NODE.JS GLOBAL

Middleware. Frameworks

FEBRUARY, 2019

AGENDA

1. What is Express
2. What is middleware
3. Build in modules
4. App structure

WHAT IS EXPRESS

Express

Fast, unopinionated, minimalist web framework for Node.js

```
$ npm install express
```

REQUEST RESPONSE CYCLE

1. Request
2. Configuration
3. Application middlewares
4. Route middlewares
5. View (html / json)
6. Response

MIDDLEWARE IS THE BACKBONE

Express = Routing + Middlewares

MIDDLEWARE CONCEPT

1. Execute any code.
2. Make changes to the request and the response objects.
3. End the request-response cycle.
4. Call the next middleware in the stack.

MIDDLEWARE TYPES

1. Application-level
2. Router-level
3. Error-handling
4. Built-in
5. 3rd party (ex.: xxx-parser)

APPLICATION LEVEL MIDDLEWARE

1. Change anything to the request and the response objects and then call next()

```
app.use(function (req, res, next) {
    console.log('Time: %d', Date.now());
    next();
});
```

2. You can attach middleware to the specific route

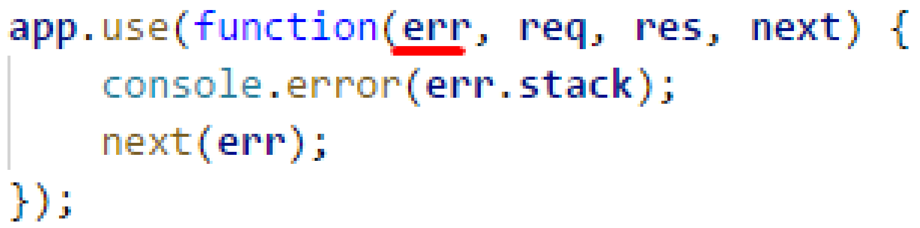
```
app.use('/abcd', function (req, res, next) {
    console.log('Time: %d', Date.now());
    next();
});
```

3. Do not forget about next(). This code will never return control to the route handler

```
app.use(function(req, res, next) {
    console.log('Time: %d', Date.now());
    res.send('Hello World');
});
```

ERROR-HANDLING MIDDLEWARE

1. Has four arguments – and the first is error



```
app.use(function(err, req, res, next) {
    console.error(err.stack);
    next(err);
});
```

2. You can call it by **next(err)** in any middleware

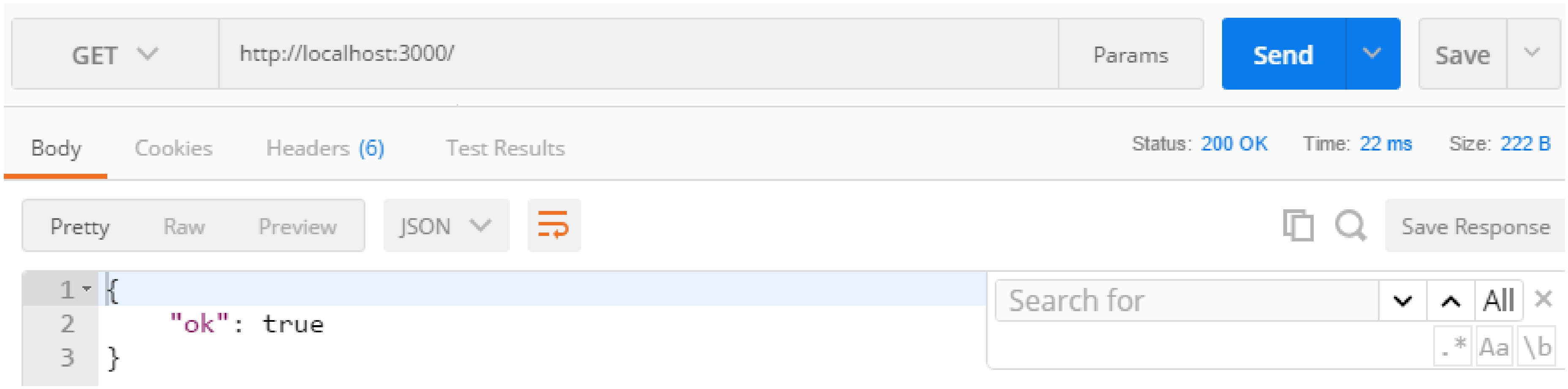
3. Once **next(err)** is called, all other non-error middlewares would be skipped

