

Node.js and Express

2024/25 COMP3322 Modern Technologies on WWW

Content

- Introduction to Node.js
- How do Node.js and Express work?
- Using Express
 - Routing
 - Request object and Response object
 - Middleware
 - Serving static files
- Using template engine - Pug
- Cookies and Sessions

Node.js

- Node.js is an open-source, cross-platform, runtime environment that allows developers to create **server-side** applications in JavaScript.
- It is available on Microsoft Windows, macOS, Linux, Solaris, FreeBSD, etc.
- Node.js has an event-driven architecture capable of asynchronous I/O.
 - This is the major difference between Node.js and PHP.
 - Most functions in **PHP are blocked until completion**, while Node.js functions are **non-blocking** (i.e. asynchronous) and use callbacks to signal completion or failure.
 - These design choices aim to optimize throughput and scalability in web applications with many input/output operations.

Differences between Node.js and Browser

- In Node.js, there are no DOM, no UI, and all the other objects that are provided by the browser.
 - There is no window object, the global object is renamed to "global".
- Unlike the browser where JavaScript is sandboxed for safety purpose, node.js has full access to the system like any other native application.
 - This means never run untrusted JavaScript in node.js.
- We can use both `require()` and `import` in Node.js to load in a module, while we are limited to `import` in the browser.

Global Variables

- When you're in the global scope of a browser, using **var** define a global variable (under the Window object)
- In Node.js, a variable declared by **var** is limited to that module only, i.e., not be available by the top-level scope of the program
- If you want variables to be global (i.e. across modules), you can declare them specifically to the global object
 - `global.greeting = "Hello World";`
 - `console.log(greeting); //Hello World`

Node.js

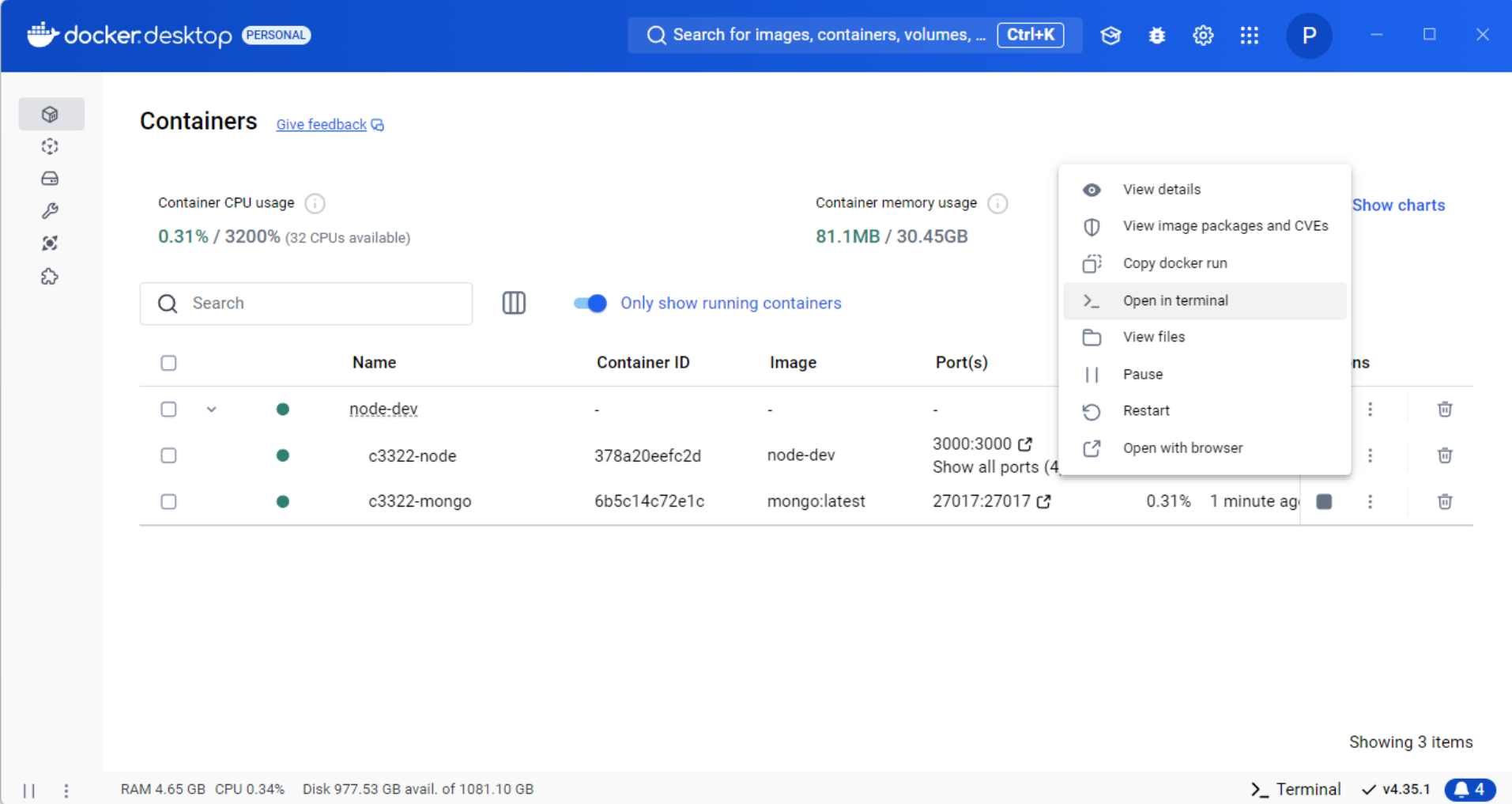
- Node.js on its own does not provide a lot of features to support web-development.
- If you want to add specific handling for different HTTP requests (e.g. GET, POST, DELETE, etc.), handle requests at different URL paths, dynamically create the response, then you have to write the code yourself, or you can avoid reinventing the wheel and **use a web framework**.
 - Express is the most popular Node web framework.
- The node package manager (**NPM**) is the pre-installed package manager for the Node.js server platform.
 - It serves two functions: installing packages and managing dependencies.
 - It provides access to hundreds of thousands of reusable packages.

Install Node on your computer (Not recommended)

- https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/development_environment#Installing_Node
- For Windows and macOS
 - Download the installer directly from node.js site:
 - <https://nodejs.org/en/download/>
- For Ubuntu 20.04/22.04 (or on Windows 10/11 Linux subsystem)
 - Don't install directly from the normal Ubuntu repositories
 - Open a terminal and run the following commands:

```
curl -sL https://deb.nodesource.com/setup_22.x | sudo -E bash -  
sudo apt-get install -y nodejs
```

Use the Course's Docker



The screenshot shows the Docker Desktop application window. The top bar is blue with the Docker logo, 'docker.desktop PERSONAL', a search bar, and icons for images, containers, settings, and a profile icon. The left sidebar contains icons for containers, images, volumes, and settings. The main area is titled 'Containers' and shows system usage: 'Container CPU usage 0.31% / 3200% (32 CPUs available)' and 'Container memory usage 81.1MB / 30.45GB'. Below this is a search bar and a toggle for 'Only show running containers'. A table lists three running containers: 'node-dev', 'c3322-node', and 'c3322-mongo'. A context menu is open over the 'c3322-mongo' container, showing options like 'View details', 'View image packages and CVEs', 'Copy docker run', 'Open in terminal', 'View files', 'Pause', 'Restart', and 'Open with browser'. The bottom status bar shows system resources: 'RAM 4.65 GB CPU 0.34% Disk 977.53 GB avail. of 1081.10 GB', a 'Terminal' button, version 'v4.35.1', and a notification badge with the number '4'.

