香 港 中 文 大 學
The Chinese University of Hong Kong

CSCI2720 - Building Web Applications

Lecture 14: Node.JS with Express

Dr Colin Tsang

Outline

- Overview of Node.js
- Express Basics
  - Routing
  - Retrieving data from query string (GET) and from body (POST)
  - Generating content of a response
  - Retrieving and setting header fields from a request
  - Retrieving and setting cookie and sessions

Node.js and Express

- *Node.js* – a JavaScript run-time environment
  - Was first released in 2009
  - Makes writing servers, including web servers, easier
  - Runs JS on server side (using Chrome V8 engine)
  - It uses *non-blocking I/O*
    - No waiting for I/O, network operations and other software

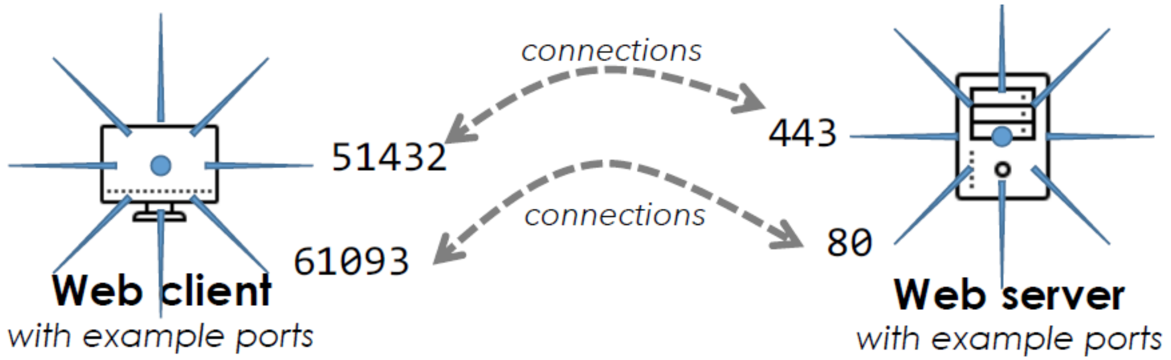
- *Express* – a module (add-in) for Node.js
  - Allows easy set up of web and mobile applications

Node.js and Express

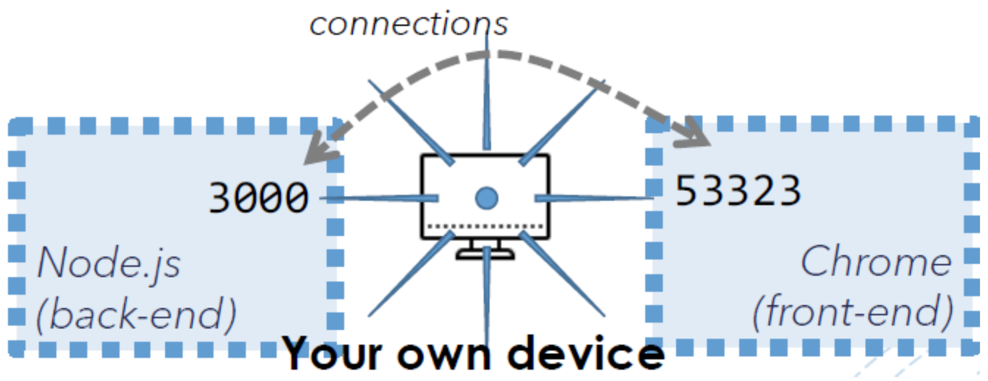
- With Node.js, to implement a web application, a common approach is to create a custom-made web server by
  - Incorporating a web application framework like Express
  - *Including only the needed modules*
  - Writing application specific script in JavaScript

Usual scenario of Node.js

- This is often how people set up Node.js with Express



- But you can also do this:



Process of the web server

- Typical steps involved in a Request-Response cycle GET PUT
  1. Routing (deciding the actions to take based on URL and HTTP method)
  2. Retrieve data from an HTTP request Fetch
  3. Process the data, e.g.,
     - Validation
     - Apply business logic
     - Update database
  4. Generate an HTTP response

The Express Framework

- Express is a minimal and flexible Node.js web application framework
- Core features of Express allows one to
  - Define a routing table
    - To map request URL and HTTP method to an action
  - Set up middleware to respond to HTTP Requests
  - Use template engine to produce HTML output

- Ref: http://www.tutorialspoint.com/nodejs/nodejs_express_framework.htm

Installation

- To use Node.js, it needs to be installed onto the machine which will act as the web server
- Available as *Current* and *LTS* (long term support) versions
  - Multiple platforms
  - https://nodejs.org/en/download/current/
- Although you can run a zip version without installing, Node.js cannot listen to the server ports on a machine without administrator rights.
- Cloud version: try StackBlitz for a blank Node.js project, using Google Chrome for support of WebContainers

NPM

- Node.js allows the management of modules through npm
- Modules are like libraries, and we can install them when needed
- Set up the folder first
  - **npm init**
    - The installed modules will exist as a folder *node_modules* under the app folder
- To install additional modules using npm, e.g., Express
  - **npm install express**
- More steps are indeed required by Express, see:
  - https://expressjs.com/en/starter/installing.html

Helo World!

