{

    "node-websocket-server":{

        "scope": "javascript",

        "prefix": "node-websocket-server",

        "body": [

            "import WebSocket, { WebSocketServer } from 'ws';\n",

            "const wss = new WebSocketServer({",

            "\tport: 4001",

            "});\n",

            "wss.on('connection', client => {",

                "\tconsole.log('client connected')",

                "\tsetInterval(()=>{",

                "\t\tclient.send(`Hello, Client`)",

                "\t}, 2000)\n",

                "\tclient.on('message', message => {",

                "\t\tconsole.log(message.toString())",

                "\t})",

                "\tclient.on('close', () => {",

                "\t\tconsole.log('РџРѕР»СЊР·РѕРІР°С‚РµР»СЊ РѕС‚РєР»СЋС‡РёР»СЃСЏ');",

                "\t})",

            "})"

        ]

    },

    "node-websocket-client":{

        "scope": "javascript",

        "prefix": "node-websocket-client",

        "body": [

            "import WebSocket from 'ws';\n",

            "const ws = new WebSocket('ws://localhost:4001');",

            "ws.on('open', () => {",

                "\tws.onmessage = message => {",

                    "\t\tconsole.log('Message: %s', message.data);",

                "\t};",

                "\tsetInterval(()=>{",

                    "\t\tws.send('Hello, server')",

                "\t}, 3000)",

            "});"

        ]

    },

    "node-websocket-file-upload":{

        "scope": "javascript",

        "prefix": "node-websocket-file-upload",

        "body": [

            "const fs = require('fs');",

            "const WebSocket = require('ws');\n",

            "//Server",

            "const wss = new WebSocket.Server({port: 4000, host: 'localhost'});\n",

            "let k = 0;",

            "wss.on('connection', ws =>{",

                "\tconst duplex = WebSocket.createWebSocketStream(ws, {encoding: 'utf8'});",

                "\tlet wfile = fs.createWriteStream(`./upload/file${++k}.txt`);",

                "\tduplex.pipe(wfile);",

            "});",

            "\n//Client",

            "const ws = new WebSocket('ws://localhost:4000');\n",

            "ws.on('open', () => {",

                "\tconst duplex = WebSocket.createWebSocketStream(ws, {encoding: 'utf8'});",

                "\tlet rfile = fs.createReadStream(`./MyFile.txt`);",

                "\trfile.pipe(duplex);",

            "});"

        ]

    },

    "node-rpc-server":{

        "scope": "javascript",

        "prefix": "node-rpc-server",

        "body": [

            "const rpcWSS = require('rpc-websockets').Server\n",

            "let server =  new rpcWSS({port: 4000, host: 'localhost'});\n",

            "server.setAuth(credentials => credentials.login === 'admin' && credentials.password === 'admin');\n",

            "server.register('func1', params => {",

                "\treturn 1;",

            "}).public();\n",

            "server.register('func2', params => {",

                "\treturn 1;",

            "}).protected();",

        ]

    },

    "node-rpc-client":{

        "scope": "javascript",

        "prefix": "node-rpc-client",

        "body": [

            "const rpcWSC = WebSocket =  require('rpc-websockets').Client;",

            "let ws = new rpcWSC('ws://localhost:4000');\n",

            "ws.on('open', () => {",

                "\tws.call('func1').then(r => { console.log('func1 = ', r); });",

                "\tws.login({login: 'admin', password: 'admin'})",

                    "\t\t.then(() => {",

                        "\t\t\tws.call('func2').catch(e => { console.log('catch func2: ', e)}).then(r => { console.log('func2 = ', r);});",

                    "\t\t})",

            "});"

        ]

    },

    "node-rpc-client-parallel":{

        "scope": "javascript",

        "prefix": "node-rpc-client-parallel",

        "body": [

            "const async = require('async');",

            "const rpcWSC = require('rpc-websockets').Client;",

            "let ws = new rpcWSC('ws://localhost:4000');\n",

            "ws.on('open', () => async.parallel({",

                "\tsquare: (cb)=>{ws.call('square', [5]).catch((e)=>cb(e,null)).then((r)=>cb(null,r));},",

                "\tsum: (cb)=>{ws.call('sum', [3,4,5]).catch((e)=>cb(e,null)).then((r)=>cb(null,r));},",