docker.desktop PERSONAL

Search for images, containers, volumes, ... Ctrl+K

Containers [Give feedback](#)

Container CPU usage ⓘ
0.31% / 3200% (32 CPUs available)

Container memory usage ⓘ
81.1MB / 30.45GB

Search

Only show running containers

	Name	Container ID	Image	Port(s)	
<input type="checkbox"/>	node-dev	-	-	-	
<input type="checkbox"/>	c3322-node	378a20eefc2d	node-dev	3000:3000 Show all ports (4)	
<input type="checkbox"/>	c3322-mongo	6b5c14c72e1c	mongo:latest	27017:27017	0.31% 1 minute ago

- View details
- View image packages and CVEs
- Copy docker run
- Open in terminal
- View files
- Pause
- Restart
- Open with browser

Showing 3 items

RAM 4.65 GB CPU 0.34% Disk 977.53 GB avail. of 1081.10 GB

Terminal v4.35.1 4

Similar to import but only available in Node

A Simple Node Program

```
//Load HTTP module
const http = require("http");
const hostname = '0.0.0.0';
const port = 3000;

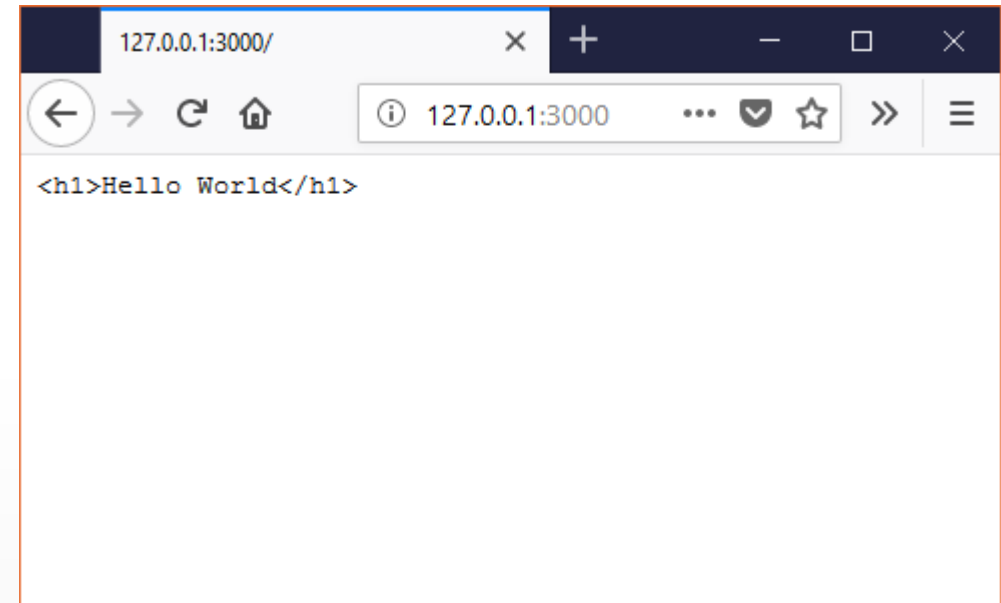
//Create HTTP server and listen on port 3000 for requests
const server = http.createServer((req, res) => {

  //Set the response HTTP header with HTTP status and Content type
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('<h1>Hello World</h1>\n');
});

//listen for request on port 3000, and as a callback function have
//the port listened on logged
server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

imports the "http" module

```
> node hellonode.js
Server running at http://0.0.0.0:3000/
```



```
<html>
  <head></head>
  <body>
    <pre><h1>Hello World</h1></pre>
  </body>
</html>
```

How do Node.js+Express work?

- A web server can be seen as a function that takes in a HTTP request and outputs an HTTP response.
- To do so, the web server needs to
 - know **the type** of request (GET, POST, HEAD, etc.).
 - know the request **resource** (identify by **the path, query string**, etc).
 - perform some **computation** (according to the request and resource).
 - generate and return the response.

How do Node.js+Express work?

- Express provides mechanisms to:
 - Specify which functions are called for requests with different HTTP **request verbs** (e.g., GET & POST) at different URL **route paths** - routing.
 - **Specify which template** ("view") **engine** is used, where are the template files located, and which templates to use to render the response.
 - Set common web application settings like the port to use for connecting.
 - Specify where to **find the static files**, e.g. CSS files and image files.
 - Use any **database mechanism** supported by Node.

How do Node.js+Express work?

- Express provides mechanisms to:
 - Add request processing "middleware" at any point within the **request handling pipeline**.
 - **Middlewares are functions** executed in the middle of the request handling pipeline
 - Middleware function may produce an output which could be the final output or could be used by the next middleware until the cycle is completed.
 - Thus, to serve a request, we may have more than one middleware and they execute one after the other in the order to generate the output.
 - There are middlewares for cookies, sessions, getting POST/GET parameters, etc.

Middleware

- As implied by the name, Middleware is a function appears **in the middle** between an initial request and final handler.
- Middleware is commonly used to perform tasks like **body parsing** for URL-encoded or JSON requests, **cookie parsing** for basic cookie handling, enable CORS requests, or even building JavaScript modules on the fly.
- All middleware functions have access to the **request object** (req), the **response object** (res), and the **next middleware** function in the request handling pipeline.
- Middleware functions are always invoked **in the order** in which they are appeared in the program.

Middleware

- A middleware function can perform the following tasks:
 - It can **execute** any code.
 - It can make **changes to** the **request** and the **response objects**.
 - It can **end** the request-response pipeline.
 - It can **call** the next middleware function in the pipeline.
- If the function is not the end of the pipeline, it must call **next()** to pass control to the next function.

A Simple Express Example

creates an express app

imports the "express" module

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

app.get('/', (req, res) => {
  res.send('<h1>Hello World!</h1>')
});

app.listen(8000, () => {
  console.log('Example app listening on port 8000!')
});
```

The app.get() function only responds to HTTP GET requests with the specified URL path ('/'). – Specify the "route"

```
> node index.js
Example app listening on port 8000!
```



A middleware function is defined for handling GET request. The callback takes a request object and a response object as arguments.

