ExpressJS Report

Written by:

- Soufiane Amini

Under the supservision of:

- Pr. Amal Ourdou

Summary

ExpressJS

Figure 1: *ExpressJS* logo

1 What is ExpressJS?

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications APIs.

ExpressJS allows us to create a simple webserver, add middlewares depending on our needs, as well as get and respond to requests to make a REST API.

2 Middlewares

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.

Example 1: A json middleware that transforms the request body into a json object. Example 2: A middleware that verifies that the request is made by an authorized person.

3 Project Showcase

3.a Directory Structure

```
.
├── dist
│   └── index.js
├── index.ts
├── package.json
├── package-lock.json
├── rest.http
└── tsconfig.json

2 directories, 6 files
```

The project contains a package.json file that contains the dependencies and dev dependencies, an index.ts file containing the server application, a tsconfig.json for setting options for the typescript compiler, a dist that contains the transpiled output of the typescript compiler, as well as a rest.http file for testing our server.

3.b Initializing a Node Project

To initialise a node project we run the command

```
npm init -y
```

Next, we set up the scripts to run the code in package.json. We use nodemon to automatically rerun the server whenever changes occur, to facilitate development:

```
npm run watch
```

3.c Install Express

Next, we install Express using the following command:

```
npm i express
```

We can also install types for express using the following command:

```
npm i --save-dev @types/express
```

3.d Set up Express

First we import express, and call the express function to create an express application.

```
import express from "express"
const app = express()
const port = 3001
```

Listing 1: Import express

We will need to work with json in our requests, so we introduce a middleware that parses the body from the request and injects it into the `req.body` field.

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

Listing 2: Use express.json() middleware

Finally, we run the server by listening to the port we set earlier in the program:

```
app.listen(port, () => {
  console.log(`Item storage app listening on port ${port}`)
})
```

Listing 3: Listening on port 3001

3.e Create POST endpoint

This will allows us to add an item to our items list.

```
type Item = {
  id: number
  name: string
}

let items: Item[] = []

app.post("/", (req, res) => {
  console.log(req.body)
  items.push(req.body)
  res.send("Item has been created!")
})
```

Listing 4: Post Endpoint

3.f Create GET endpoint

This endpoint allows us to retrieve all items that are currently stored in the server.

```
app.get("/", (req, res) => {
  res.send(items)
})
```

Listing 5: Get Endpoint

3.g Create GET endpoint by ID

This endpoint allows us to get a specific item by its id.

```
app.get("/:id", (req, res) => {
  const id = +req.params.id
  const item = items.find((i) => i.id === id)

  if (item) {
    res.send(item)
  } else {
    res.send(`Item with id: ${id} not found.`)
  }
})
```

Listing 6: Get Endpoint by ID

3.h Create a PUT endpoint by ID

This endpoint allows us to update an item by its id.

```
app.put("/:id", (req, res) => {
  const id = +req.params.id
  const item = items.find((i) => i.id === id)

  if (item) {
    item.name = req.body.name
    res.send("Item has been updated successfully!")
  } else {
    res.send(`Item with id: ${id} not found.`)
  }
})
```

Listing 7: Put Endpoint

3.i Create a DELETE endpoint

This endpoint allows us to delete an item by its id.

```
app.delete("/:id", (req,res) => {
  const id = +req.params.id
  const exists = items.some((i) => i.id === id)
  if (!exists) {
    res.send(`Item with id: ${id} does not exist.`)
  } else {
    items = items.filter(i => i.id !== id)
    res.send(`Item with id: ${id} has been deleted.`)
  }
})
```