            "}, (error, result) => {",

                "\tif(error)",

                    "\t\tconsole.log('error =', error);",

                "\telse",

                    "\t\tconsole.log('result =', result);",

                "\tws.close();",

            "});"

        ]

    },

    "node-fact":{

        "scope": "javascript",

        "prefix": "node-fact",

        "body": [

            "const factorial = n => {",

                "\treturn (n !== 1) ? n \* factorial(n - 1) : 1;",

            "}"

        ]

    },

    "node-fibonachi":{

        "scope": "javascript",

        "prefix": "node-fibonachi",

        "body": [

            "const fib = n => {",

                "\tvar fibonacci = [0, 1];",

                "\tfor (i = 2; i < n; i ++) {",

                    "\t\tfibonacci[i] = fibonacci[i-1] + fibonacci[i-2];",

                "\t}",

                "\treturn fibonacci;",

            "}"

        ]

    },

    "node-event-socket":{

        "scope": "javascript",

        "prefix": "node-event-socket",

        "body": [

            "const rpcServer = require('rpc-websockets').Server;\n",

            "const eventSocket = new rpcServer({ port: 4000, host: 'localhost', path: '/'});\n",

            "eventSocket.event('A');",

            "eventSocket.event('B');\n",

            "console.log('Choose A, B event');\n",

            "let input = process.stdin;",

            "input.setEncoding('utf-8');\n",

            "input.on('data', data => {",

                "\teventSocket.emit(data.slice(0, -2))",

            "});"

        ]

    },

    "node-event-socket-client":{

        "scope": "javascript",

        "prefix": "node-event-socket-client",

        "body": [

            "const rpc = require('rpc-websockets').Client;",

            "const eventSocket = new rpc('ws://localhost:4000');\n",

            "eventSocket.on('open', () => {",

                "\teventSocket.subscribe('A');",

                "\teventSocket.on('A', () => console.log('It is A event!' + new Date().toString()));",

            "});"

        ]

    },

    "node-fs-watch":{

        "scope": "javascript",

        "prefix": "node-fs-watch",

        "body": [

            "fs.watch(pathToFile, {encoding: 'buffer'}, (eventType, filename) => {",

                "\tif(eventType === 'change')",

                    "\t\tconsole.log('changed')",

            "});"

        ]

    },

    "node-fs-read-file-sync":{

        "scope": "javascript",

        "prefix": "node-fs-read-file-sync",

        "body": [

            "const fs = require('fs');\n",

            "const readFile = () => {",

                "\ttry {",

                    "\t\treturn fs.readFileSync(pathToFile).toString();",

                "\t} catch (e) {",

                    "\t\tprocess.stderr.write(",

                        "\t\t\t'err: file not exists'",

                    "\t\t);",

                    "\t\tprocess.exit(1);",

                "\t}",

            "};"

        ]

    },

    "node-fs-read-file-async":{

        "scope": "javascript",

        "prefix": "node-fs-read-file-async",

        "body": [

            "var fs = require('fs');\n",

            "fs.readFile('input.txt', function (err, data) {",

                "\tif (err) return console.error(err);",

                "\tconsole.log(data.toString());",

            "});"

        ]

    },

    "node-http-server":{

        "scope": "javascript",

        "prefix": "node-http-server",

        "body": [

            "var http = require('http');",

            "http.createServer((request, response) => {",

                "\tresponse.writeHead(200, {'Content-Type': 'text/plain'});",

                "\tresponse.end('Hello World');",

            "}).listen(8080);\n",

            "console.log('Server running at http://127.0.0.1:8080/');"

        ]

    },

    "node-tcp-server":{

        "scope": "javascript",

        "prefix": "node-tcp-server",

        "body": [

            "const net = require('net');\n",

            "const host = '127.0.0.1';",

            "const port = 40000;\n",

            "net.createServer(sock =>",

            "{",

                "\tconsole.log('Server Connected: ' + sock.remoteAddress + ':' + sock.remotePort);",

                "\tsock.on('data', data =>",

                "\t{",

                    "\t\tconsole.log('Message from Client: ' + data);",

                    "\t\tsock.write('ECHO: ' + data);",

                "\t});",

                "\tsock.on('close', () => console.log('Client refused connection'));",

            "}).listen(port, host);\n",

            "console.log('TCP-server started');"