A Simple Express Example

creates an express app

imports the "express" module

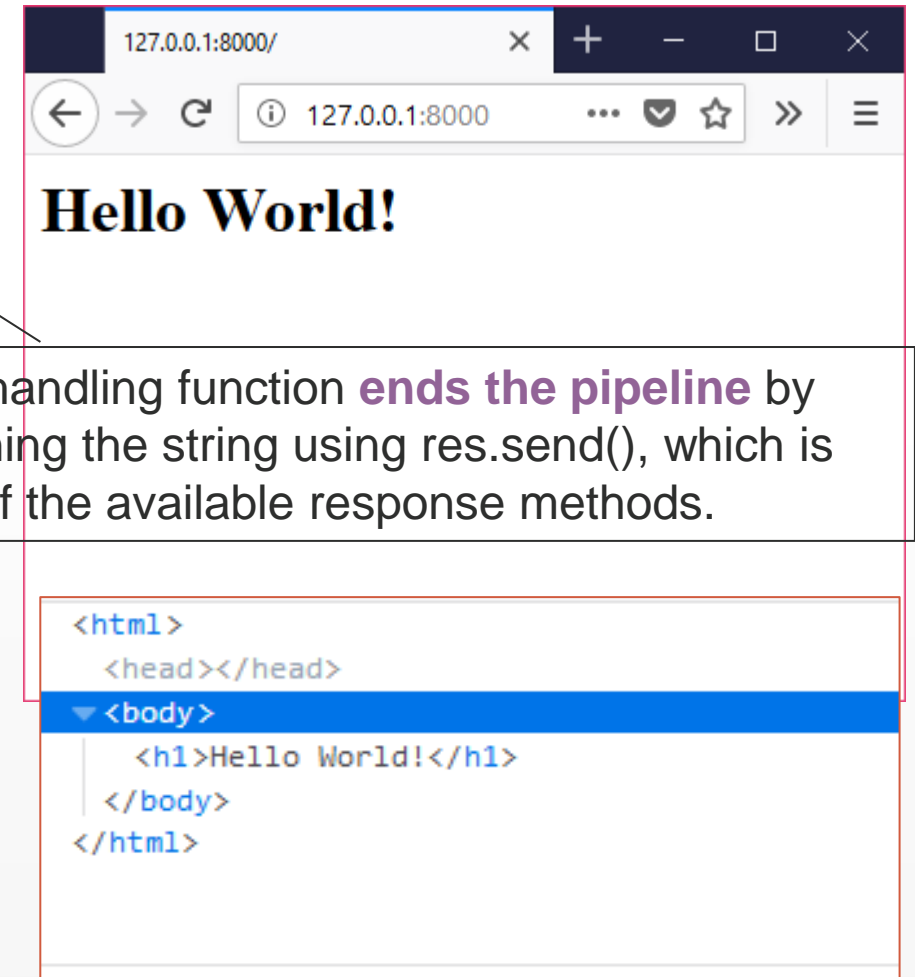
```
const express = require('express')
const app = express();

app.get('/', (req, res) => {
  res.send('<h1>Hello World!</h1>')
});

app.listen(8000, () => {
  console.log('Example app listening on port 8000!')
});
```

This starts a server listening for connection at port 8000.

```
> node index.js
Example app listening on port 8000!
```



Using Express

- To run the simple express program, you have to execute the following steps in a terminal:
 - Create a project folder for your application
 - Use npm to initialize the project
 - Install express to your project
 - The default is to install express 4.x
 - Create a file name index.js and copy the code to it
 - Start the server
- Or we can install the express generator and use it to generate the application skeleton.
 - `npm install express-generator -g` (already been installed on our docker)
 - the `-g` flag installs the tool globally in the system so that you can call it from anywhere
 - You will learn about the use of express generator in Workshop 5.

```
mkdir myapp  
cd myapp
```

```
npm init  
npm install express
```

```
node index.js
```

Routing

- Routing refers to how the system responds to a client request **to a particular endpoint** – **a path** and a specific **HTTP request** (GET, POST, etc).
- Assume an instance of express is created with the name app. `const app = express();`
- Route definition takes the following structure:
 - `app.METHOD(PATH, HANDLER)`

```
app.get('/', function (req, res) {  
  res.send('<h1>Hello World!</h1>')  
})
```

```
app.put('/user', function (req, res) {  
  res.send('Got a PUT request at /user')  
})
```

```
app.post('/', function (req, res) {  
  res.send('Got a POST request')  
})
```

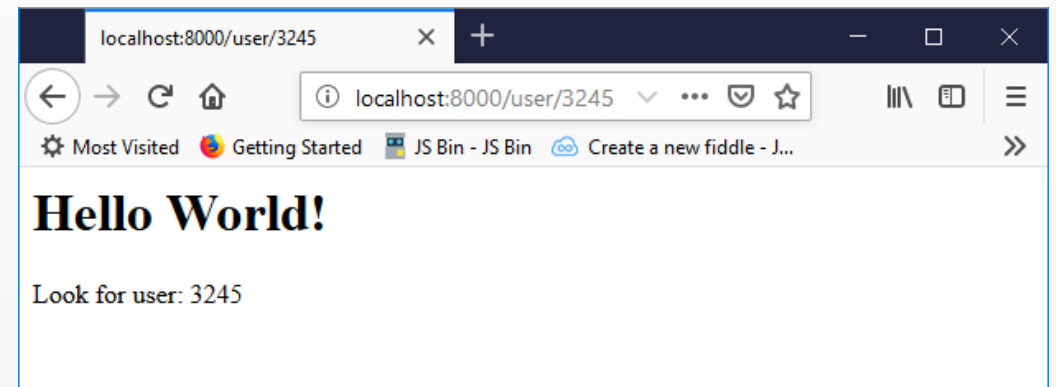
```
app.delete('/user', function (req, res) {  
  res.send('Got a DELETE request at /user')  
})
```

app.METHOD(PATH, HANDLER [,HANDLER . . .])

Route Handlers and Route parameters

- For the handler, it could be:
 - A **single** callback function
 - **More than** one callback function; separated by commas
 - An **array of** callback functions
 - If a callback function is not the end of the chain, it must call **next()** to pass control to the next function.
- We can capture the value(s) **at a specific** position(s) in the **URL path** as **Route Parameter**(s).
 - The captured values are populated in the **req.params** object, with the name of the route parameter specified in the path as their respective keys.
 - Now we can use the same set of route handlers to process different paths under the same path tree.
 - e.g. '/user/1234', '/user/2468'

```
app.get('/user/:userid',  
  (req, res, next) => {  
    let who = req.params.userid;  
    res.locals.rdata = 'Look for user: '+who;  
    next();  
  },  
  (req, res) => {  
    let msg = '<h1>Hello World!</h1>';  
    msg += res.locals.rdata;  
    res.send(msg);  
  }  
);
```



Request and Response Objects

- The req object represents the received **HTTP request**. Here are some properties that can be accessed:

Properties	Description
req.app	Returns a reference to the current app instance.
req.baseUrl	It specifies the URL path on which a router instance was mounted.
req.cookies	When we use cookie-parser middleware, this property is an object that contains cookies sent by the request.
req.hostname	It contains the hostname from the "host" http header.
req.ip	It specifies the remote IP address of the request.
req.method	Returns the HTTP method of the request
req.originalUrl	This property holds the original request URL.
req.params	An object containing properties mapped to the named route parameters .
req.path	It contains the path part of the request URL.
req.protocol	The request protocol string, "http" or "https" when requested with TLS.
req.query	An object containing a property for each query string parameter in the route.

Request and Response Objects

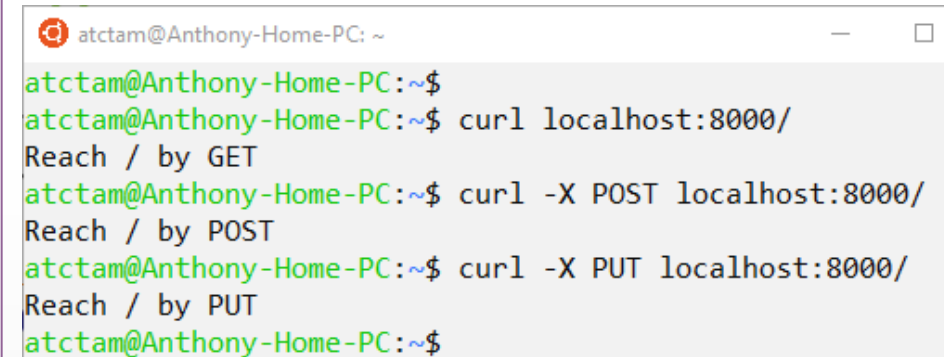
- The res object represents the **HTTP response** that an Express app is going to send.