- After setting up Node.js and Express, create app.js (the entry point) anywhere on your computer
  - Note: this .js file is not run in a browser, but at the web server of Node/Express

- See: https://stackblitz.com/edit/colin-node-helloworld-vrnosv
  - **req** is an object representing the current HTTP request
  - **res** is an object representing the current HTTP response

Hello World!

- Start the server with the command:
  - `node app.js`
  - The system path may need to be adjusted for the command to run
- Now the server is ready to be accessed at port 3000
  - To try on your own machine, access http://localhost:3000
- See: https://expressjs.com/en/starter/hello-world.html

Express basics

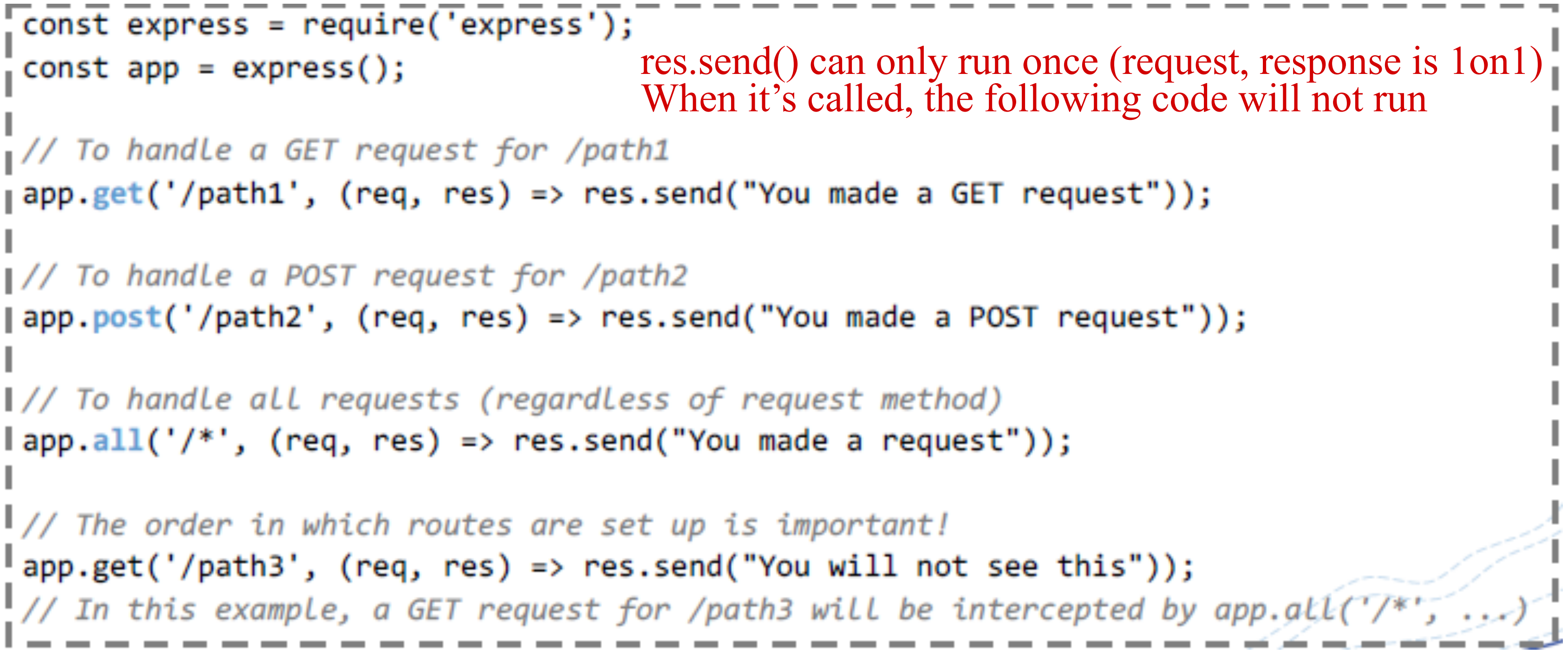
- We are going to discuss the following basic Express function:
  - Routing
  - Retrieving data from query string (GET) and from body (POST)
  - Generating content of a response
  - Retrieving and setting header fields from a request
  - Retrieving and setting cookie and sessions

Routing

- Routing is to determine the response based on the request URI and method
  - *Virtual files and paths* can be specified in the URL
  - The path specified by the URL is simply parsed as a string
- In express, routing depends on the HTTP method
  - `app.METHOD(route_path, callback);`
  - *METHOD* can be on of **GET**, **POST**, **PUT**, **DELETE**, **ALL**
    Usually not allowed
- The route path can be strings, string pattern, or regular expressions
  - See: https://expressjs.com/en/guide/routing.html
- Query strings are not part of the route path
  - If a URL is http://hostname/x/y?key1=value1, only */x/y* will be matched against the route path