        ]

    },

    "node-tcp-client":{

        "scope": "javascript",

        "prefix": "node-tcp-client",

        "body": [

            "const net = require('net');\n",

            "const host = '127.0.0.1';",

            "const port = 40000;",

            "const client = new net.Socket();\n",

            "client.connect(port, host, () =>",

            "{",

                "\tconsole.log('Client connected:  ', client.remoteAddress + ' ' + client.remotePort);",

                "\tclient.write('Hello');",

            "})\n",

            "client.on('data', data =>",

            "{",

                "\tconsole.log('Server message: ', data.toString());",

                "\tclient.destroy();",

            "});"

        ]

    },

    "node-udp-server":{

        "scope": "javascript",

        "prefix": "node-udp-server",

        "body": [

            "var dgram = require('dgram');",

            "var server = dgram.createSocket('udp4');\n",

            "const host = '127.0.0.1';",

            "const port = 3000;\n",

            "server.on('message', (message, rinfo) => {",

                "\tconsole.log('server got: ' + message + ' from client ' + rinfo.address + ':' + rinfo.port);",

                "\tvar answer = 'ECHO: ' + message;",

                "\tserver.send(answer, 0, answer.length, rinfo.port, host, () =>{});",

            "});\n",

            "server.bind(port, host, () => console.log('UDP-server started'));\n",

            "server.on('error', error => {",

                "\tconsole.log('server error:' + error.stack);",

                "\tserver.close();",

            "});"

        ]

    },

    "node-udp-client":{

        "scope": "javascript",

        "prefix": "node-udp-client",

        "body": [

            "var dgram = require('dgram');",

            "var client = dgram.createSocket('udp4');\n",

            "const host = '127.0.0.1';",

            "const port = 3000;\n",

            "process.stdin.on('data', (data) => {",

                "\tclient.send(data, 0, data.length, port, host, () => console.log('successful'))",

            "});\n",

            "client.on('message', (message)=> {",

                "\tconsole.log('client got: ' + message);",

                "\tclient.close();",

            "});\n",

            "client.on('close', () => process.exit(0));"

        ]

    },

    "node-async-await":{

        "scope": "javascript",

        "prefix": "node-async-await",

        "body": [

            "async function getData() {",

                "\treturn Promise.resolve('data');",

            "}\n",

            "async function getMoreData(data) {",

                "\treturn Promise.resolve(data + 'more data');",

            "}\n",

            "async function getAll() {",

                "\tconst data = await getData();",

                "\tconst moreData = await getMoreData(data);",

                "\treturn `All the data: ${data}, ${moreData}`;",

            "}\n",

            "getAll().then((all) => {",

                "\tconsole.log('all the data')",

            "})"

        ]

    },

    "node-event-emmiter":{

        "scope": "javascript",

        "prefix": "node-event-emmiter",

        "body": [

            "var events = require('events');\n",

            "var eventEmitter = new events.EventEmitter();\n",

            "eventEmitter.emit('my\_event');\n",

            "eventEmitter.on('my\_event', () => {",

              "\tconsole.log('data received successfully.');",

            "});"

        ]

    },

    "node-simple-promise":{

        "scope": "javascript",

        "prefix": "node-simple-promise",

        "body": [

            "const promise = new Promise((resolve, reject) => {",

                "\tif (true) ",

                  "\t\tresolve('data')",

                "\telse",

                  "\t\treject('error')",

             "})"

        ]

    },

    "node-promise-all":{

        "scope": "javascript",

        "prefix": "node-promise-all",

        "body": [

            "const getData = () => Promise.resolve('data');",

            "const getMoreData = () => Promise.resolve('more data');\n",

            "Promise.all(",

                "\tgetData(),",

                "\tgetMoreData()",

            ").then(result => {",

                "\tconst [data, moreData] = result;",

            "})"

        ]

