divide

4	Use built-in http module to create and start backend server	Note: pay attention to urlencode/urldecode

Lab2 – 2 – Login

1	Create separate HTML login file	- Bootstrap - Form action=http://localhost:<port>/login method="POST"
2	Create POST endpoint (/login)	- Use querystring to parse body login info - Hardcode email and password - Use url module to create routers
3	Use fs to read HTML login file resource	- Handle errors/exceptions
4	Use built-in http module to create and start backend server	

Lab2 – 3 – API endpoint

```
const Students = [
    {id: 1, name: 'Student 1', email: 'email1@student.com'},
    {id: 2, name: 'Student 2', email: 'email2@student.com'},
    {id: 3, name: 'Student 3', email: 'email3@student.com'},
    {id: 4, name: 'Student 4', email: 'email4@student.com'},
    {id: 5, name: 'Student 5', email: 'email5@student.com'},
    {id: 6, name: 'Student 6', email: 'email6@student.com'},
    {id: 7, name: 'Student 7', email: 'email7@student.com'},
    {id: 8, name: 'Student 8', email: 'email8@student.com'},
    {id: 9, name: 'Student 9', email: 'email9@student.com'},
    {id: 10, name: 'Student 10', email: 'email10@student.com'}
]
```

1	Use built-in http module to create and start backend server	
2	Create endpoints	- GET /students: get all JSON.stringify() - POST /student: add a new student (201/204 err) - GET /students/:id: get one student - PUT /students/:id: update a student - DELETE /students/:id: delete a student
3	Students data	- Create a dummy array object - Add/update/delete an array element: basic programming
4	Verify APIs	- Postman/CURL

Thank you