香港中文大學
The Chinese University of Hong Kong

NODE.JS WITH EXPRESS

CSCI2720 2022–23 Term 1

Building Web Applications

Dr. Chuck-jee Chau *and previous contributors*
chuckjee@cse.cuhk.edu.hk

OUTLINE

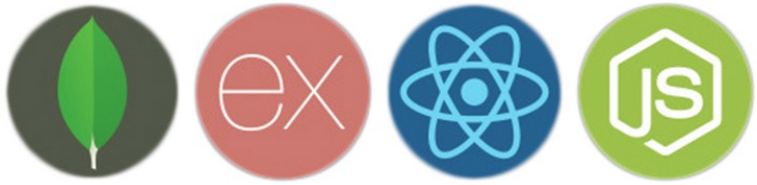
- Overview of Node.js
- Express Basics
  - Routing
  - Retrieving data from query string (GET) and from body (POST)
  - Generating content of a response
  - Retrieving and setting header fields from a request
  - Retrieving and setting cookie and sessions

NODE.JS AND EXPRESS

- ***Node.js*** – A JavaScript run-time environment
  - was first released in 2009
  - makes writing servers (including *web servers*) easier
  - runs JavaScript on server side (using Chrome V8 engine)
- Node.js 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



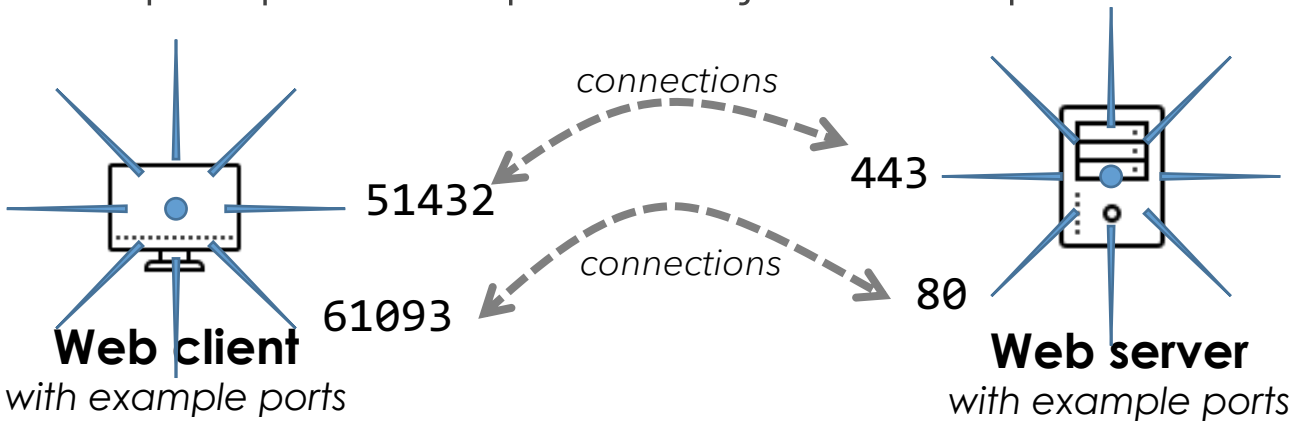
M E R N

THE stack!

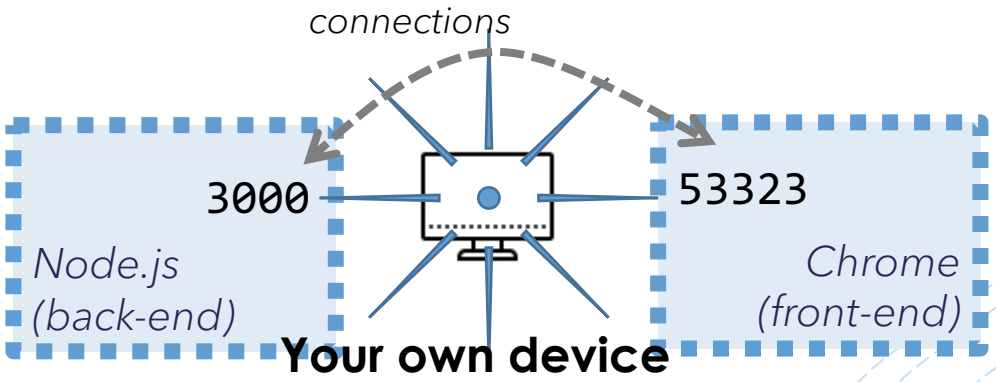
See: https://blog.hyperiondev.com/index.php/2018/09/10/everything-need-know-mern-stack/

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
  1. Routing
     - Deciding the actions to take based on URL and HTTP method
  2. Retrieve data from an HTTP request
  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 URI 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/*
- *Note*: 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.com* 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` *(and accept default answers)*

  - 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 *http://expressjs.com/en/starter/installing.html*

HELLO 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

https://stackblitz.com/edit/chuckjee-node-helloworld

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

// Assign a callback function to handle ALL requests
app.all('/*', function (req, res) {
  // When this callback function is called, send this to client
  res.send('Hello World!');
});