Route based on request method

- The code can be found at: https://stackblitz.com/edit/colin-node-param-pgv5ga
- You can only use **res.send()** once for a response.



```
const express = require('express');
const app = express();

// To handle a GET request for /path1
app.get('/path1', (req, res) => res.send("You made a GET request"));

// To handle a POST request for /path2
app.post('/path2', (req, res) => res.send("You made a POST request"));

// To handle all requests (regardless of request method)
app.all('/*', (req, res) => res.send("You made a request"));

// The order in which routes are set up is important!
app.get('/path3', (req, res) => res.send("You will not see this"));
// In this example, a GET request for /path3 will be intercepted by app.all('/*', ...)
```

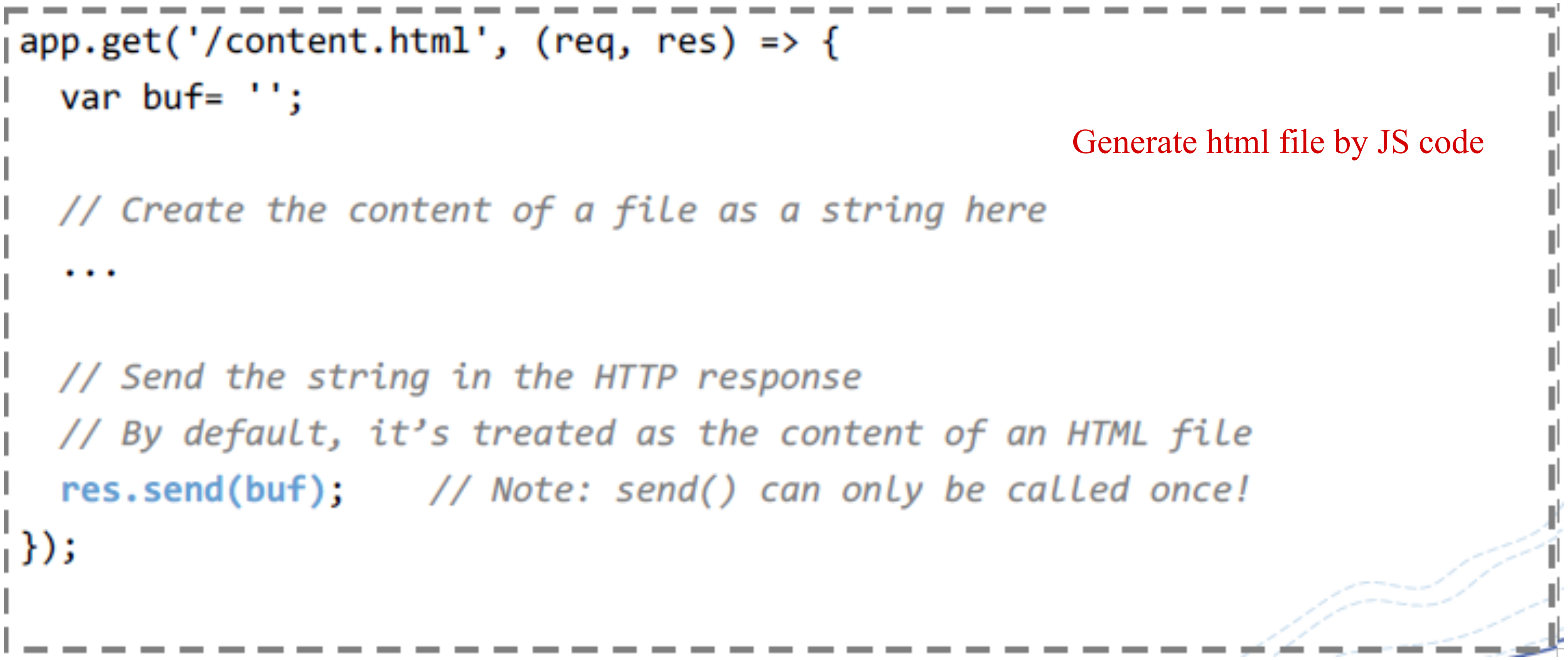
res.send() can only run once (request, response is 1on1)
When it's called, the following code will not run

Execuation Order

Route Path

```
// Regular expression matching: e.g., any path that ends with .jpg
// Note: The expression is not enclosed by any quotes
app.all(/.*\.jpg$/, (req, res) => res.send("You requested a JPG file"));

// Route parameters matching
// e.g., http://hostname/course/2720/lecture/6
app.all('/course/:cID/lecture/:lID', (req, res) => res.send(req.params));
   // Output: {"cID":"2720", lID":"6"}

// hyphen and dot (- and .) are interpreted literally
// e.g., http://hostname/csci2720-t2
app.all('/:course-:tutorial', (req, res) => res.send(req.params));
    // Output: {"course":"csci2720", "tutorial":"t2"}
```

Generating file content dynamically



```
app.get('/content.html', (req, res) => {
  var buf= '';

  // Create the content of a file as a string here
  ...

  // Send the string in the HTTP response
  // By default, it’s treated as the content of an HTML file
  res.send(buf);    // Note: send() can only be called once!
});
```