Properties /Methods	Description
res.app	Returns a reference to the current app instance.
res.headersSent	A Boolean property that indicates if the app has sent the HTTP headers for the response.
res.locals	It specifies an object that contains response local variables scoped to the request.
res.append()	Appends the specified value to the HTTP response header field.
res.cookie()	Sets the HTTP Set-Cookie header with the options provided.
res.clearCookie()	Clears the cookie specified by name and given options.
res.end()	Use to quickly end the response without any data.
res.get()	This method gets HTTP response header specified by field.
res.json()	This method returns the response in JSON format .
res.redirect()	Redirects to the URL derived from the specified path, with specified status.
res.render()	Renders a view and sends the rendered HTML string to the client.
res.send()	Sends the HTTP response body .
res.sendStatus()	Sets the response HTTP status code to statusCode and send its string representation as the response body.
res.set()	Sets the response's HTTP header field to input value.
res.status()	Sets the HTTP status for the response.

Routing

- We can create **chainable route handlers** for a **single route path** for different HTTP requests by **using `app.route()`**

```
app.route('/')
  .get( (req, res) => {
    res.send("Reach "+req.originalUrl+" by "
+req.method);
  })
  .post( (req, res) => {
    res.send("Reach "+req.originalUrl+" by "
+req.method);
  })
  .put( (req, res) => {
    res.send("Reach "+req.originalUrl+" by "
+req.method);
  });
```



```
atctam@Anthony-Home-PC: ~
atctam@Anthony-Home-PC:~$ curl localhost:8000/
Reach / by GET
atctam@Anthony-Home-PC:~$ curl -X POST localhost:8000/
Reach / by POST
atctam@Anthony-Home-PC:~$ curl -X PUT localhost:8000/
Reach / by PUT
atctam@Anthony-Home-PC:~$
```

Routing

- We can use the `express.Router` class to create modular, **mountable** route handlers.
- Adv:
 - Organize the routings in separate files.
 - This router module can be used to handle requests to different paths of **similar structure**.

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

var tryRouter = require('./tryRouter');

app.use('/c3322', tryRouter);
app.use('/c3230', tryRouter);

app.listen(8000, () => {
  console.log('Example app listening on port 8000!')
});
```

tryRouter.js

```
const express = require('express')
const router = express.Router();

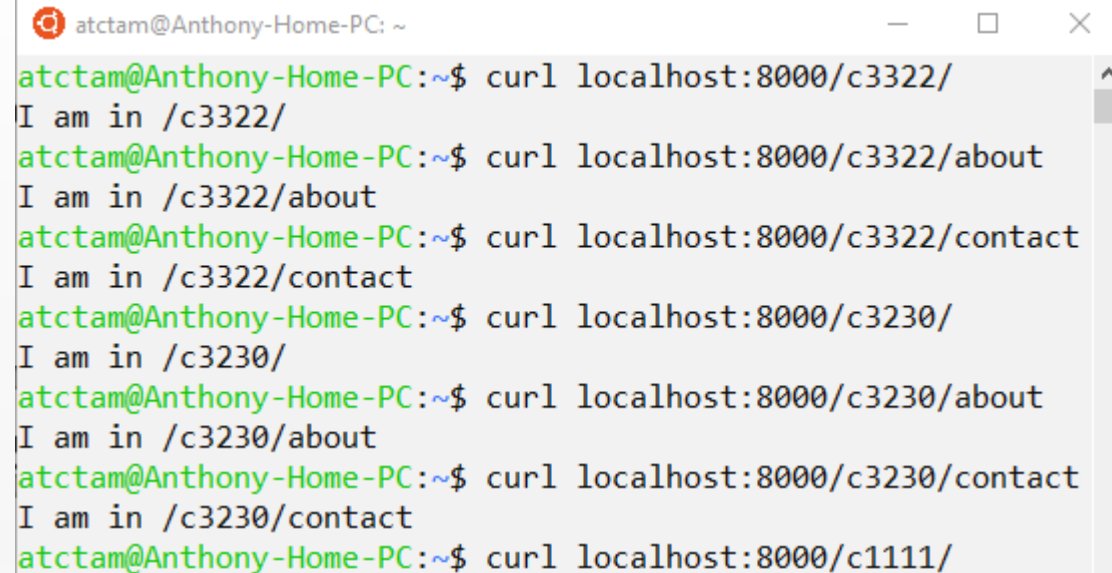
function output(req, res) {
  res.send("I am in " + req.originalUrl);
}

router.get('/', output);

router.get('/about', output);

router.get('/contact', output);

module.exports = router;
```



```
atctam@Anthony-Home-PC: ~
atctam@Anthony-Home-PC:~$ curl localhost:8000/c3322/
I am in /c3322/
atctam@Anthony-Home-PC:~$ curl localhost:8000/c3322/about
I am in /c3322/about
atctam@Anthony-Home-PC:~$ curl localhost:8000/c3322/contact
I am in /c3322/contact
atctam@Anthony-Home-PC:~$ curl localhost:8000/c3230/
I am in /c3230/
atctam@Anthony-Home-PC:~$ curl localhost:8000/c3230/about
I am in /c3230/about
atctam@Anthony-Home-PC:~$ curl localhost:8000/c3230/contact
I am in /c3230/contact
atctam@Anthony-Home-PC:~$ curl localhost:8000/c1111/
```

Route Paths

<https://expressjs.com/en/4x/api.html#path-examples> <https://expressjs.com/en/5x/api.html#path-examples>

- Query strings are not part of the route path.
- In addition to Route Parameters (slide# 19), we can represent **Route paths** as strings, string patterns, or regular expressions.
- Here are some examples of route paths based on string patterns:

'/' – for 4.x, could match any path start from root	'/*xxx' – for 5.x, could match any path start from root (where xxx is any variable name)
	'/{*xxx}' – for 5.x, could match any path start from root or '/'
'/ab?cd' – for 4.x, could be '/acd' and '/abcd'	'/a(b?)cd' – for 5.x could be '/acd' and '/abcd'
'/ab+cd' – for 4.x, possible matches are: '/abcd', '/abbcd', '/abbbcd', . . .	(No longer support in 5.x)
'/ab*cd' – possible matches are: '/abcd', '/abxyzcd', '/ab12vn34cd', . . .	(No longer support in 5.x)

- In addition, we can define an array of path strings, path patterns, & regular expression patterns that use the same (set of) handler function(s)

```
app.get(['/greet', '/hello', /^\/abc[0-9]?$/], (req, res) => { . . . . . })
```


'/*' – for 4.x, could match any path start from root

'/*xxx' – for 5.x, could match any path start from root (where xxx is any variable name)

'/{*xxx}' – for 5.x, could match any path start from root or '/'

'/ab?cd' – could be '/acd' and '/abcd'

'/a(b?)cd' – for 5.x could be '/acd' and '/abcd'

'/ab+cd' – possible matches are: '/abcd', '/abbcd', '/abbbcd', . . .

(No longer support in 5.x)

'/ab*cd' – possible matches are: '/abcd', '/abxyzcd',
'/ab12vn34cd', . . .

(No longer support in 5.x)

Routing

app.all(PATH, HANDLER [,HANDLER . . .])

- There is a **special routing** method, app.all(), used to load middleware functions at a path **for all HTTP request** methods.
- For example, the following callback is executed for requests to /secret whether using GET, POST, PUT, DELETE, or any other HTTP request method

```
app.all('/secret', (req, res, next) => {  
  console.log('Accessing the secret section ...');  
  next(); // pass control to the next handler  
})
```

Application-Level Middleware

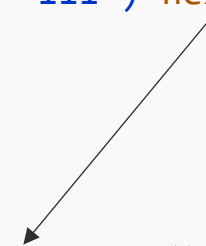
- We can use `app.use()` and `app.METHOD()` functions to **bind** application-level middleware to the **app object**.
- `app.use()` could be use with or without a mount path.

