Node.js: implementare API REST da zero

In questo articolo vedremo come implementare una semplice API REST senza usare ExpressJS per dimostrare come questo approccio sia poco pratico.

Definiamo il modello di dati 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);
```

Poichè non avremmo accesso all'oggetto `request.body`, dobbiamo implementare da zero una funzione helper per i metodi HTTP POST e PUT.

```
'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 pratica il flusso di dati della richiesta viene assemblato fino ad ottenere una stringa che verrà convertita in un oggetto JSON.

A questo punto possiamo creare il controller che gestirà le route.

```
'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;
```

Poichè non disponiamo del metodo middleware `response.json()` di ExpressJS, dobbiamo manualmente servire JSON impostando l'header HTTP corretto e usando `JSON.stringify()`.

A questo punto dobbiamo definire le route dell'applicazione. Non disponendo delle utility di ExpressJS, dobbiamo far ricorso alle espressioni regolari sulla proprietà `request.url` e ad un confronto sulla proprietà `request.method` per stabilire come rispondere alla richiesta in base al verbo HTTP usato.

```
'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]+/.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]+/.test(url) && method
=== 'PUT') {
```

```
        const id = req.url.split('/')[3];
        return controller.update(req, res, id);
    }

    if(/\/api\/posts\/[a-z0-9]+/.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' }));
};
```

Come si può notare, il ricorso ai blocchi `if` può solo essere mitigato usando il costrutto `return` per evitare inutili blocchi `else`.

La nostra applicazione avrà questa struttura finale:

```
'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);
```

Conclusione

Come si può notare, usando questa soluzione non si possono evitare alcune notevoli ridondanze nella scrittura del codice. Una soluzione di questo tipo non è scalabile, in quanto qualora dovesse aumentare il livello di complessità delle API, il codice richiesto aumenterebbe notevolmente.