Generate html file by JS code

Serving static files

- **res.sendFile()** transfers the file at the given absolute path
  - You cannot use both **res.send()** and **res.sendFile()** in one response
- It sets the *Content-Type* response HTTP header field based on the file extension

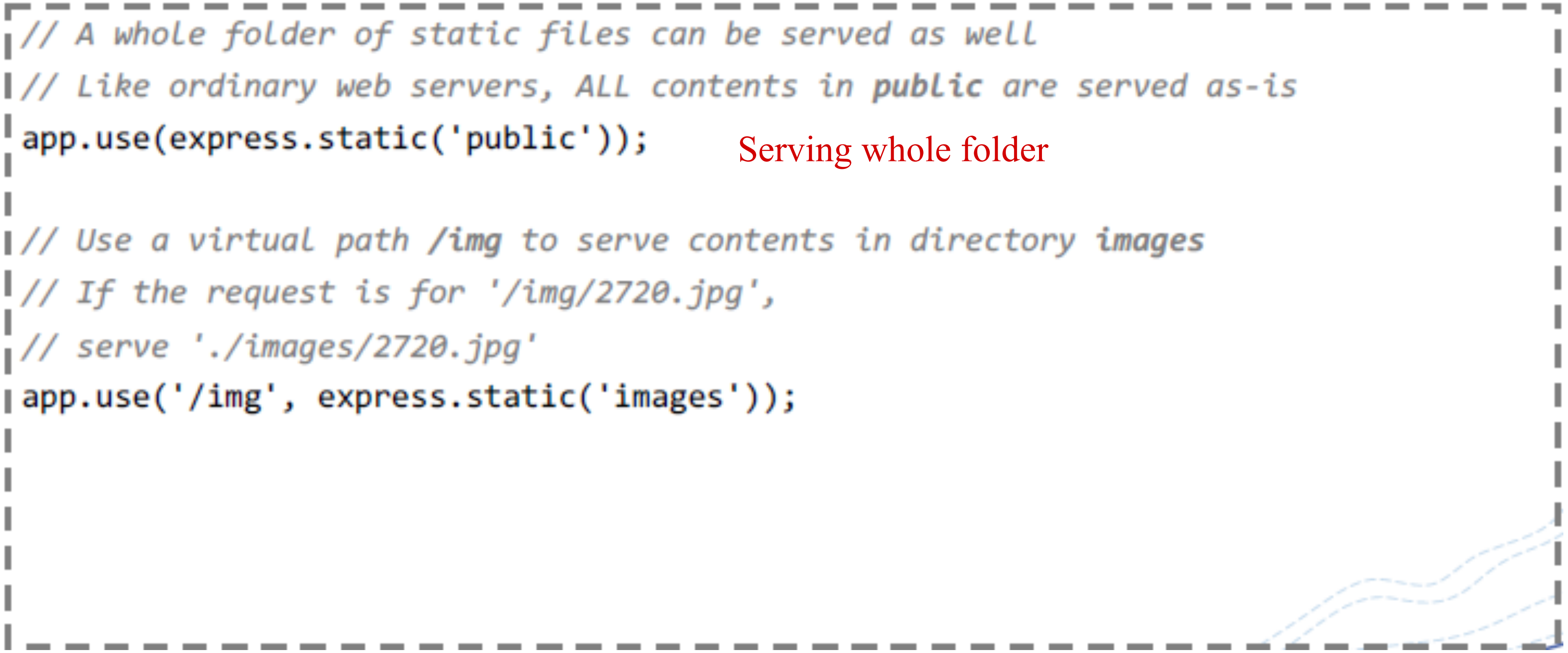
```
app.get('/', (req, res) => {
  // Send the file 'index.html' in the folder of the current script
  res.sendFile(__dirname + '/index.html');
  // __dirname holds absolute path of the folder of the current script
});
```

Static html file

See: https://expressjs.com/en/4x/api.html

Serving static files

- More methods to serve static files:

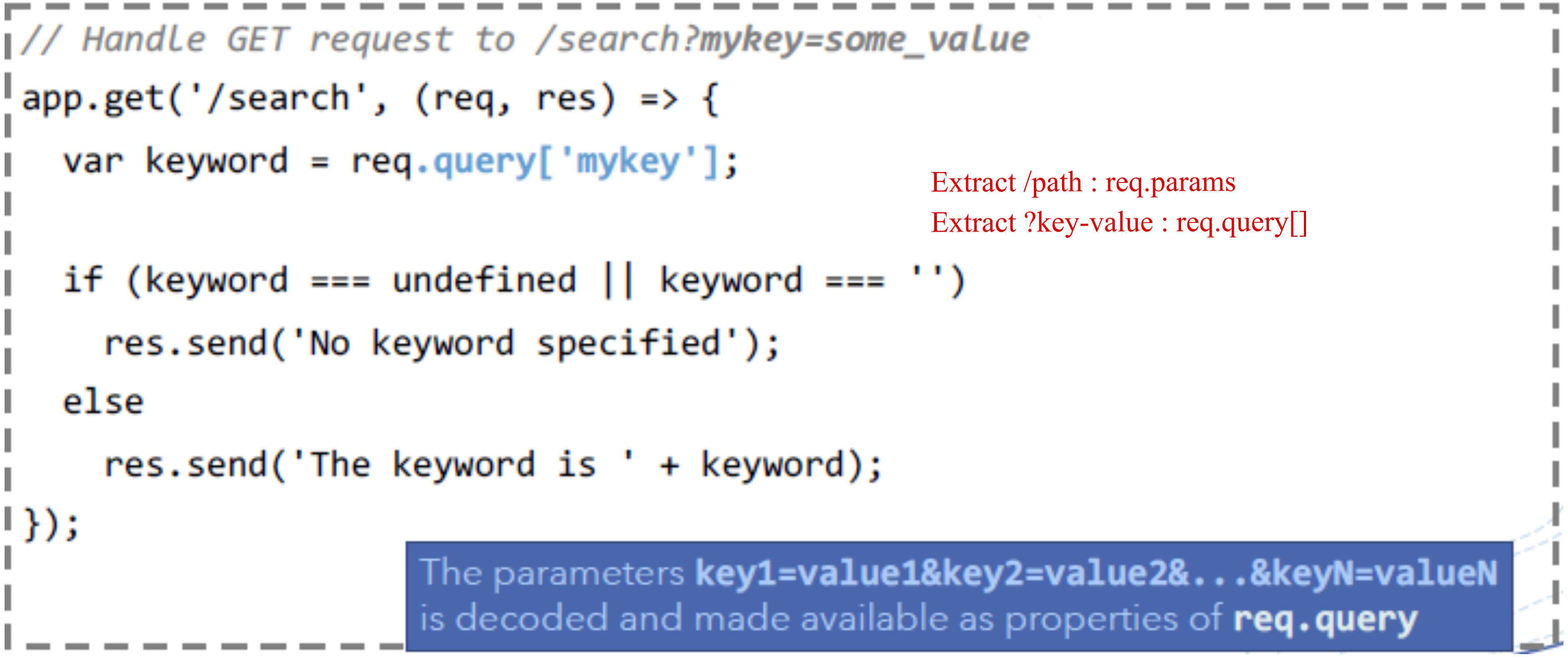


```
// A whole folder of static files can be served as well
// Like ordinary web servers, ALL contents in public are served as-is
app.use(express.static('public'));

// Use a virtual path /img to serve contents in directory images
// If the request is for '/img/2720.jpg',
// serve './images/2720.jpg'
app.use('/img', express.static('images'));
```

Serving whole folder

GET parameters from a query string

- The code can be found at: https://stackblitz.com/edit/colin-node-getpost-ld7jxc



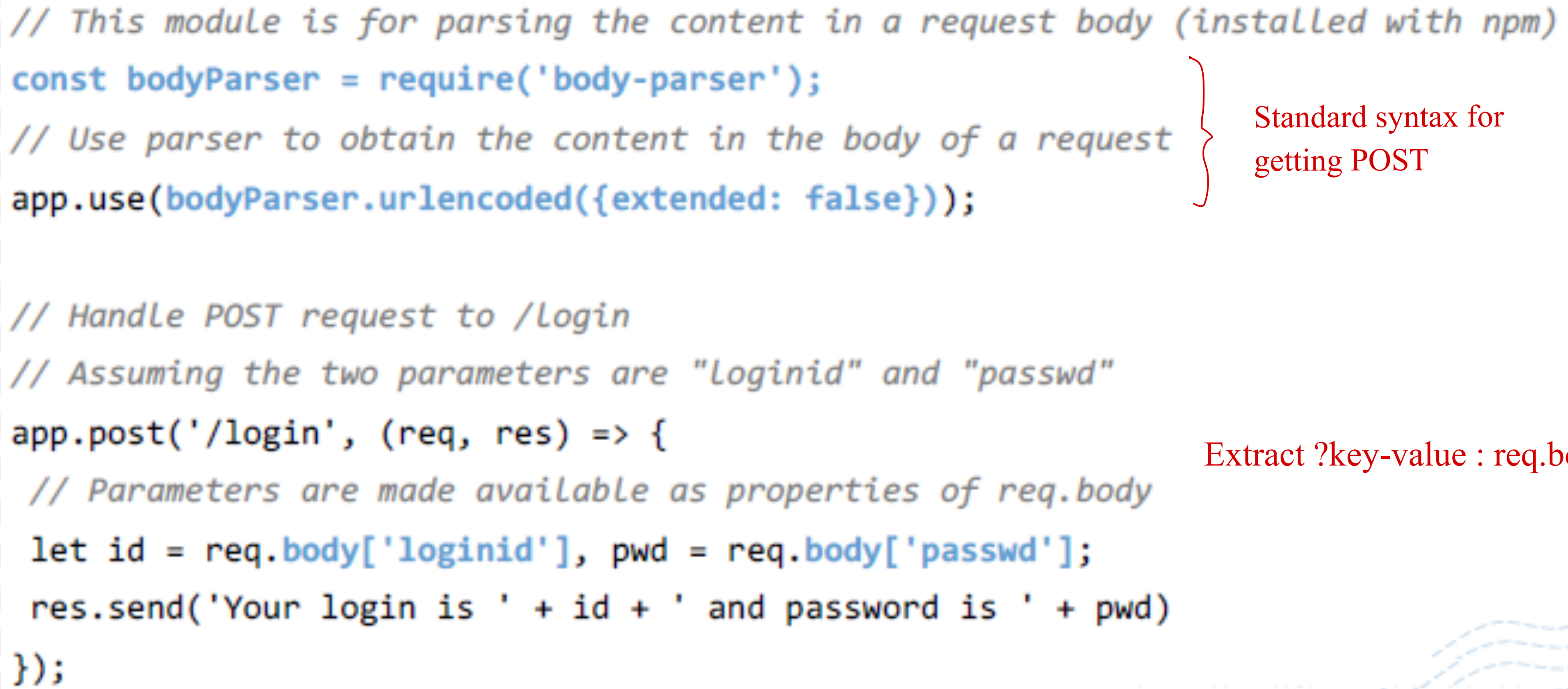
```
// Handle GET request to /search?mykey=some_value
app.get('/search', (req, res) => {
  var keyword = req.query['mykey'];

  if (keyword === undefined || keyword === '')
    res.send('No keyword specified');
  else
    res.send('The keyword is ' + keyword);
});
```