BACK TO EXPRESS

1. express()
2. Application
3. Request
4. Response
5. Router

SIMPLE EXPRESS APP

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

app.listen(3000);

app.get('/', function (req, res) {
    res.json( { ok : true } );
});

```



GET http://localhost:3000/ Params Send Save

Body Cookies Headers (6) Test Results Status: 200 OK Time: 22 ms Size: 222 B

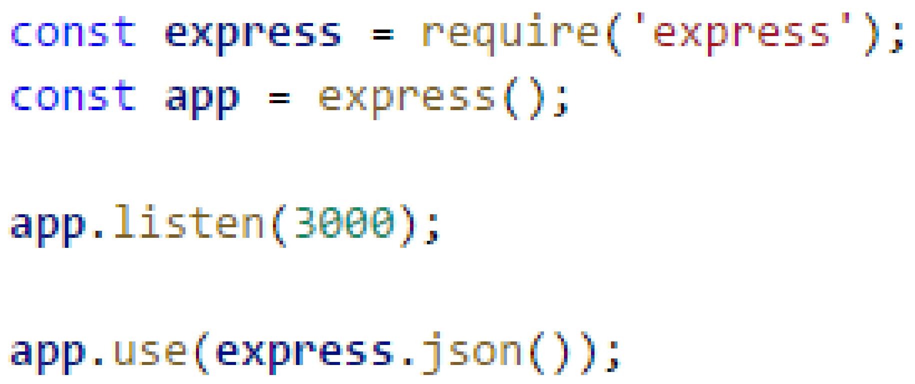
Pretty Raw Preview JSON Save Response

```
{
    "ok": true
}
```

express()

1. Creates an Express application

2. Provide some useful middlewares
   - **static** to server static files
   - **json** – to parse JSON to body
   - **urlencode** – to parse urlencoded body



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

app.listen(3000);

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

APPLICATION

1. Routing HTTP requests (all, METHOD)
2. General app configuration (enable / disable)
3. Adding middleware (use)
4. Global parameters handling
5. Registering and using a template engine(engine, render)

APPLICATION: ROUTES

1. all
2. METHOD
3. use (for middleware)

APPLICATION: PROPERTIES

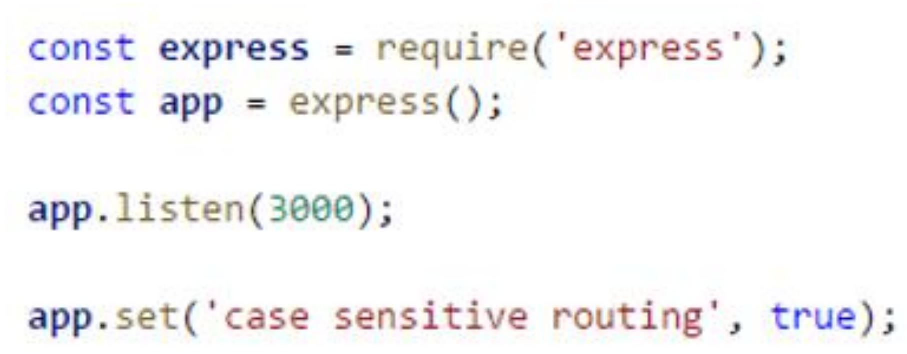
1. listen
2. locals
3. set & get
4. disable & enable