Listing 8: Delete Endpoint

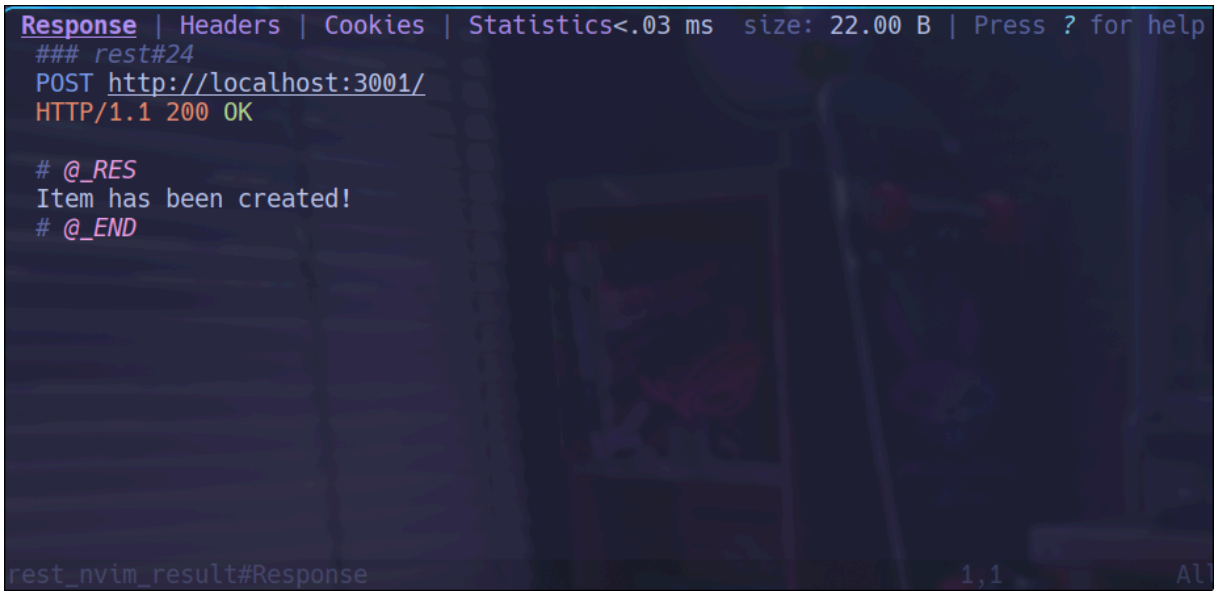
3.j Testing Endpoints with a REST Client

In order to test our endpoints, we're going to use a REST Client similar to POSTMAN, using rest.nvim. In it, we can define a rest.http file that contains multiple different requests on different endpoints, that allows us to test our application.

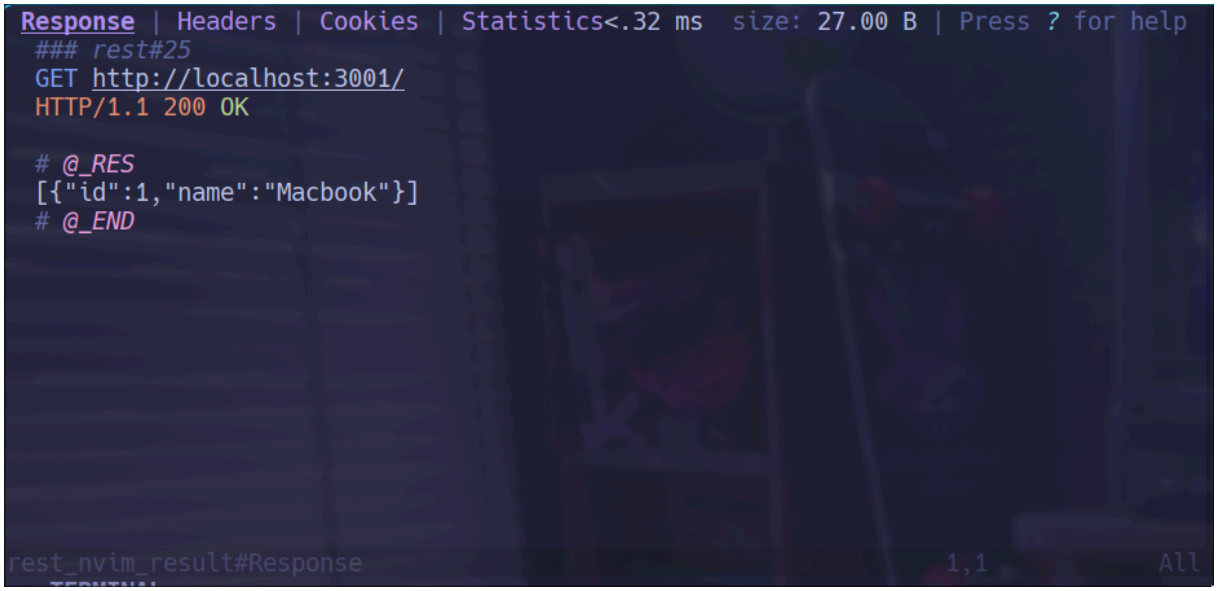
```
POST http://localhost:3001/
Content-Type: application/json

{
  "id": 1,
  "name": "Macbook"
}

###

GET http://localhost:3001/

###

GET http://localhost:3001/1

###

PUT http://localhost:3001/1
Content-Type: application/json

{
  "id": 1,
  "name": "Mactest"
}

###

DELETE http://localhost:3001/1
```