```
app.use(function () {}) //Is executed every time the app receives a request
```

```
app.use('/someroute', function() {}) //Added to a specific path; including subpaths
```

- Like `app.METHOD()`, `app.use()` can have one or more middleware functions.
- To **skip** the rest of the middleware functions from a router **sub-stack**, call **`next('route')`** to pass control to the next route.

```
app.get('/user/:id', function () {  
  if (req.params.id == '111') next('route');  
  else next();  
}, function () {  
  res.send("Normal");  
});  
  
app.get('/user/:id', function () {  
  res.send("Special");  
});
```



Error-handling Middleware

- Express comes with a default error handler that takes care of any errors that might be encountered in the app
- We can define our custom-made error-handling middleware
 - Error-handling middleware always takes **four** arguments.
 - This is to identify it as an error-handling middleware function.
 - Even if you don't need to use the next(), you must specify it to maintain the signature.
 - Define error-handling middleware functions in the same way as other middleware functions like the following:

```
app.use(function (err, req, res, next) {  
  console.error(err.stack);  
  res.status(500).send('Something broke!');  
})
```

Router-Level Middleware

- Similar to application-level middleware, it is bound to an instance of **express.Router()**.

- router.use()
- router.METHOD()

```
//This code is executed for every request to the router
router.use(function (req, res, next) {
  console.log('Time:', Date.now());
  next();
})
```

- We can skip the following middleware functions in a route by using next('route').
- We can even skip the rest of the router's middleware functions, call next('router') to pass control back out of the router instance.

Built-in Middleware

- Express has the following **built-in** middleware functions:
 - **express.static** serves static assets such as HTML files, images, and so on.
 - **express.json** parses incoming requests with JSON payloads.

```
app.use(express.json());
```

 - if request body has a valid JSON string, we can get the JS object via **req.body** property
 - **express.urlencoded** parses incoming requests with URL-encoded payloads.

```
app.use(express.urlencoded());
```

 - if request body has the form data, we can get the JS object via **req.body** property
- A few middleware functions were included in Express before 4.x, you can find the list @ <https://expressjs.com/en/api.html#express>

Static Files

- Express supports serving static files such as images, CSS style files, and JavaScript files.
- We use the `express.static()` middleware function to **specify the root directory** from which the static files are located.
 - For example, use the following code to serve images, CSS files, and JavaScript files in a directory named `public`:

```
app.use(express.static('public'))
```
- Express looks up the files **relative to** the static directory, so **the name of the static directory is not part of the URL**.

```
http://localhost:8000/images/cat.jpg
http://localhost:8000/css/style.css
http://localhost:8000/js/ajax.js
http://localhost:8000/images/logo.png
http://localhost:8000/hello.html
```

```
your-project
|- node_modules ...
|- public
|   |- css
|   |   `-- style.css
|   |- images
|   |   |- logo.png
|   |   `-- cat.jpg
|   |- js
|   |   `-- ajax.js
|   `-- hello.html
`-- index.js
```

Static Files

- We can create a “virtual path” in the URL for serving the static files.
 - A virtual path is the path **not actually exists** in the file system

```
app.use('/static', express.static('public'))
```

```
http://localhost:8000/static/images/cat.jpg  
http://localhost:8000/static/css/style.css  
http://localhost:8000/static/js/ajax.js  
http://localhost:8000/static/images/logo.png  
http://localhost:8000/static/hello.html
```


Demo – Our Account Registration Form

Form Validation

localhost:8000/express-form-ajax.html

Account Registration Form

Student Name:

Student No.:

Age:

Student Email:

2 requests | 2.64 KB / 3.17 KB transferred | Finish: 141 ms

Status	Method	File	Domain	Cause	Type
200	GET	express...	localhost:8000	document	html
404	GET	favicon...	localhost:8000	img	html

Form Validation

localhost:8000/express-form-ajax.html

Account Registration Form

Student Name:

Student No.: You have registered before!!

Age:

Student Email:

5 requests | 2.67 KB / 3.63 KB transferred | Finish: 2.96 min

Status	Method	File	Domain	Cause	Type
200	GET	express...	localhost:8000	document	html
404	GET	favicon...	localhost:8000	img	html
200	GET	checkin...	localhost:8000	xhr	xml
200	GET	checkin...	localhost:8000	xhr	xml
200	GET	checkin...	localhost:8000	xhr	html

Form Validation

localhost:8000/express-form-ajax.html

Account Registration Form

Student Name:

Student No.:

Age:

Student Email:

4 requests | 2.64 KB / 3.41 KB transferred | Finish: 1.98 min

Status	Method	File	Domain	Cause	Type
200	GET	express...	localhost:8000	document	html
404	GET	favicon...	localhost:8000	img	html
200	GET	checkin...	localhost:8000	xhr	xml
200	GET	checkin...	localhost:8000	xhr	xml

```
Select atctam@atctamp: ~/Corsair/c3322/static
atctam@atctamp:~/Corsair/c3322/static$ node index
Example app listening on port 8000!
Got a request: 3015123456
Got a request: 3015123456
Got a request: 3015999999
```

Demo – Our Account Registration Form

```
ajax-express
├── index.js
├── package-lock.json
├── package.json
└── public
    └── express-form-ajax.html
```

express-form-ajax.html – only change from using checking.php to checking

```
function ajaxRequest() {
    ajaxObj.onreadystatechange = ajaxResponse;
    ajaxObj.open('GET', "checking?number="+snum.value, true);
    ajaxObj.send();
}
snum.addEventListener('blur', ajaxRequest);
```

Demo – Our Account Registration Form

This is for locating the static file
– express-form-ajax.html

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

app.use(express.static('public'));

const USERNAME = '3015999999';

app.get('/checking', (req, res) => {
  console.log("Got a request: "+req.query.number);
  if ((req.query.number) && (req.query.number == USERNAME))
    res.send('You have registered before!!');
  else
    res.end();
})

app.listen(8000, () => {
  console.log('Example app listening on port 8000!')
});
```

index.js

```
<?php
define("USERNAME", '3015999999');

if (isset($_GET['number']) && ($_GET['number'] == USERNAME))
{
  echo "You have registered before!!";
} else {
  echo "";
}
?>
```

checking.php

This is for the route:
GET /checking

Using Template Engines

- What is a template engine?
 - A template engine allows you to **define templates** for your web application.
 - The engine **replaces the (JavaScript) variables** in the template with actual values **at runtime** while transforming the template to an actual **HTML file** which is then sent to the client.
- There are several template engines you can use with Express, e.g., **Pug**, Mustache, and EJS.
- Jade (**which is renamed to Pug**) is the default template engine of Express application generator. Of this reason, we shall learn using Pug as the template engine.
- To **install pug** to your project: `npm install pug`

Using Pug

- To use Pug, **include these two lines** in your code.
- Create a Pug template file, e.g., hello.pug.

```
html
  head
    title= title
  body
    h1= message
    <p>Interesting!!!</p>
```

- Then create a route to **render** the hello.pug file.