```
app.enabled('isAdmin');
// => false

app.enable('isAdmin');
app.enabled('isAdmin');
// => true
```

```
app.set('title', 'Express API Reference');
app.get('title'); // "Express API Reference"
```

APPLICATION: SETTINGS

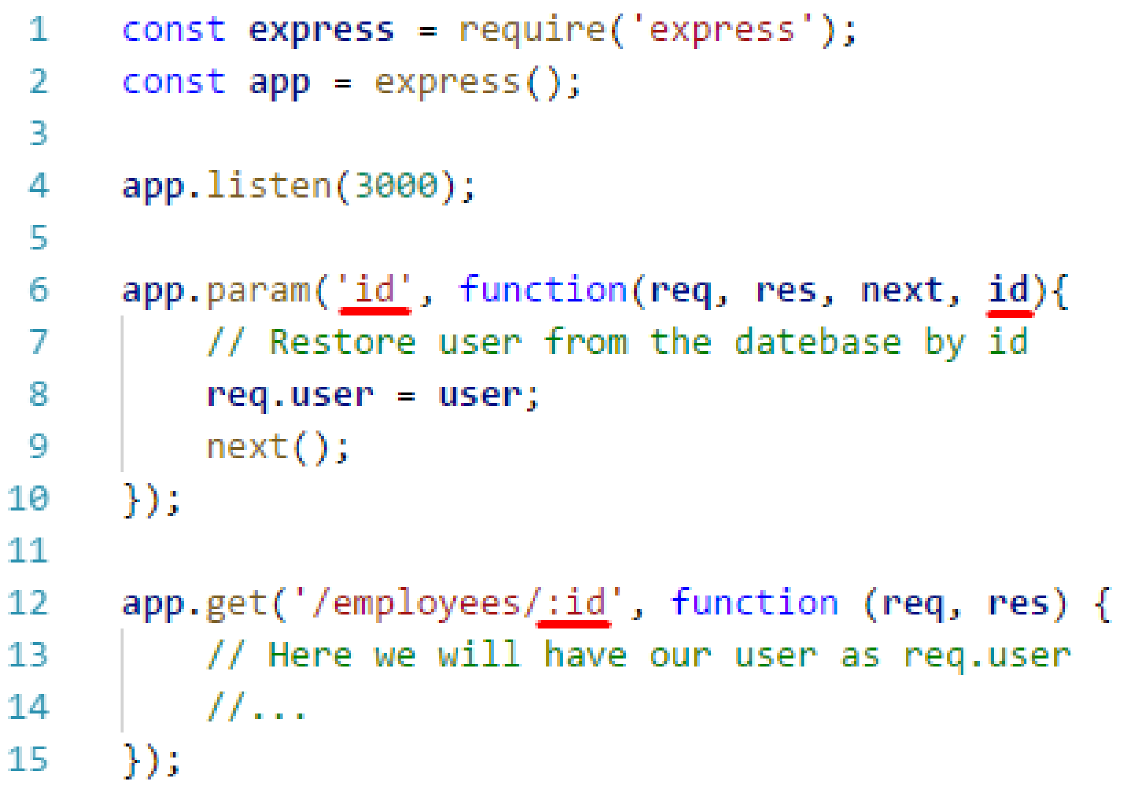
1. case sensitive routing
2. strict routing
3. x-powered-by
4. trust proxy



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

app.listen(3000);

app.set('case sensitive routing', true);
```

APPLICATION: PARAM

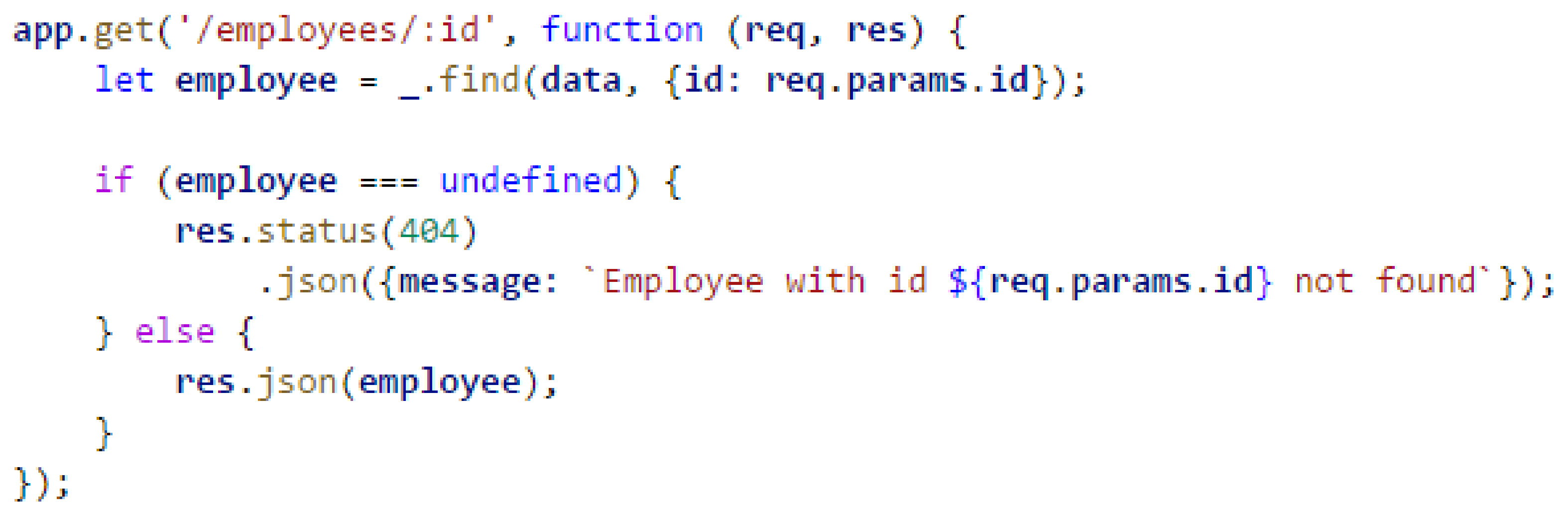
1. Adds callback triggers to route specific route-parameters
2. .. also can be used to load user settings