Listing 9: Rest Client

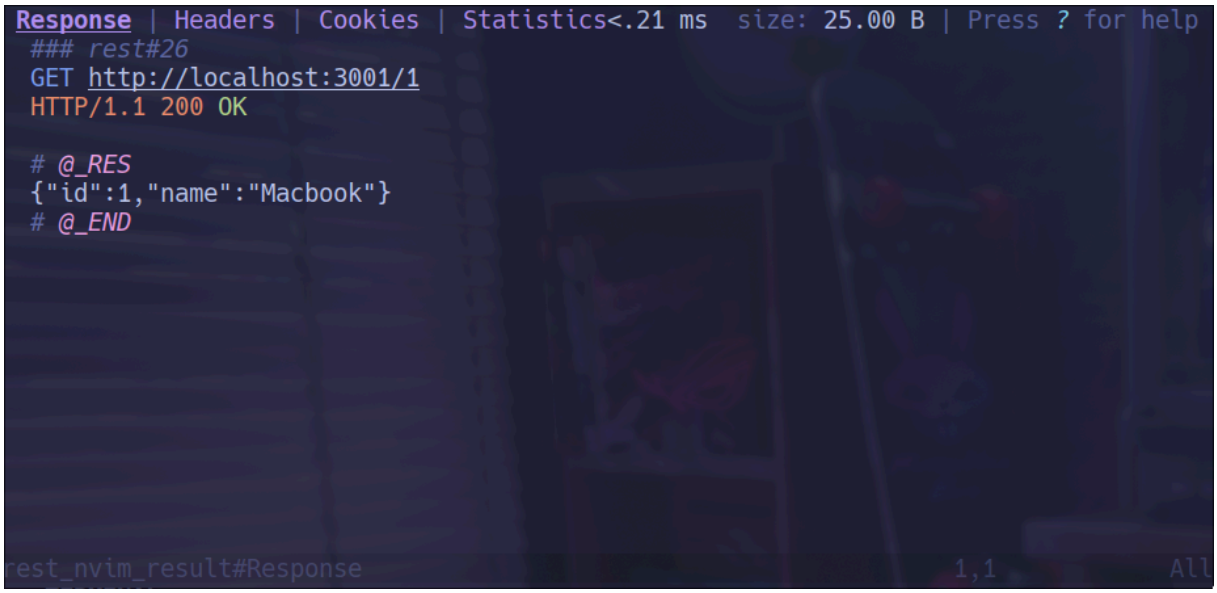
3.k Test POST



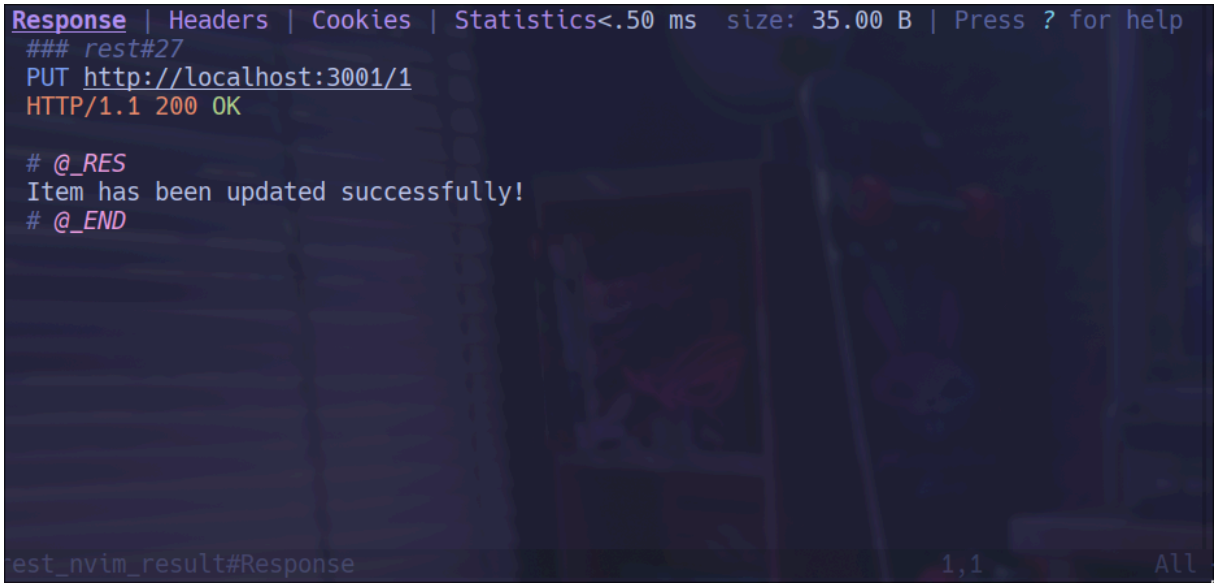