    },

    "node-parse-xml-from-request":{

        "scope": "javascript",

        "prefix": "node-parse-xml-from-request",

        "body": [

            "import {parseString} from 'xml2js';\n",

            "let data = '';",

            "req.on('data', chunk => { data += chunk; });",

            "req.on('end', () => {",

                "\tres.writeHead(200, {'Content-type': 'application/xml'});",

                "\tparseString(data, (err, result) => {",

                    "\t\tlet id = result.request.$.id;",

                    "\t\tlet xSum = 0;",

                    "\t\tlet mSum = '';",

                    "\t\tresult.request.x.forEach((p) => {",

                        "\t\t\txSum += parseInt(p.$.value);",

                    "\t\t});",

                    "\t\tresult.request.m.forEach((p) => {",

                        "\t\t\tmSum += p.$.value;",

                    "\t\t});",

                    "\t\tlet xmlDoc = xmlbuilder.create('response').att('id', id);",

                    "\t\txmlDoc.ele('sum').att('result', xSum).up().ele('concat').att('element', 'm').att('result', mSum);",

                    "\t\tres.end(xmlDoc.toString());",

            "\t});",

            "});",

            "\n//Sample XML:",

            "//<request id=\"28\">",

            "//\t<x value=\"1\"/>",

            "//\t<x value=\"5\"/>",

            "//\t<m value=\"a\"/>",

            "//\t<m value=\"b\"/>",

            "//\t<m value=\"c\"/>",

            "//</request>"

        ]

    },

    "node-static-file-server":{

        "scope": "javascript",

        "prefix": "node-static-file-server",

        "body": [

            "const fs = require('fs')\n",

            "class StaticFileServer {",

                "\tconstructor(filepath = 'static') {",

                    "\t\tthis.filepath = filepath",

                "\t}\n",

                "\tfileNotFoundError(resp) {",

                    "\t\tresp.statusCode = 404",

                    "\t\tresp.statusMessage = 'Resource not found'",

                    "\t\tresp.end('Resource not found')",

                "\t}\n",

                "\tpathIsValid(extension, filename) {",

                    "\t\tconst r = new RegExp(`^\/.+\\.${extension}$`)",

                    "\t\treturn r.test(filename)",

                "\t}\n",

                "\tsendFile(req, resp, headers) {",

                    "\t\tfs.access(this.#makePathStatic(req.url), fs.constants.R\_OK, (err) => {",

                        "\t\t\tif (err) this.fileNotFoundError(resp)",

                        "\t\t\telse this.#pipeFile(req, resp, headers)",

                    "\t\t})",

                "\t}\n",

                "\t#makePathStatic(filename) {",

                    "\t\treturn `${this.filepath}${filename}`",

                "\t}\n",

                "\t#pipeFile(req, resp, headers) {",

                    "\t\tresp.writeHead(200, headers)",

                    "\t\tfs.createReadStream(this.#makePathStatic(req.url)).pipe(resp)",

                "\t}\n",

            "}\n",

            "module.exports = (path) => new StaticFileServer(path)"

        ]

    },

    "node-immediate":{

        "scope": "javascript",

        "prefix": "node-immediate",

        "body": [

            "const calcFactorialImmediate = (n, cb) => {",

                "\tfunction helper (sum,i) {",

                    "\t\tif (i === 0) return cb(sum);",

                    "\t\telse setImmediate( helper.bind(null,sum\*i,i-1) );",

                "\t};",

                "\treturn helper(1,n);",

            "}\n",

            "const input = 10",

            "calcFactorialNextTick(input, result => {",

                "\tconsole.log(result);",

            "})"

        ]

    },

    "node-next-tick":{

        "scope": "javascript",

        "prefix": "node-next-tick",

        "body": [

            "const calcFactorialNextTick = (n, cb) => {",

                "\tfunction helper (sum,i) {",

                    "\t\tif (i === 0)",

                        "\t\t\treturn cb(sum);",

                    "\t\telse",

                        "\t\t\tprocess.nextTick(helper.bind(null, sum\*i , i - 1));",

                "\t};",

                "\treturn helper(1,n);",

            "}\n",

            "const input = 10",

            "calcFactorialNextTick(input, result => {",

                "\tconsole.log(result);",

            "})"

        ]