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

app.listen(3000);

app.param('id', function(req, res, next, id){
    // Restore user from the datebase by id
    req.user = user;
    next();
});

app.get('/employees/:id', function (req, res) {
    // Here we will have our user as req.user
    //...
});
```

REQUEST

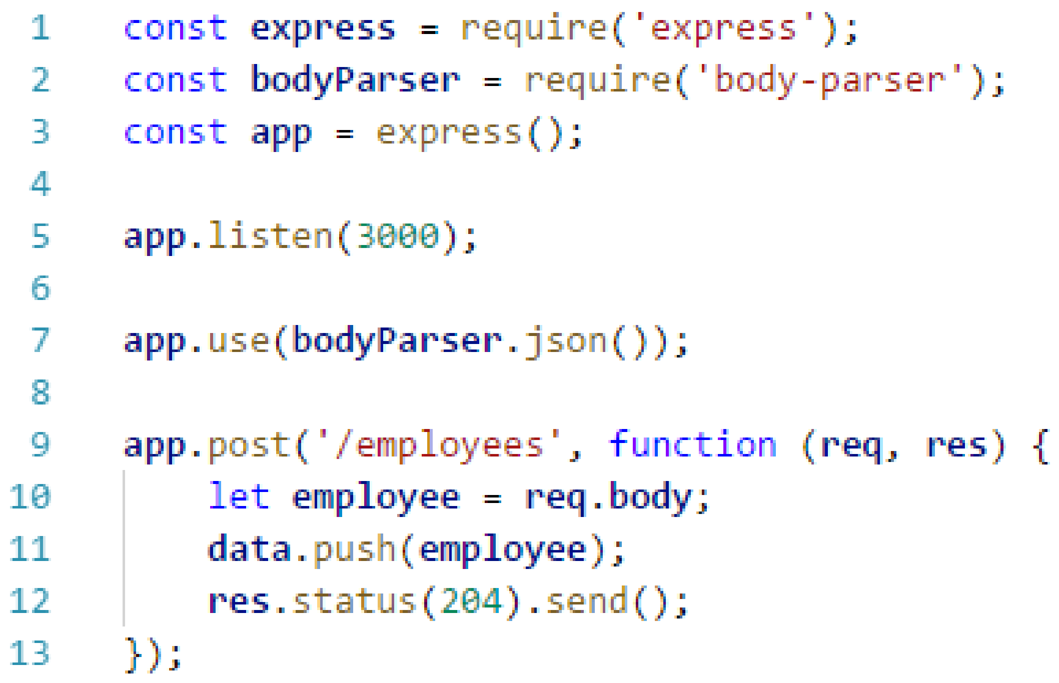
1. Extended version of **IncomingMessage** from http module
2. body
3. cookies
4. params (path params)
5. query (query params)



```
app.get('/employees/:id', function (req, res) {
    let employee = _.find(data, {id: req.params.id});

    if (employee === undefined) {
        res.status(404)
            .json({message: `Employee with id ${req.params.id} not found`});
    } else {
        res.json(employee);
    }
});
```

REQUEST: BODY

1. Contains submitted data as key-value pairs and undefined by default
2. Use body-parser to populate body from json



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

app.listen(3000);

app.use(bodyParser.json());

app.post('/employees', function (req, res) {
    let employee = req.body;
    data.push(employee);
    res.status(204).send();
});
```

REQUEST: BODY (4.16.0+)

1. Contains submitted data as key-value pairs and undefined by default
2. ~~Use body-parser to populate body from json~~
3. Use **express.json** middleware to populate body from json

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

app.listen(3000);

app.use(express.json());

app.post('/employees', function (req, res) {
    let employee = req.body;
    data.push(employee);
    res.status(204).send();
});
```

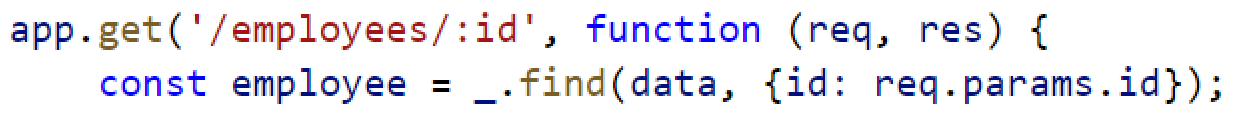
REQUEST: COOKIES

1. Contains cookies as key-value pairs
2. Use cookie-parser to populate