Extract /path : req.params
Extract ?key-value : req.query[]

The parameters **key1=value1&key2=value2&...&keyN=valueN** is decoded and made available as properties of **req.query**

POST parameters in request body

In the http request body, not link -> more secure (protected by http encoding method)

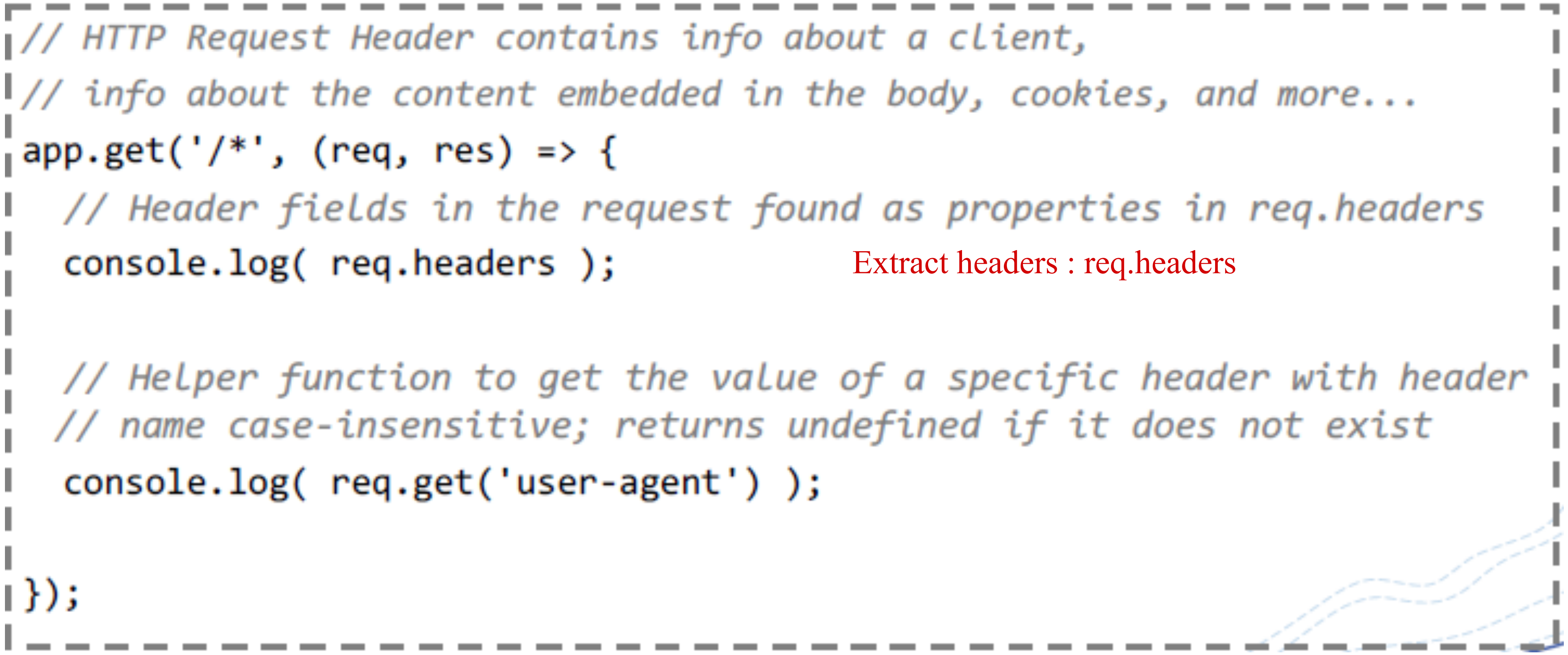


```
// This module is for parsing the content in a request body (installed with npm)
const bodyParser = require('body-parser');
// Use parser to obtain the content in the body of a request
app.use(bodyParser.urlencoded({extended: false}));

// Handle POST request to /login
// Assuming the two parameters are "loginid" and "passwd"
app.post('/login', (req, res) => {
  // Parameters are made available as properties of req.body
  let id = req.body['loginid'], pwd = req.body['passwd'];
  res.send('Your login is ' + id + ' and password is ' + pwd)
});
```