    },

    "node-http-client-xml":{

        "scope": "javascript",

        "prefix": "node-http-client-xml",

        "body": [

            "let http=require('http');",

            "const options = {",

                "\thostname: '127.0.0.1',",

                "\tport: 3000,",

                "\tpath: '/',",

                "\tmethod: 'POST',",

                "\theaders: {",

                "\t\t'Content-Type': 'application/xml',",

                    "\t\t\t}",

                "\t};\n",

            "let req=http.request(options, (res)=>{",

                "\tlet xxxxml='';",

                "\tres.on('data',(chunk)=>{xxxxml+=chunk;})",

                "\tres.on('end',()=>{console.log(xxxxml)})",

            "})\n",

            "req.write(`",

                "\t<request id='A'>",

                "\t<x val='1'></x>",

                "\t<x val='2'></x>",

                "\t<x val='-51'></x>",

                "\t<s> string1 </s>",

                "\t<s> FFFFFFF </s>",

                "\t<s> string 2 </s>",

                "\t</request>",

            "`);\n",

            "req.end();"

        ]

    },

    "node-parametrized\_module":{

        "scope": "javascript",

        "prefix": "node-parametrized\_module",

        "body": [

            "var modules = {",

                "\tA: require('./A'),",

                "\tB: require('./B'),",

                "\tC: require('./C')",

            "}",

            "function X(impl) {",

                "\tif(impl in modules)",

                    "\t\treturn new modules[impl];",

                "\telse",

                    "\tthrow new Error('Unknown impl: ' + impl);",

            "}",

            "module.exports = X;\n",

            "//Calling",

            "var foo = new X('A');",

            "foo.method();",

            "// => 'A'",

            "var bar = new X('B');"

        ]

    },

    "node-getMethod-params":{

        "scope": "javascript",

        "prefix": "node-getMethod-params",

        "body": [

            "const http = require('http');",

            "const url = require('url');/n",

            "http.createServer((request, response) => {",

                "\tconsole.log('server work');",

                "\tif (request.method == 'GET') {",

                    "\t\t// GET -> РїРѕР»СѓС‡РёС‚СЊ РѕР±СЂР°Р±РѕС‚Р°С‚СЊ",

                    "\t\tlet urlRequest = url.parse(request.url, true);",

                    "\t\tconsole.log(urlRequest.query.test); // ! GET Params",

                    "\t\tresponse.end(urlRequest.query.test);",

                "\t}",

            "}).listen(3000);"

        ]

    },

    "node-postMethod-params":{

        "scope": "javascript",

        "prefix": "node-postMethod-params",

        "body": [

        "let body = '';",

        "request.on('data', chunk => {",

            "\tbody += chunk.toString();",

        "});",

        "request.on('end', () => {",

            "\tconsole.log(body);",

            "\tlet params = parse(body);",

            "\tconsole.log(params);",

            "\tconsole.log(params.hi);",

            "\tresponse.end('ok');",

        "});"

        ]

    },

    "node-sendFile":{

        "scope": "javascript",

        "prefix": "node-sendFile",

        "body": [

            "const http = require('http')",

            "const fs = require('fs')",

            "http.createServer(function (request, response) {",

                "\tconsole.log(`Р—Р°РїСЂРѕС€РµРЅРЅС‹Р№ Р°РґСЂРµСЃ: ${request.url}`)",

                "\tif (request.url.startsWith('/public/')) {",

                    "\t\t// РїРѕР»СѓС‡Р°РµРј РїСѓС‚СЊ РїРѕСЃР»Рµ СЃР»РµС€Р°",

                    "\t\tconst filePath = request.url.substr(1)",

                    "\t\tfs.readFile(filePath, function (error, data) {",

                        "\t\t\tif (error) {",

                        "\t\t\tresponse.statusCode = 404",

                        "\t\t\tresponse.end('Resourse not found!')",

                        "\t\t\t} else {",

                        "\t\t\tresponse.setHeader('Content-Type', 'text/html')",

                        "\t\t\tresponse.end(data)",

                        "\t\t}",

                    "\t})",

                "\t} else {",

                  "\t\t// РІРѕ РІСЃРµС… РѕСЃС‚Р°Р»СЊРЅС‹С… СЃР»СѓС‡Р°СЏС… РѕС‚РїСЂР°РІР»СЏРµРј СЃС‚СЂРѕРєСѓ hello world!",

                  "\t\tresponse.end('Hello World!')",

                "\t}",

          "}).listen(3000)"

        ]

    },

}