```javascript
const express = require('express');
const cookieParser = require('cookie-parser')
const app = express();

app.listen(3000);

app.use(cookieParser());

app.get('/', function(req, res) {
  console.log('Cookies: ', req.cookies)
})
```

REQUEST: PARAMS

1. Object with properties mapped from named route “parameters”
2. .. or array of elements captured by regular expressions



```
app.get('/employees/:id', function (req, res) {
    const employee = _.find(data, {id: req.params.id});
```

Route path: '/employees/:id'
Request url: http://localhost:3000/employees/42
req.params: {"id": "42"}

Route path: '/employees/*'
Request url: http://localhost:3000/employees/42
req.params[0]: "42”

Route path: '/employees/*'
Request url: http://localhost:3000/employees/42/awards/17
req.params[0]: "42/awards/17"

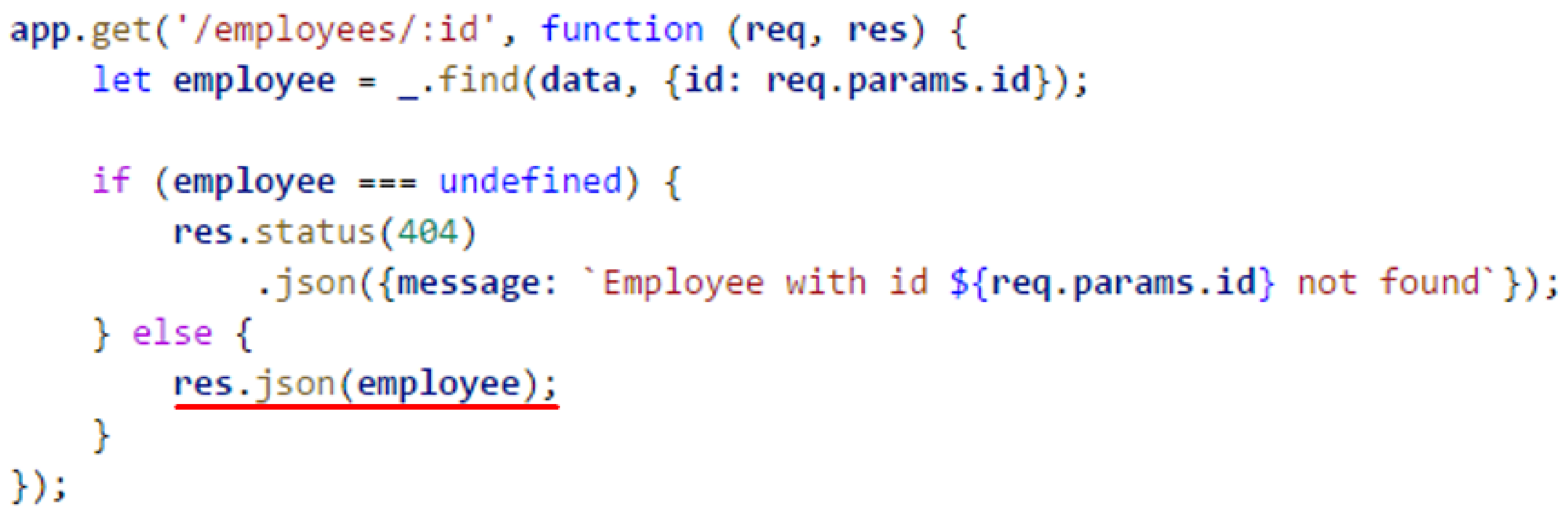
REQUEST: QUERY

1. Object with properties mapped from query string parameters
2. If there is no query string, it is the empty object, {}.
3. You have to merge req.params and req.query in order to get all incoming parameters – from recourse path and query string

Request url: http://localhost:3000/employees?order=desc
req.query: {"order": ”desc"}

RESPONSE

1. Represents the HTTP response that an Express app sends for HTTP request
2. locals
3. Sending response



```
app.get('/employees/:id', function (req, res) {
    let employee = _.find(data, {id: req.params.id});

    if (employee === undefined) {
        res.status(404)
            .json({message: `Employee with id ${req.params.id} not found`});
    } else {
        res.json(employee);
    }
});
```

RESPONSE : LOCALS

1. Contains response local variables scoped to the request
2. Available only to current request / response cycle (unlike app.locals)
3. .. useful for storing authenticated user, user settings, etc.