3.l Test GET

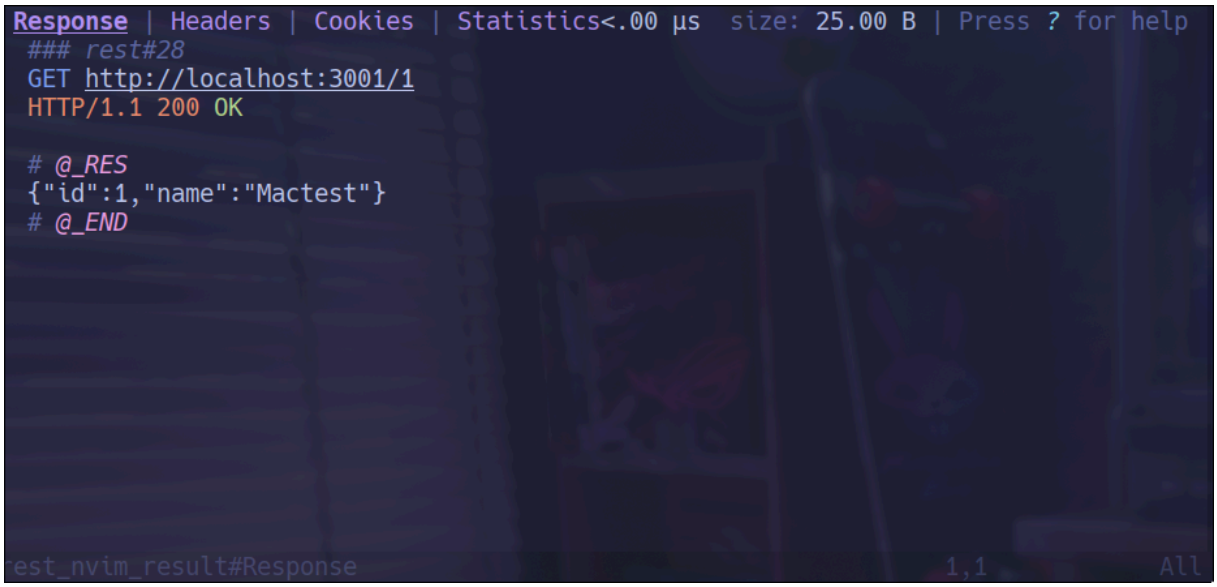


3.m Test GET with ID



3.n Test PUT





3.o Test DELETE