// Set the web server to listen to port 3000 (can be any port)
const server = app.listen(3000);
```

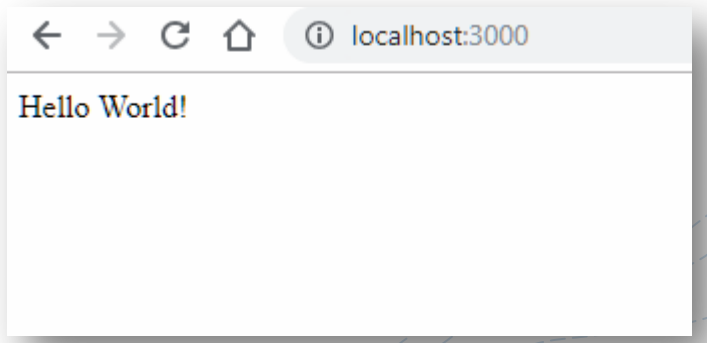
`req` is an object representing the current HTTP request
`res` is an object representing the current HTTP response

HELLO WORLD!

- Then, 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 the same machine, access **`http://localhost:3000`**
- *See: https://expressjs.com/en/starter/hello-world.html*



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 one of **`get`**, **`post`**, **`put`**, **`delete`**, **`all`**
- The route path can be strings, string patterns, 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

https://stackblitz.com/edit/chuckjee-node-param

```
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('/*', ...)
```

ROUTE PATH

https://stackblitz.com/edit/chuckjee-node-param

```
// Exact match (match 'index' respectively)
app.all('/index', (req, res) => res.send("Looking for index?"));

// String patterns matching
// '?': this character/string can exist or not
app.all('/csci?2720', (req, res) => res.send("csci2720 or csc2720?"));
app.all('/c(sci)?2720', (req, res) => res.send("csci2720 or c2720?"));

// '+': this character/string can occur multiple times
app.all('/cu+hk', (req, res) => res.send(" cuhk or cuuhk or cuuuuuuhk?"));

// '*': any character/string
app.all('/dir1/*', (req, res) => res.send("This is something in dir1"));
```

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"}
```

https://stackblitz.com/edit/chuckjee-node-param

GENERATING FILE CONTENT DYNAMICALLY

https://stackblitz.com/edit/chuckjee-node-param

```
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!
});
```

SERVING STATIC FILES

- **`res.sendFile()`** transfers the file at the given ***absolute path***
- It sets the **`Content-Type`** response HTTP header field based on the filename 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
});
```

- *See: https://expressjs.com/en/4x/api.html#res.sendFile*

SERVING 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'));
```

GET PARAMETERS FROM A QUERY STRING

https://stackblitz.com/edit/chuckjee-node-getpost

```
// 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);
});
```

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

POST PARAMETERS IN REQUEST BODY

```
// 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)
});
```

https://stackblitz.com/edit/chuckjee-node-getpost

Note: Try this using Postman!

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') );

});
```

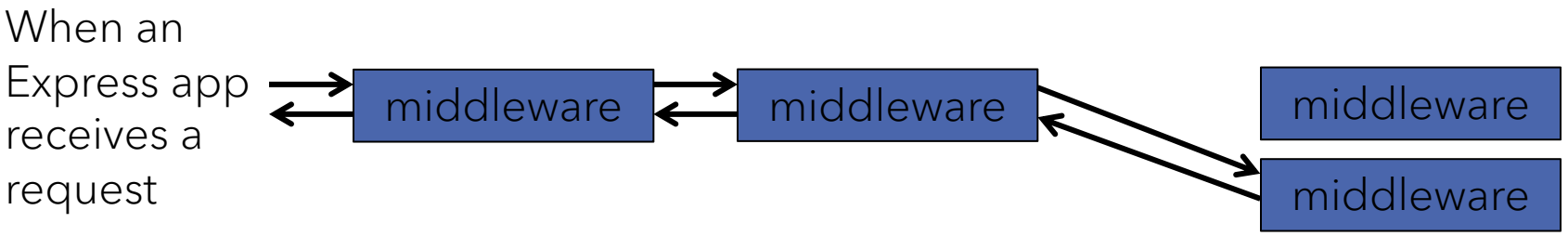
SETTING RESPONSE HEADERS

```
// 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

```
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

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: http://expressjs.com/guide/using-middleware.html*

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 these two designs?

NODE.JS/EXPRESS VS. OTHER TECHNOLOGIES

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

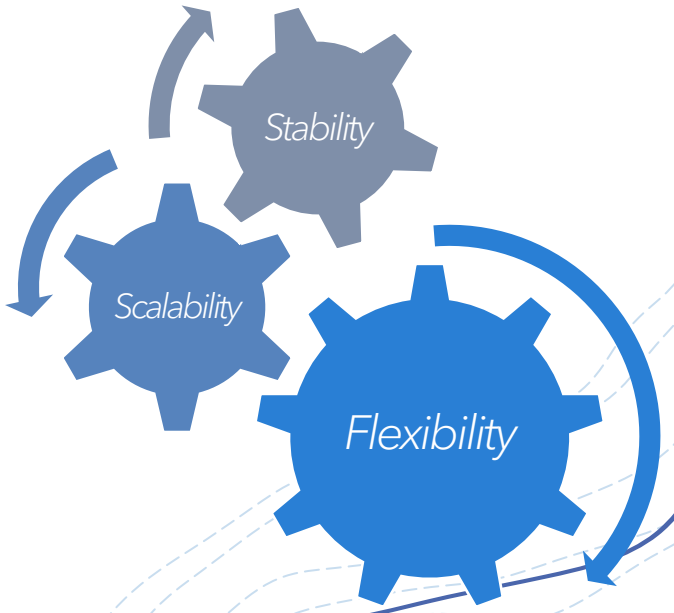
Multi-purpose intensive applications

Good for large amount of data processing

Wide community support

LAMP stack

Apache/PHP



See: https://hackernoon.com/nodejs-vs-php-which-is-better-for-your-web-development-he7oa24wp

OTHER WEB SERVERS ON NODE.JS

- Express is only one of the implementations of web servers on Node.js
- 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/*

READ FURTHER…

Express 4 APIs

http://expressjs.com/4x/api.html

Express Routing

http://expressjs.com/guide/routing.html