```
app.use(function(req, res, next){
  res.locals.user = req.user;
  res.locals.authenticated = ! req.user.anonymous;
  next();
});
```

SENDING RESPONSE

1. **end** – quickly end the response without any data
2. **sendStatus** – ответ только со статусом и его описанием
3. **send** – default response with data (Buffer, String, object, or Array)
4. **sendFile** – sends the file at the given path to the client
5. **json** –JSON response with proper content-type
6. **jsonp** – JSON response with JSONP support. Callback called *callback* by default
7. **redirect** – redirects to the specified URL (or path). You can set status code
8. **render** – renders a view and sends the rendered HTML string to the client

ROUTER

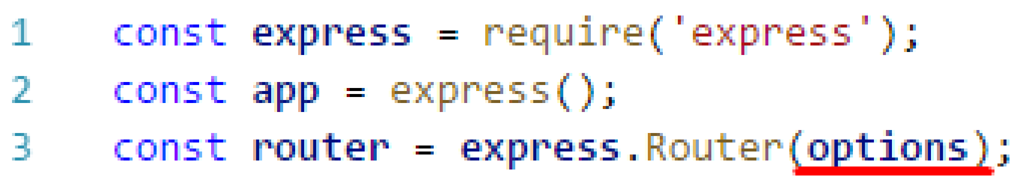
1. The object with isolated instance of middleware and routes
2. METHOD
3. all
4. param
5. route
6. use

ROUTER: EXAMPLE

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

app.listen(3000);

router.get('/employees/:id', function(req, res) {
  res.json( {id : req.params.id });
});

app.use('/', router);
```

ROUTER: OPTIONS

1. **caseSensitive**: when disabled treat /Users as /users
2. **strict** – when disabled treat /Users as /Users/
3. **mergeParams** - Preserve the req.params values from the parent router
4. All options are disabled by default



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

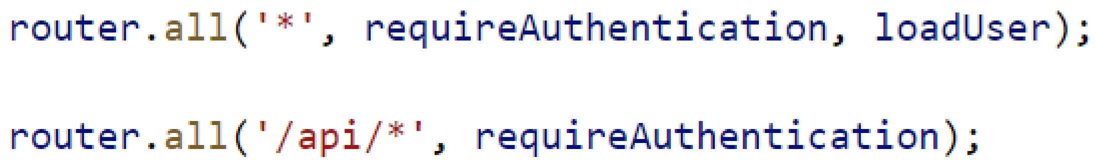
ROUTER: METHOD

Provide the routing functionality for specific HTTP methods (verbs)

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

app.listen(3000);

router.get('/', function(req, res) {
  res.json( { ok : true });
});

app.use('/', router);
```

ROUTER: ALL

1. Provide the routing functionality for all HTTP methods
2. .. useful for checking user authentication and load user settings



```
router.all('*', requireAuthentication, loadUser);

router.all('/api/*', requireAuthentication);
```

ROUTER: PARAM

1. Adds callback triggers to route specific route-parameters
2. .. also can be used to load user settings

```
router.param('id', function (req, res, next, id) {
  console.log('CALLED ONLY ONCE');
  next();
});

router.get('/employees/:id', function (req, res, next) {
  console.log('although this matches');
  next();
});

router.get('/employees/:id', function (req, res) {
  console.log('and this matches too');
  res.end();
});
```

ROUTER: ROUTE

1. Returns an instance of a single route
2. Useful for avoiding duplicate route naming and thus typing errors

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

router.param('id', function(req, res, next, id) {
  req.employee = _.find(data, {id: id});
  next();
});

router.route('/employees/:id')
.all(function(req, res, next) {
  // runs for all HTTP verbs first
  // think of it as route specific middleware!
  next();
})
.get(function(req, res, next) {
  res.json(req.employee);
})
```

ROUTER: ROUTE NEXT

1. You can provide multiple callback functions that behave just like middleware
2. Call next('route') to bypass the remaining route callbacks

```
router.route('/emploees/:id')
    .get(function(req, res, next) {
        // if the user ID is 0, skip to the next route
        if (req.params.id == 0) next('route');
        // otherwise pass the control to the next middleware function
        else next();
    }, function(req, res, next) {
        // some special logic for requests with param id != 0
        next();
    })
    .get(function(req, res, next){
        res.json();
    });
```