Standard syntax for getting POST

Extract ?key-value : req.body[]

Retrieving request headers

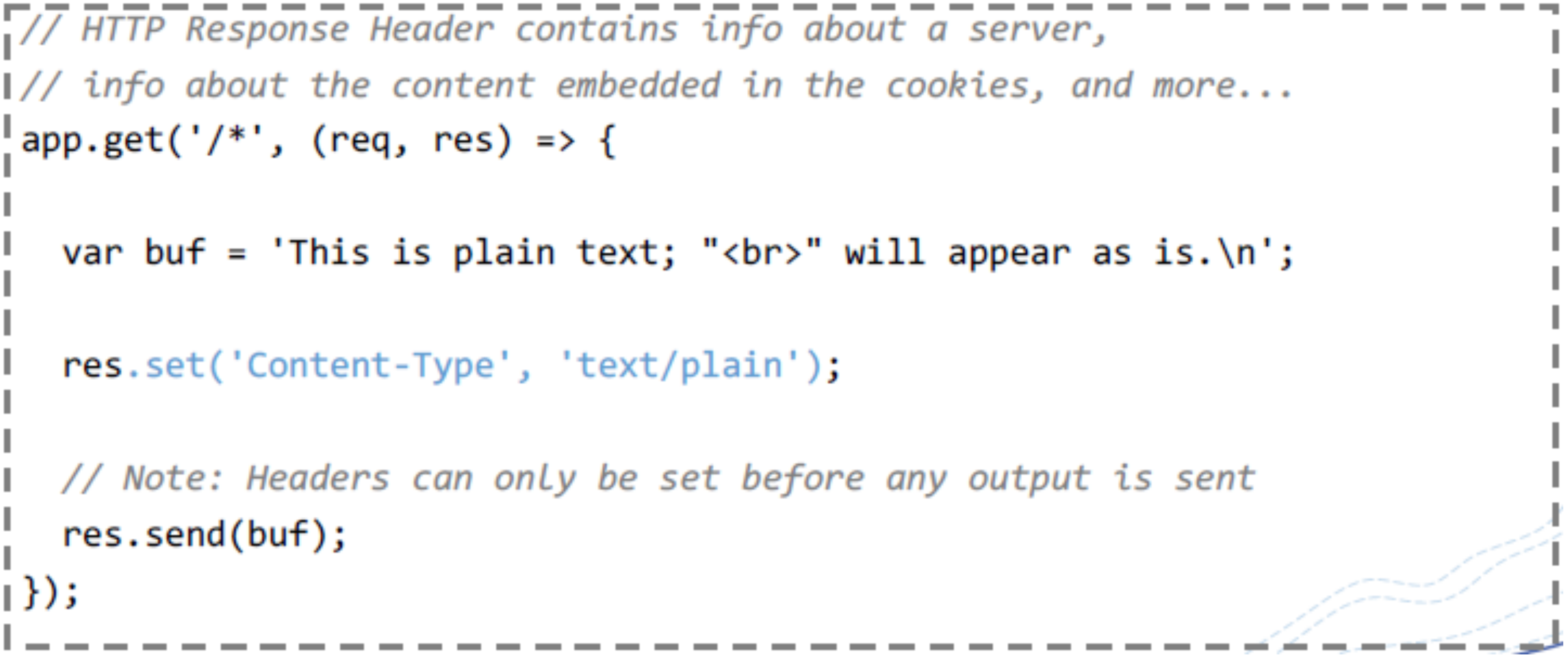


```
// HTTP Request Header contains info about a client,
// info about the content embedded in the body, cookies, and more...
app.get('/*', (req, res) => {
  // Header fields in the request found as properties in req.headers
  console.log( req.headers );

  // Helper function to get the value of a specific header with header
 // name case-insensitive; returns undefined if it does not exist
  console.log( req.get('user-agent') );

});
```