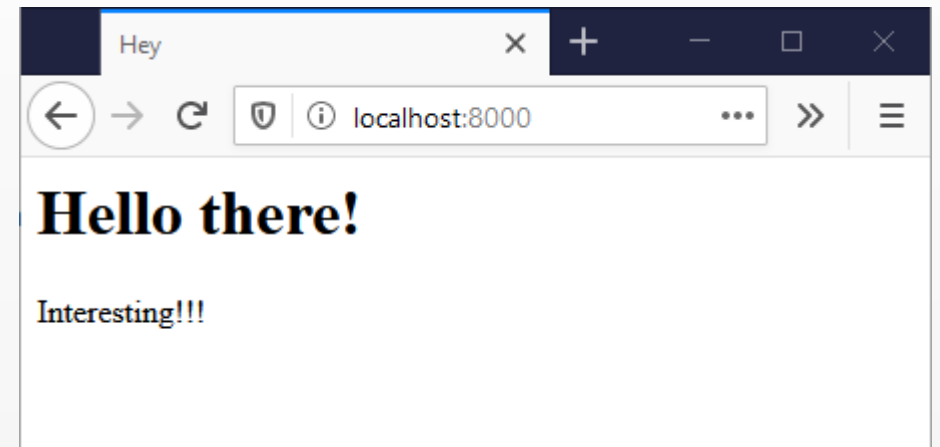
```
app.get('/', function (req, res) {
  res.render('hello',
    { title: 'Hey', message: 'Hello there!' })
})
```

To use Pug template

```
app.set("view engine", "pug");
app.set("views", "./views");
```

Set the directory where the
template files are located

Render the view **hello.pug**



Learning Pug

- Pug relies on **indentation** to describe the structure of the template. There are **no closing tags**.

p Hello World  <p>Hello World</p>

p
| Hello <i>World</i>  <p>Hello <i>World</i></p>

Literal
HTML


 li Item A
 li Item B
 li Item C



 Item A
 Item B
 Item C

//- This is a pug comment
// This is for html  <!-- This is for html-->
p This is a paragraph <p>This is a paragraph</p>

Learning Pug - Attributes

`a(href='google.com') Google`



`Google`

`input(
 type='checkbox'
 name='milk'
 checked
)`



`<input type="checkbox" name="milk" checked="checked" />`

`a(style={color: 'red', background: 'green'})`



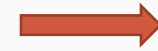
``

`p.button`



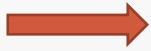
`<p class="button"></p>`

`a#link`



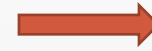
``

`.content`



`<div class="content"></div>`

`#content`



`<div id="content"></div>`

Learning Pug – Adding Code

Code starts with '-' is not added to the output

```
- for (var x = 0; x < 3; x++)  
  li= 'item'+x
```

```
<li>item0</li>  
<li>item1</li>  
<li>item2</li>
```

With '=', JavaScript expression is evaluated and output

```
ul  
  each val in [1, 2, 3, 4, 5]  
    li= val
```

```
<ul>  
  <li>1</li>  
  <li>2</li>  
  <li>3</li>  
  <li>4</li>  
  <li>5</li>  
</ul>
```

```
- var n = 0;  
ul  
  while n < 4  
    li= n++
```

```
<ul>  
  <li>0</li>  
  <li>1</li>  
  <li>2</li>  
  <li>3</li>  
</ul>
```


Learning Pug – Adding Code

```
html
  body
    h1 Hello World!

    if user != null
      p Hi there, #{user}!
      p= "Hi there, "+user+"!"
    else
      p Hi there, unknown person!
```

By default, the contents **after a tag** are assumed to be **a string**, but **the code in between #{ and }** is evaluated.

With **=**, this indicates that the contents **after that tag** is a **JavaScript expression** instead.

In principle, we can pass a function expression to the template!!

```
app.get('/', function (req, res) {
  res.render('test1',
    {user: 'Thomas'})
})
```

Hello World!

Hi there, Thomas!

Hi there, Thomas!

```
app.get('/', function (req, res) {
  res.render('test1')
})
```

Hello World!

Hi there, unknown person!

Learning Pug – Including External Files

We can insert the contents of another Pug file into current one.

```
//- index.pug views/index.pug
doctype html
html
  include head.pug
  body
    h1 My Web Page
    p Welcome to my boring web site.
    #place
    include foot.pug
    script
      include ../public/js/other.js
```

```
//- head.pug views/head.pug
head
  title My Homepage
  script(src='/js/jquery-3.3.1.js')
  <link rel="stylesheet" type="text/css" href="/style/style.css">
```

```
//- foot.pug views/foot.pug
footer#footer
  p Copyright (c) fool
```

```
h1 {
  color: green;
}
p {
  font-weight : bold;
}
div {
  background-color: yellow;
  width: 50%;
}
```

public/style/style.css

```
$(document).ready( () => {
  $('#place').html("<p>Dynamically added paragraph</p>");
})
```

public/js/other.js

```
morePug/
├── index.js
├── node_modules
├── package-lock.json
├── package.json
├── public
│   ├── js
│   │   ├── jquery-3.3.1.js
│   │   └── other.js
│   └── style
│       └── style.css
└── views
    ├── foot.pug
    ├── head.pug
    └── index.pug
```

Learning Pug – Including External Files

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

index.js

```
app.use(express.static('public'));
app.set("view engine", "pug");
app.set("views", "views");
```

```
app.get('/', (req, res) => {
  res.render('index');
})
```

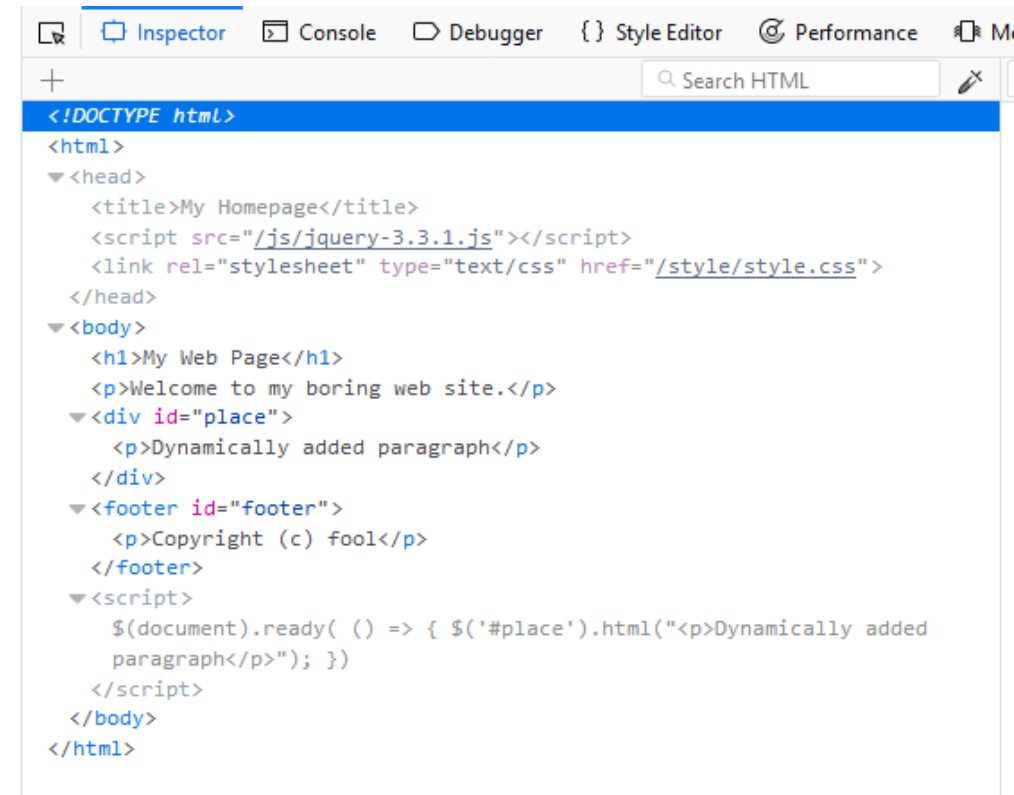
```
app.listen(8000, () => {
  console.log('Example app list');
});
```

My Web Page

Welcome to my boring web site.