ROUTER: USE

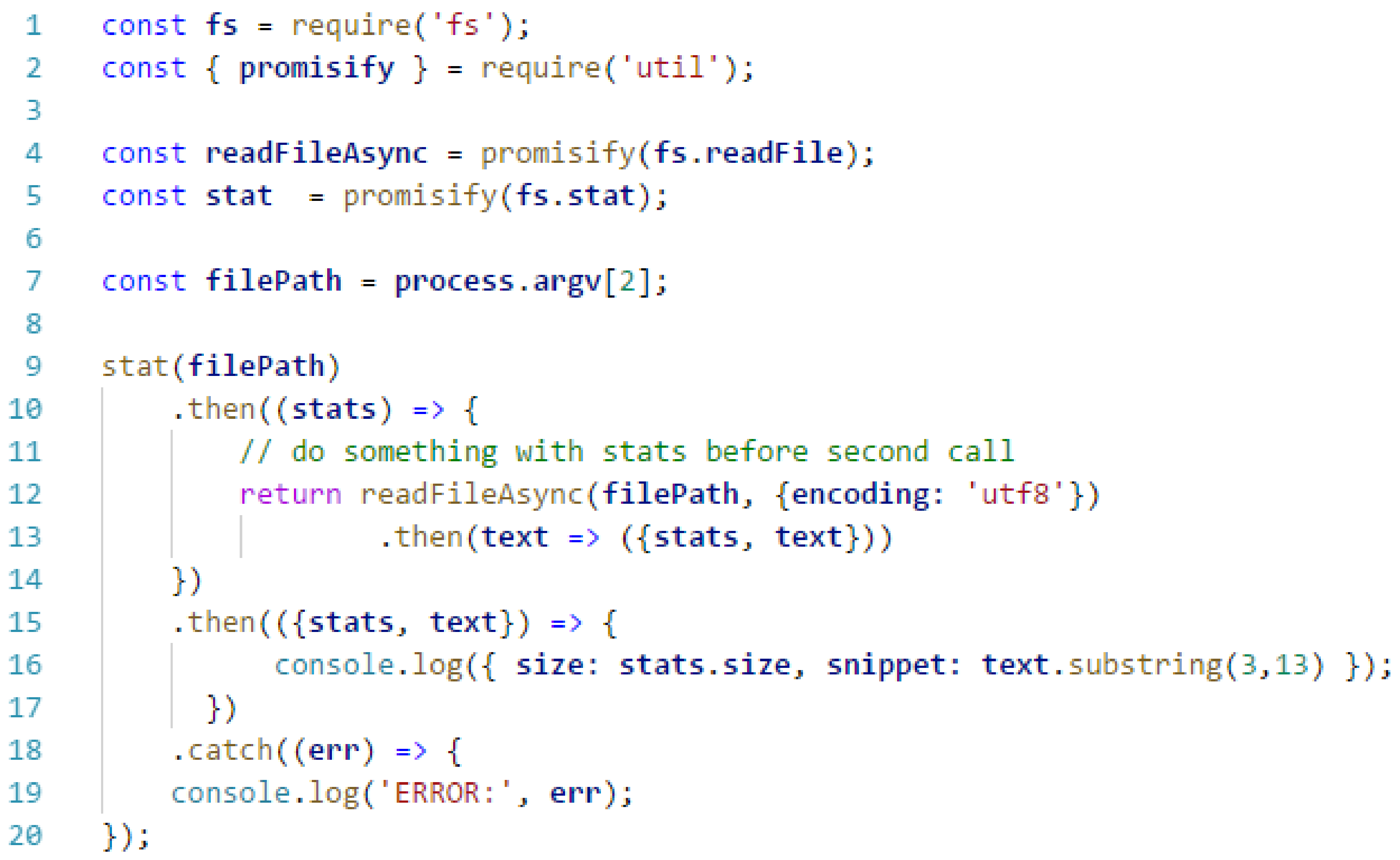
1. Adds the specified middleware function
2. .. and optional mount it to the specific path

```
router.use(function(req, res, next) {
  console.log('%s %s %s', req.method, req.url, req.path);
  next();
});

app.use('/employees', router);
```

MIDDLEWARE CONCEPT. ONE MORE TIME

1. Execute any code.
2. Make changes to the request and the response objects.
3. End the request-response cycle.
4. Call the next middleware in the stack.

