Middleware

Middleware

- ***Middleware*** functions are functions that have access to the request object (req), the response object (res), and the next middleware function in the application's request-response cycle. The next middleware function is commonly denoted by a variable named next.
- Middleware functions can perform the following tasks:
  - Execute any code.
  - Make changes to the request and the response objects.
  - End the request-response cycle.
  - Call the next middleware in the stack.
- If the current middleware function does not end the request-response cycle, it must call next() to pass control to the next middleware function. Otherwise, the request will be left hanging.

Middleware

- The following shows the elements of a middleware function call(link):

```
var express = require('express');
var app = express();
app.get('/', function(req, res, next) {
        next();
});
app.listen(3000);
```

- Where get: HTTP method for which the middleware function applies.
- '/': Path (route) for which the middleware function applies.
- function (): The middleware function.
- req: HTTP request argument to the middleware function.
- res: HTTP response argument to the middleware function.
- next: Callback argument to the middleware function.

app.use()

- Since path defaults to “/”, middleware mounted without a path will be executed for every request to the app.

```
// this middleware will be executed for every request to the app
app.use(function (req, res, next) {
    console.log('Time: %d', Date.now());
next(); });
```

- Middleware functions are executed sequentially, therefore, the order of middleware inclusion is important:

```
// this middleware will not allow the request to go beyond it
app.use(function(req, res, next) {
    res.send('Hello World'); });
// requests will never reach this route
app.get('/', function (req, res) {
    res.send('Welcome');
});
```

app.use()

- **app.use([path,] function [, function...])**
- Mounts the specified middleware function or functions at the specified path. If path is not specified, it defaults to '/'.
- NOTE: A route will match any path that follows its path immediately with a “/”. For example, app.use('/apple', …) will match “/apple”, “/apple/images”, “/apple/images/news”, and so on.
- Eg.,

```
app.use('/fsd', function(req, res, next) {
console.log(req.originalUrl); // '/fsd/fsd3'
console.log(req.baseUrl); // '/fsd'
console.log(req.path); // '/fsd3'
next(); });
```

- Mounting a middleware function at a path will cause the middleware function to be executed whenever the base of the requested path matches the path.

Using Middleware

- Express is a routing and middleware web framework that has minimal functionality of its own: An Express application is essentially a series of middleware function calls.
- ***Middleware*** functions are functions that have access to the request object functions are functions that have access to the request object (req), the response object (res), and the next middleware function in the application's request-response cycle. The next middleware function is commonly denoted by a variable named next.
- Middleware functions can perform the following tasks:
  - Execute any code.
  - Make changes to the request and the response objects.
  - End the request-response cycle.
  - Call the next middleware function in the stack.
- If the current middleware function does not end the request-response cycle, it must call next() to pass control to the next middleware function. Otherwise, the request will be left hanging.
- An Express application can use the following types of middleware:
  - Application-level middleware
  - Router-level middleware
  - Error-handling middleware
  - Built-in middleware
  - Third-party middleware
- NOTE: You can load application-level and router-level middleware with an optional mount path. You can also load a series of middleware functions together, which creates a sub-stack of the middleware system at a mount point.

Router-level Middleware

- Router-level middleware works in the same way as application-level middleware, except it is bound to an instance of express.Router().

```
var router = express.Router();
```

- Load router-level middleware by using the router.use() and router.METHOD() functions.
- The following example code replicates the middleware system that is shown above for application-level middleware, by using router-level middleware

Error-handling Middleware

- Define error-handling middleware functions in the same way as other middleware functions, except with four arguments instead of three, specifically with the signature (err, req, res, next)):

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

- NOTE: Error-handling middleware always takes ***four*** arguments. You must provide four arguments to identify it as an error-handling middleware function. Even if you don't need to use the next object, you must specify it to maintain the signature. Otherwise, the next object will be interpreted as regular middleware and will fail to handle errors.
- For details about error-handling middleware, see: Error handling.

Built-in Middleware

- **express.static**: This function is based on serve-static, and is responsible for serving static assets such as HTML files, images, and so on.
- The function signature is:

  express.static(root, [options])
- The root argument specifies the root directory from which to serve static assets.
- For information on the options argument and more details on this middleware function, see express.static.

Serving Static files in Express

- To serve static files such as images, CSS files, and JavaScript files, use the express.static built-in middleware function in Express.
- Pass the name of the directory that contains the static assets to the express.static middleware function to start serving the files directly. For example, use the following code to serve images, CSS files, and JavaScript files in a directory named public:

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

- Now, you can load the files that are in the public directory:

  http://localhost:3000/images/kitten.jpg

  http://localhost:3000/css/style.css

  http://localhost:3000/js/app.js

  http://localhost:3000/images/bg.png

  http://localhost:3000/hello.html

- To use multiple static assets directories, call the express.static middleware function multiple times:

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

Serving Static files in Express 2

- Now, you can load the files that are in the public directory from the /static path prefix.

  http://localhost:3000/static/images/kitten.jpg

  http://localhost:3000/static/css/style.css

  http://localhost:3000/static/js/app.js

  http://localhost:3000/static/images/bg.png

  http://localhost:3000/static/hello.html

- However, the path that you provide to the express.static function is relative to the directory from where you launch your node process. If you run the express app from another directory, it's safer to use the absolute path of the directory that you want to serve:

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

Third-party Middleware

- Use third-party middleware to add functionality to Express apps.
- Install the Node.js module for the required functionality, then load it in your app at the application level or at the router level.
- The following example illustrates installing and loading the cookie-parsing middleware function cookie-parser.

```
$ npm install cookie-parser
```

```
var express = require('express');
var app = express();
var cookieParser = require('cookie-parser');
// load the cookie-parsing middleware
app.use(cookieParser());
```