Node.js

Day 4

Alexandre Perrin

July 13, 2017

nomades.ch

Today

- More about Express.js
    - Middleware
    - Routing
    - Middleware modules list
- Authentication and authorization
    - Verifying passwords
    - Passportjs
- Let’s code!

More about Express.js

Application Middleware

We need to have code executed for each requests, like logging for example. How can we achieve that with Express.js?

Using Express Middleware:

```javascript
"use strict";

const express = require("express");
const app = express();

/* functions given app.use() are Express Middleware */
app.use(function (req, res, next) {
    /* this codepath will be executed *first* at every request */
    console.log(req.url + " requested at " + new Date());
    return next(); /* call the next Middleware in the stack */
});

/* functions given to app.get(), app.post() etc. are Middleware too called
   Route handlers. Route handlers are only executed when the method and path
   match the request. */
app.get('/hello', function (req, res, next) {
    res.send('Hello World!') /* shortcut the Middleware stack by ending the
                                request-response cycle */
});

/* functions given app.use() are Express Middleware */
app.use(function (req, res, next) {
    console.log("never used");
```

Error Handlers

```javascript
"use strict";

const express = require("express");
const app = express();

app.get('/ok', function (req, res, next) {
    res.send("ok");
});
app.get('/oups', function (req, res, next) {
    /* this will pass to the error handler Middleware */
    next(new Error("Got 99 problems but error handling ain't one"));
});

/* this Middleware is an error handler because it takes four arguments,
   the first one being the error. */
app.use(function (err, req, res, next) {
    console.error(err.stack);
    res.status(500).send('Something broke!'); /* shortcut the Middleware stack
                                                 by ending the request-response
                                                 cycle */
});

app.listen(3000, () => console.log('listening on port 3000!'));
```

Notice how error-handling Middleware must be ”used” *last*.

Route handlers

Route Handlers can be stacked:

```
"use strict";

const express = require("express");
const app = express();

app.get('/hello', function (req, res, next) {
    res.locals.message = "Hello";
    return next(); /* call the next Middleware in the stack */
}, function (req, res, next) {
    res.locals.message += " World";
    return next(); /* call the next Middleware in the stack */
}, function (req, res, next) {
    return res.send(res.locals.message); /* end the request-response cycle */
});

app.listen(3000, () => console.log('listening on port 3000!'));
```

In practice, this is useful with generated middleware, e.g.

```
app.get('/admin', authorize('admin'), (req, res, next) => {
    res.send("Welcome to the admin page!");
});
```

Route path

You may use Regular Expression in Route paths:

```
"use strict";

const express = require("express");
const app = express();

app.get(/.*fly$/, function (req, res) {
    res.send(req.url + " is in the air"); /* end the request-response cycle */
});

app.get('*', function (req, res) {
    res.send("nope."); /* end the request-response cycle */
});

app.listen(3000, () => console.log('listening on port 3000!'));
```

Parameters

As you probably know by now, Route paths may have parameters:

```
"use strict";

const express = require("express");
const app = express();

app.get('/flights/:from/:to', function (req, res) {
    return res.send(req.params); /* end the request-response cycle */
});

app.listen(3000, () => console.log('listening on port 3000!'));
```

The Express Route Tester is handy to debug and understand how Express.js match parameters.

Parameters

Using `app.param()` or `router.param()` we can execute code when a named parameter is given:

```
app.param('post_id', function (req, res, next, post_id) {
    Post.find(post_id, (err, post) => {
        if (err) {
            /* this will pass to the error handler Middleware */
            return next(err);
        } else if (!post) {
            /* end the request-response cycle */
            return res.status(404 /* Not Found */).send();
        }
        req.locals.post = post;
        return next(); /* call the next Middleware in the stack */
    });
});

app.get('/api/posts/:post_id', function (req, res, next) {
    return res.send(req.locals.post_id);
});
```

Middleware modules list

There are some very useful ”generic” and configurable Express.js Middleware modules listed at expressjs.com.

If you don’t find what you need, then you’ll have to write your own Middleware.

Authentication and authorization

Definition

- *Authentication* is the process of verifying who you are.
- *Authorization* is the process of verifying that you are allowed to do what you request.

Passwords verification

Traditionally, authentication is performed by providing a user id (e.g. an email) and a secret password. To authenticate a user, you only need to be able to *verify* its password.

The most naive and simple way to verify a password is to store it ”as-is” in the database and do a comparison with the one provided for authentication.

Obviously this is a terrible idea. So how should password verification be implemented?

use bcrypt unless you really know what you are doing.

bcrypt for Node.js

Documentation at the GitHub project page.

```
% npm install bcrypt
```

Storing password hashs:

```
"use strict";

const bcrypt = require("bcrypt");
const cost = 10; // the bcrypt cost, 4 is the minimum.
const password = "Open Sesame";

// hash the password at user creation time (or password reset etc.).
bcrypt.hash(password, cost).then(hash => {
    // save the hash in a database.
    console.log(hash);
});
```

bcrypt for Node.js

Verifying passwords:

```
"use strict";

const bcrypt = require("bcrypt");
const hash = "$2a$10$cuD53jYHpp9bOFsDAT4n0uop42Ib1/FCTSVK1hoN8ZroNlTBo4FDe";

[
    "Open Mustard",
    "Open Sesame",
].forEach(password => {
    // verify the password
    bcrypt.compare(password, hash).then(success => {
        if (success)
            console.log(`${password} authorized`);
        else
            console.log(`${password} unauthorized`);
    });
});
```

Passportjs

Passportjs is an *authentication* Middleware for Node.js. It plays very well with Express.js and support many ”Strategies” like username and password, Google, Twitter, GitHub, Facebook etc. etc.

```
% npm install passport passport-http
```

ACL

ACL is the most popular Access Control List module for Node.js. It is an *authorization* framework that can be used as Middleware with Express.js.

Let’s code!

Yet Another Blog Engine

1. Uses some Express.js Middleware modules, like errorhandler, and morgan.
2. Add authentication to your application using Passportjs with a Strategy.

Questions?

Read on later

OAuth Has Ruined Everything by Burke Holland.