
THE UNIVERSITY
of ADELAIDE
Web and Database Computing
adelaide.edu.au
Server Architecture and Routes: Express Server Architecture

HTTP Server and Express

Express server files

Server

- `bin` contains program files used to run our server
  - `www` Node.js script that sets up and starts the basic server.
- `node_modules` contains modules installed via npm
- `routes` contains Node.js code files for serving dynamic content
  - `index.js` & `users.js` Node.js scripts allow modularising of code.
- `app.js` is the main Node.js application file
- `package-lock.json` locks the versions of modules used
- `package.json` contains a metadata about our app

Client

- `public` contains files we can serve statically including any client-side code.

bin/www

Node.js script that sets up and starts the basic server.

- Utilises the Node.js HTTP library to start a basic server.
- Binds the server to a port on the host system and starts listening for requests
- Provides the `req` and `res` objects for processing
- Passes those objects to Express.

Demo

Walking through the bin/www file.

Express as a framework

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.

- A request is passed to Express
- Express passes the request from one function to the next, until the request reaches an endpoint.
  - Requests can be filtered by path and method.

app.js

Our app.js file is the main file for our web application.

- Contains libraries and middleware that will be used across the whole application.

Demo

Walking through the app.js file.

routes/index.js & users.js

Files in the routes folder allow us to split our routes into smaller modules to make them easier to manage.

Demo

Walking through files in the routes folder.

Routes and Middleware

What is Middleware?

From expressjs.com:

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 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.

Why Middleware?

- Allows efficient rule-based code reuse.
- Separate our code into distinct functions.
- Easily include external libraries.

Basic Middleware

```
// In app.js
app.use(function(req, res, next) {
    // Do stuff
    next();
});

// In routes/*.js
router.use(function(req, res, next) {
    // Do stuff
    next();
});
```

The `.use` method applies this middleware to all requests. `next` is a callback used to call the next piece of middleware in the chain.

- Calling next allows the request to continue being processed.
- Not calling `next` will cause the request to stop being processed, and the request may hang/timeout if `res.end`/`res.send` not used (no response sent).

Basic Middleware ... with a path

We can apply paths to middleware the same way we do to GET/POST requests.

```
// In app.js
app.use('/some/path', function(req, res, next) {
    // Do stuff
    next();
});

// In routes/*.js
router.use('/some/path', function(req, res, next) {
    // Do stuff
    next();
});
```

Routes are middleware?!

In fact, you specify the method to middleware the same way we do to GET/POST requests. Our routes are just middleware without the `next();`.

```
// In app.js
app.get('/some/path', function(req, res, next) {
    // Do stuff
    next();
});

// In routes/*.js
router.post('/some/path', function(req, res, next) {
    // Do stuff
    next();
});
```

Routes are middleware?!

In fact, you specify the method to middleware the same way we do to GET/POST requests. Our routes are just middleware without the `next();`.

```
// In app.js
app.get('/some/path', function(req, res, next) {
    // Do stuff
    next();
});

// In routes/*.js
router.post('/some/path', function(req, res, next) {
    // Do stuff
    next();
});
```

Summary

- Express uses a middleware architecture.
- Middleware allows requests to be handled by multiple independant pieces of code.
- Improves code reuse.
- You can easily write your own middleware using `next()` function;
- Routes are middleware that terminate.

THE UNIVERSITY
of ADELAIDE