Dynamically added paragraph

Copyright (c) fool



Learning Pug – Template Inheritance

- Template inheritance allows you to build a **base pug template** that *contains all the common elements* of your site and **defines blocks** that **child templates can override**.
- In Pug, template inheritance works via the **block** and **extends** keywords.
- In a template, a **block** is simply a “block” of Pug that a child template may replace.

Learning Pug – Template Inheritance

```
// - layout.pug
html
  head
    title My Site - #{title}
    block scripts
      script(src='/jquery.js')
  body
    block content
    block foot
      #footer
        p some footer content
```

Pug blocks can provide default content, if appropriate.

```
// - pageA.pug
extends layout.pug

block scripts
  script(src='/jquery.js')
  script(src='/pets.js')

block content
  h1= title
  - var pets = ['cat', 'dog']
  each petName in pets
    p= petName
```

To extend this layout, create a new file and **use the extends directive** with a path to the parent template. Then, define one or more blocks to override the parent block content.

Cookies

- To parse cookies with Express, we need the **cookie-parser** middleware.
- Install cookie-parser `npm install cookie-parser`
- Import cookie-parser module to your program, and cookie-parser as a middleware.

```
var cookieParser = require('cookie-parser')  
app.use(cookieParser())
```

- We can get all the cookies sent in the request via the req object
 - req.cookies returns an object with cookie names as keys.
 - if no cookie is sent, req.cookies returns {}

Cookies

- To set a cookie in the response message, we use the `cookie()` method of the `res` object.

```
res.cookie('name', 'value') // set cookie name to value
```

```
// set cookie name to value with other restrictions  
res.cookie('name', 'value', {domain: 'i.cs.hku.hk', path: '/user',  
                             expires: new Date(Date.now() + 3600000)}))
```

- `Date.now()` - returns the number of **milliseconds** elapsed since January 1, 1970 00:00:00 UTC
 - `new Date(Date.now() + 3600000)` – returns a `Date` object with time 1 hour in the future

Cookies

- To clear a cookie, use `clearCookie()` method
 - Must specify the name and the option which were specified in the `res.cookie()` (excluding `expires` and `maxAge`)

```
res.clearCookie('name', {domain: 'i.cs.hku.hk', path: '/user'})
```


Sessions

- To use server-side session, we need the express-session middleware.
- Install express-session `npm install express-session`
- The session middleware handles all things for us
 - Creating the session (and session cookie)
 - Creating the session object in `req` object
- We can add our `session variables` to the `req.session` object.
- The default storage is the MemoryStore
 - The memystore is deleted everytime the server stops; for production system, use a more stable storage (e.g., database).

Sessions

- To use session, the minimum requirement is to pass in a secret for signing the session ID cookie.

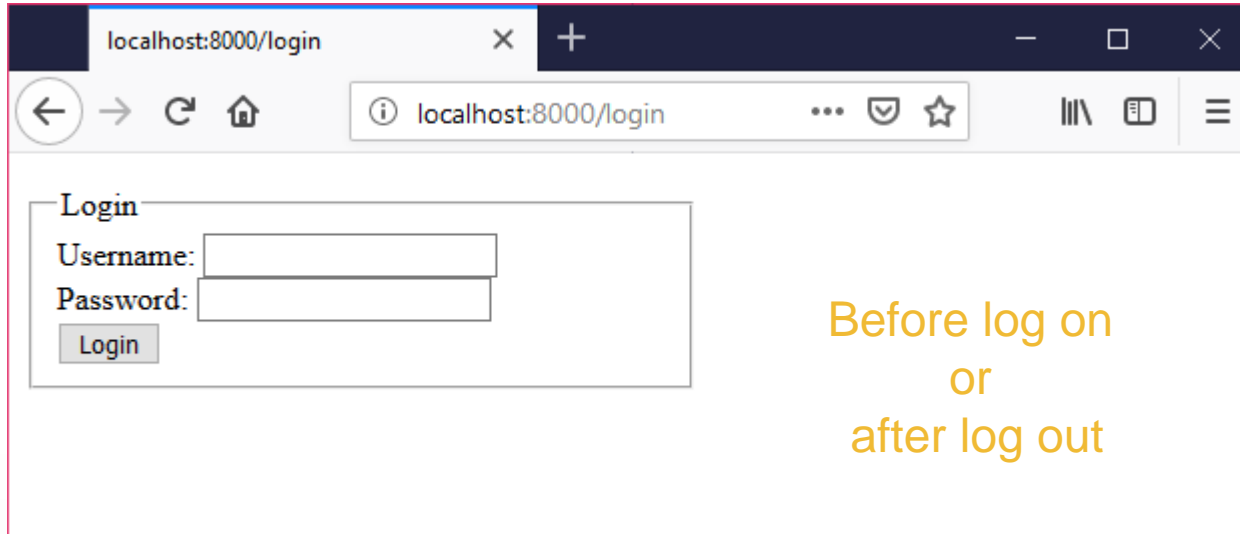
```
var session = require('express-session');  
app.use(session({secret: "something"}))
```

- Possible settings:

```
app.use(session({secret: "something", cookie: {maxAge: 600000, path: '/users'} })))
```

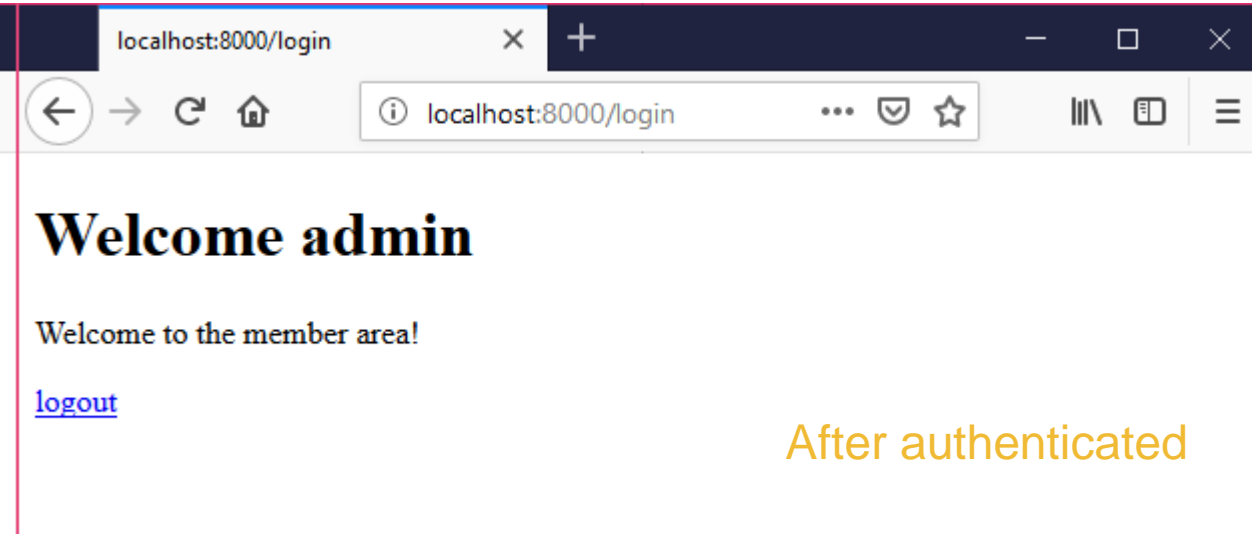
- To destroy the session, use **req.session.destroy**(callback)
 - The callback will be executed once the operation completed.
- To get the session ID, access the req property **req.sessionID** or req.session.id
- To get the properties of session cookie, use **req.session.cookie**

