Node.js: create a REST API without ExpressJS

In this article we will see how to implement a simple REST API without using ExpressJS to demonstrate how impractical this approach is.

Let's define the data model in Mongoose:

```
'use strict';

const mongoose  = require('mongoose');

const { Schema }  = mongoose;

const PostSchema = new Schema({
    title: String,
    description: String

},{collection: 'posts'});

module.exports = mongoose.model('posts', PostSchema);
```

As we do not have access to the `request.body` object, we need to implement from scratch a helper function for the HTTP `POST` and `PUT` methods.

```
'use strict';

const body = req => {
    return new Promise((resolve, reject) => {
        try {
            let data = [];

            req.on('data', chunk => {
                data.push(chunk);
            });

            req.on('end', () => {
                
resolve(JSON.parse(Buffer.concat(data).toString()));
            });

            req.on('error', err => {
                throw err;
            });
        } catch (err) {
            reject(err);
        }
    })
};

module.exports = {
    body
};
```

In practice, the data flow of the request is assembled to obtain a string that will be converted into a JSON object.

At this point we can create the controller that will manage the routes.

```
'use strict';

const Post = require('../models/post');
const { body } = require('../lib');
```

```
class PostController {
    async index(req, res) {
        try {
            const posts = await Post.find();
            res.writeHead(200, { 'Content-Type':
'application/json' });
            res.end(JSON.stringify(posts));
        } catch(err) {
            res.writeHead(500, { 'Content-Type':
'application/json' });
            res.end(JSON.stringify({ error: err }));
        }
    }

    async single(req, res, id) {
        try {

            const post = await Post.findById(id);
            if(!post) {
                res.writeHead(404, { 'Content-Type':
'application/json' });
                res.end(JSON.stringify({ error: 'Not
Found' }));
            } else {
                res.writeHead(200, { 'Content-Type':
'application/json' });
                res.end(JSON.stringify(post));
            }

        } catch(err) {
            res.writeHead(500, { 'Content-Type':
'application/json' });
            res.end(JSON.stringify({ error: err }));
        }
    }

    async create(req, res) {
        try {
            const data = await body(req);
            const newPost = await new
Post(data).save();

            res.writeHead(201, { 'Content-Type':
'application/json' });
            res.end(JSON.stringify(newPost));
        } catch(err) {
            res.writeHead(500, { 'Content-Type':
'application/json' });
            res.end(JSON.stringify({ error: err }));
        }
    }

    async update(req, res, id) {
        try {
            const post = await Post.findById(id);

            if(!post) {
                res.writeHead(404, { 'Content-Type':
'application/json' });
                res.end(JSON.stringify({ error: 'Not
Found' }));
            } else {
                const data = await body(req);
                const updatedPost = await
Post.findByIdAndUpdate(id, data);

                res.writeHead(200, { 'Content-Type':
'application/json' });
```

```
                res.end(JSON.stringify(updatedPost));
            }
        } catch(err) {
            res.writeHead(500, { 'Content-Type':
'application/json' });
            res.end(JSON.stringify({ error: err }));
        }
    }

    async remove(req, res, id) {
        try {
            const post = await Post.findById(id);

            if(!post) {
                res.writeHead(404, { 'Content-Type':
'application/json' });
                res.end(JSON.stringify({ error: 'Not
Found.' }));
            } else {
                const removedPost = await
Post.findByIdAndRemove(id);

                res.writeHead(200, { 'Content-Type':
'application/json' });
                res.end(JSON.stringify(removedPost));
            }
        } catch(err) {
            res.writeHead(500, { 'Content-Type':
'application/json' });
            res.end(JSON.stringify({ error: err }));
        }
    }
};
```

```
module.exports = PostController;
```

Since we don't have ExpressJS's `response.json()` middleware method, we have to manually serve JSON by setting the correct HTTP header and using `JSON.stringify()`.

At this point we need to define the application routes. Since the ExpressJS utilities are not available, we need to use regular expressions on the `request.url` property and a comparison on the `request.method` property to determine how to respond to the request based on the HTTP verb used.

```
'use strict';

module.exports = (req, res, controller) => {
    const { url, method } = req;

    if(url === '/api/posts' && method === 'GET') {
        return controller.index(req, res);
    }

    if(/\/api\/posts\/[a-z0-9]{24}/.test(url) &&
method === 'GET') {
        const id = req.url.split('/')[3];
        return controller.single(req, res, id);
    }

    if(url === '/api/posts' && method === 'POST') {
        return controller.create(req, res);
    }

    if(/\/api\/posts\/[a-z0-9]{24}/.test(url) &&
method === 'PUT') {
        const id = req.url.split('/')[3];
        return controller.update(req, res, id);
    }

    if(/\/api\/posts\/[a-z0-9]{24}/.test(url) &&
method === 'DELETE') {
        const id = req.url.split('/')[3];
        return controller.remove(req, res, id);
    }

    res.writeHead(404, { 'Content-Type':
'application/json' });
    res.end(JSON.stringify({ error: 'Not Found' }));
};
```

As you can see, the use of `if` blocks can only be mitigated using the `return` statement to avoid unnecessary `else` blocks.

Our application will have this final structure:

```
'use strict';

const http = require('http');
const mongoose = require('mongoose');
const PORT = process.env.PORT || 3000;
const PostController =
require('./controllers/PostController');
const postRoutes = require('./routes/posts');
const dbURL = 'mongodb://127.0.0.1:27017/database';

mongoose.connect(dbURL, {useNewUrlParser: true,
useUnifiedTopology: true, useFindAndModify: false });

const app = http.createServer((req, res) => {
    postRoutes(req, res, new PostController());
}).listen(PORT);
```

Conclusion

As you can see, using this solution you cannot avoid some significant redundancies in coding. A solution of this type is not scalable, as if the level of complexity of the API should increase, the required code would increase significantly.