Extract headers : req.headers

Setting response headers

There is header for both request and response

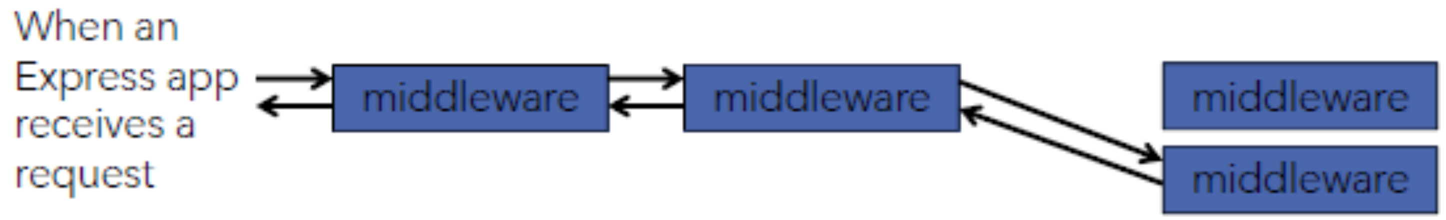


```
// HTTP Response Header contains info about a server,
// info about the content embedded in the cookies, and more...
app.get('/*', (req, res) => {

  var buf = 'This is plain text; "<br>" will appear as is.\n';

  res.set('Content-Type', 'text/plain');

  // Note: Headers can only be set before any output is sent
  res.send(buf);
});
```

Middleware and routing

- Middleware is a function in the form of:

```
function (req, res, next) { ... }
```



- An Express application is essentially a series of middleware calls
- Routing – defining how middleware(s) are used to handle a request (i.e., without *next*)

Built-in middleware

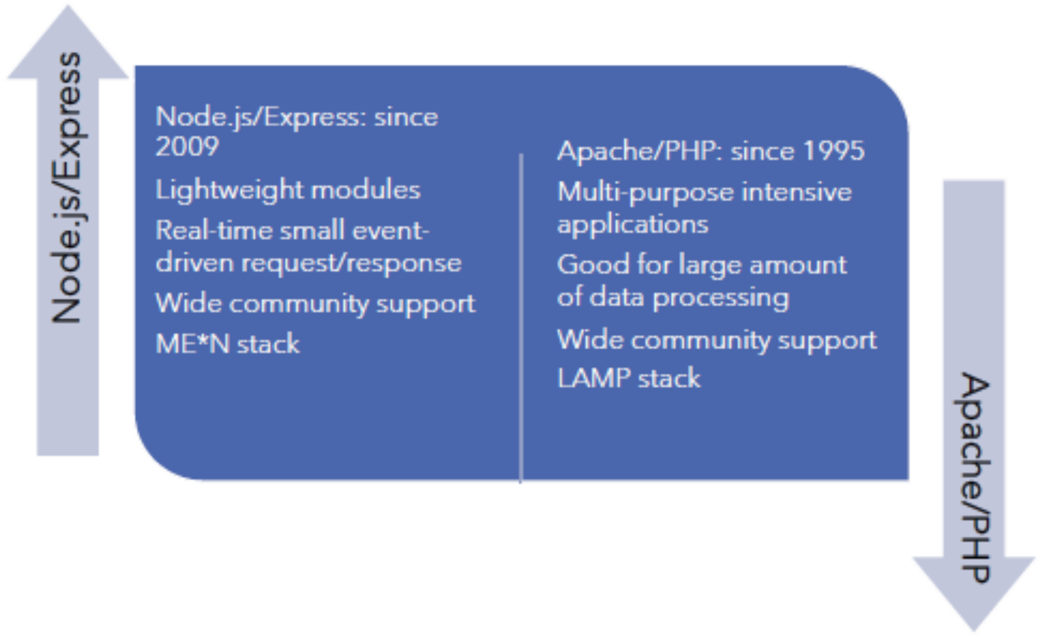
- These middleware functions come with Express:
  - **express.static()**
    - For serving static files
  - **express.json()**
    - For parsing JSON in incoming requests
  - **express.urlencoded()**
    - For parsing URL-encoded contents
- Third-party middleware can be loaded using **require()**
- See: https://expressjs.com/en/guide/using-middleware.html Useful

Designing URL

- The URL of a page to show the detailed view of an item (with a specific ID) can be represented as: http://domain/show_item?id=123456789
  - i.e., representing the ID as a name-value pair in query string
    - Query parsing in **req.query[]**
- Or as: http://domain/show_item/123456789
  - i.e., embedding the ID in a particular path fragment
    - Route parameters parsing in **req.params[]**
- What is the difference between two designs?
  - User experience

Node.JS vs Others

- See: https://hackernoon.com/nodejs-vs-php-which-is-better-for-your-web-development-he7oa24wp
- Node.js is written in JavaScript, while most of the other server-side technologies are written in C or C++.


Node.js/Express
Node.js/Express: since 2009
Lightweight modules
Real-time small event-driven request/response
Wide community support
ME*N stack
Apache/PHP: since 1995
Multi-purpose intensive applications
Good for large amount of data processing
Wide community support
LAMP stack
Apache/PHP

Node.JS vs Others

- Express is only one of the implementations of web servers on Node.js
  - Koa, Hapi, Nest.js, Sails.js, Meteor, ……
- With React, there are also other possibilities:
  - **create-react-app** runs a web server automatically with **npm start**, to show the React app in development mode
  - For static deployment, the server **serve** can be used
  - See: https://create-react-app.dev/docs/deployment/

Further reading

- Express routing guide:
  - https://expressjs.com/en/guide/routing.html

- Express 4 APIs:
  - https://expressjs.com/en/4x/api.html