Demo – The login page again



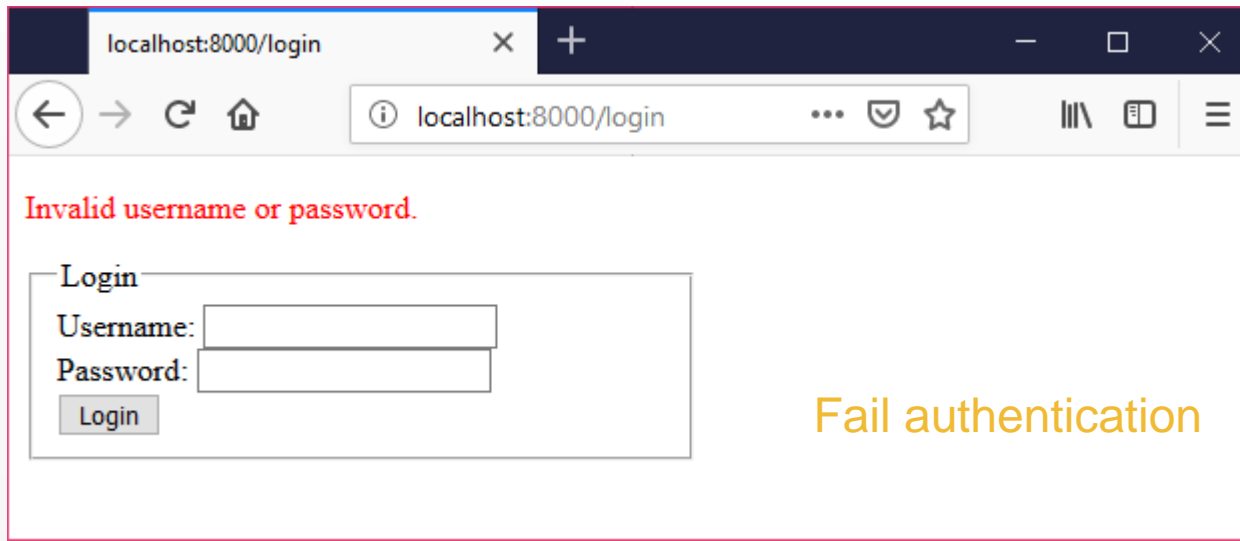
A browser window showing the login page at localhost:8000/login. The page contains a form with the title "Login", a "Username:" label, a text input field, a "Password:" label, another text input field, and a "Login" button. The browser's address bar shows "localhost:8000/login".

Before log on
or
after log out



A browser window showing the page after successful authentication. The page displays "Welcome admin" in a large, bold font, followed by "Welcome to the member area!" and a blue link labeled "logout". The browser's address bar shows "localhost:8000/login".

After authenticated



A browser window showing the login page after a failed authentication attempt. A red error message "Invalid username or password." is displayed at the top of the page. Below the message is the same login form as in the first screenshot. The browser's address bar shows "localhost:8000/login".

Fail authentication

```
session
├── index.js
├── node_modules
├── package-lock.json
├── package.json
├── views
│   ├── access.pug
│   └── login.pug
```

views/login.pug

```
<!DOCTYPE HTML>
<HTML>
<HEAD>
  style.
    #error {
      color: #ff0000;
    }
    fieldset {
      width: 50%;
    }
</HEAD>

<BODY>
  <form action="login" method="post">
    <p id="error">
      if msg != null
        span= msg
      else
        span= ""
    </p>
    <fieldset name="logininfo">
      <legend>Login</legend>
      <label for="username">Username:</label>
      <input type="text" name="username" id="username" /><br />
      <label for="password">Password:</label>
      <input type="password" name="password" id="password" /><br />
      <input type="submit" name="login" value="Login">
    </fieldset>
  </form>
</BODY>
</HTML>
```

views/access.pug

```
<!DOCTYPE HTML>
<HTML>
<BODY>
  h1 Welcome #{username}
  <p>Welcome to the member area!</p>
  <p>
    <a href="login?action=Logout">logout</a>
  </p>
</BODY>
</HTML>
```

index.js

```
//Set a predefined account
const SYSUSER = "admin";
const SYSPASSWORD = "secret"

const express = require('express')
const app = express();
//const bodyParser = require('body-parser');

app.set("view engine", "pug");
app.set("views", "views");

var session = require('express-session');

//Add session middleware to the pipeline
app.use(session({secret: "Hello World!"}));

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

//for the routes - GET /login and GET/login?action=Logout
app.get('/login', (req, res) => {
  console.log(req.session.id);
  res.render('login', {username: "admin", password: "secret"});
});
```

To get the urlencoded data carried by HTTP POST, use the built-in `express.urlencoded()` method (on Express v4.16.0 or above). Otherwise, use the `body-parser` middleware.

The extended option allows to choose between parsing the URL-encoded data with the standard `querystring` library (when false) or the `qs` library (when true)

index.js

//for the routes - GET /login and GET/login?action=Logout

```
app.get('/login', (req, res) => {  
  console.log(req.session.id);  
  if ((req.query.action) && (req.query.action == "Logout")) {  
    req.session.destroy((err) => {  
      if (err)  
        console.log("Cannot access session");  
    });  
    res.redirect('/login');  
  } else {  
    if (req.session.login)  
      res.render('access', {username: SYSUSER});  
    else  
      res.render('login');  
  }  
});
```

Destroy the
session and
redirect to
'/login'

If already
logged in

Render the login
page for the
GET

//for the route - POST /Login

```
app.post('/login', (req, res) => {  
  console.log(req.session.id);  
  if (req.session.login)  
    res.render('access', {username: SYSUSER});  
  if ((req.query.action) && (req.query.action == "Logout")) {  
    req.session.destroy((err) => {  
      if (err)  
        console.log("Cannot access session");  
    });  
    res.redirect('/login');  
  } else {  
    if (req.session.login)  
      res.render('access', {username: SYSUSER});  
    else  
      res.render('login');  
  }  
});
```

index.js

```
//for the route - POST /login
app.post('/login', (req, res) => {
  console.log(req.session.id);
  if (req.session.login)
    res.render('access', {username: SYSUSER});
  if ((req.body.username == SYSUSER) && (req.body.password == SYSPASSWORD)) {
    req.session.login = SYSUSER;
    res.render('access', {username: SYSUSER});
  } else
    res.render('login', {msg: 'Invalid username or password.'});
});

app.listen(8000, () => {
  console.log('Example app listening on port 8000!')
});
```

If already
logged in

If authenticated,
set session variable
and render the
access page

Render the login
page with error
msg

Readings

- MDN web docs
 - Express/Node introduction
 - https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction
 - Setting up a Node development environment
 - https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/development_environment

References

- Express
 - <https://expressjs.com/>
- Pug – Language reference
 - <https://pugjs.org/api/express.html>
- Rendering pages server-side with Express (and Pug)
 - <https://gist.github.com/joepie91/c0069ab0e0da40cc7b54b8c2203befe1>
- Express-session
 - <https://expressjs.com/en/resources/middleware/session.html>