OLD-FASHIONED CALLBACKS



```javascript
const fs = require('fs');
const { promisify } = require('util');

const readFileAsync = promisify(fs.readFile);
const stat  = promisify(fs.stat);

const filePath = process.argv[2];

stat(filePath)
    .then((stats) => {
        // do something with stats before second call
        return readFileAsync(filePath, {encoding: 'utf8'})
                .then(text => ({stats, text}))
    })
    .then(({stats, text}) => {
          console.log({ size: stats.size, snippet: text.substring(3,13) });
      })
    .catch((err) => {
    console.log('ERROR:', err);
});
```

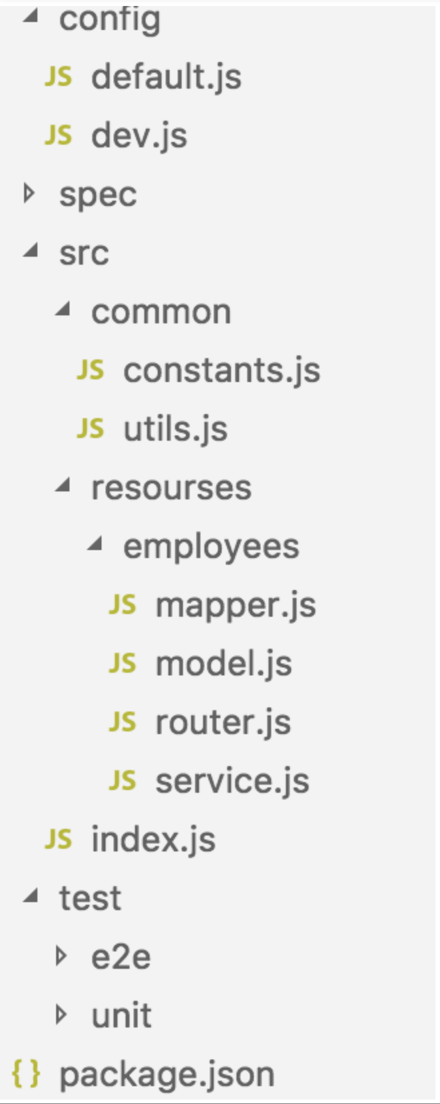
AND NOW.. ASYNC!

```
const fs = require('fs');
const { promisify } = require('util');

const readFileAsync = promisify(fs.readFile);
const stat  = promisify(fs.stat);

const filePath = process.argv[2];

(async () => {
    try {
        let stats   = await stat(filePath);
        let text    = await readFileAsync(filePath, {encoding: 'utf8'});

        console.log({ size: stats.size, snippet: text.substring(3,13) });
    } catch (err) {
        console.log('ERROR:', err);
    }
})();
```

MODERN CONTROL-FLOW WITH CO

```
const fs = require('fs');
const { promisify } = require('util');

const readFileAsync = promisify(fs.readFile);
const stat  = promisify(fs.stat);

const filePath = process.argv[2];

const co = require('co');

co(function *(){
    let stats   = yield stat(filePath);
    let text    = yield readFileAsync(filePath, {encoding: 'utf8'});

    console.log({ size: stats.size, snippet: text.substring(3,13) });
})
.catch((err) => {
    console.log('ERROR:', err);
});
```

APPLICATION STRUCTURE (ONE OF)



```
config
  default.js
  dev.js
spec
src
  common
    constants.js
    utils.js
  resourses
    employees
      mapper.js
      model.js
      router.js
      service.js
  index.js
test
  e2e
  unit
package.json
```

1. In **config** - files by environment with default one
2. **spec** with your API documentation (swagger)
3. In **src** – **common** stuff and **resources**
4. Every resource has it’s own **router** with **service**, possibly **mapper** and/or **model**

EXTRA MODULES

1. Config handler – get config for the current environment and validate it
   - config, nconf, dotenv
2. Error handler – throw error with extra details, handlers for popular errors
   - boom, http-errors
3. Log handler – log errors and debug data for different environments
   - winston, banyan
4. Validator – validate request payloads and params
   - ajv, joi
5. Helpers – add HAL links, check permissions, test request/response against spec
   - halson, ..

USEFUL LINKS

- ExpressJS API documentation
- Writing middleware for use in Express apps
- Express middleware
- Error handling
- API Design in Node.js Using Express and Mongo (lynda.com)
- Getting Started with Express.js (egghead.io)
- Designing a Beautiful REST+JSON API

NODE.JS GLOBAL

MIDDLEWARE. FRAMEWORKS
BY
VLADISLAV LOMAKO
MOHAMADALI GANJI