

Université Sultan Moulay Slimane
Ecole Nationale des Sciences Appliquées
- ENSA Khouribga -

TP REPORT

TP 2: EXPRESS JS

Prepared by:

ELHILALI ELMEHDI

Supervisor:

Amal Ourdou

Academic Year: 2024-2025

Contents

1	Introduction	2
1.1	What is Express.js?	2
1.2	What are Middlewares in Express.js?	2
2	Initialization of Node.js and Installing Express	3
2.1	Node.js Initialization	3
2.2	Installing Express.js	3
2.3	Example of code	4
3	Create Item	5
3.1	Create Item Code	5
3.2	Code Explanation	5
3.3	Create Item Postman Request	6
3.4	Create Item Terminal Output	6
4	Get All Items	7
4.1	Get All Items Code	7
4.2	Code Explanation	7
4.3	Get All Items Postman Request	7
5	Get Item By ID	8
5.1	Get Item By ID Code	8
5.2	Code Explanation	8
5.3	Get Item By ID Postman Request	8
6	Update Item	9
6.1	Update Item Code	9
6.2	Code Explanation	9
6.3	Update Postman Request	9
6.4	Update Postman Response	10
7	Delete Item	11
7.1	Delete Item Code	11
7.2	Code Explanation	11
7.3	Delete Postman Request	11
7.4	Delete Postman Response	12
8	Conclusion	13

1 Introduction

1.1 What is Express.js?

Express.js is a fast, unopinionated, and minimalist web framework for Node.js that helps in building web applications and APIs. It simplifies the creation of server-side applications by providing a wide range of features to develop robust web and mobile applications. Express.js handles HTTP requests, routes, and other common tasks necessary to develop scalable server-side applications.

With Express.js, developers can:

- Create APIs to serve dynamic content to client applications.
- Build single-page, multi-page, or hybrid web applications.
- Handle HTTP methods (GET, POST, PUT, DELETE) to manage application logic and CRUD operations.
- Integrate middleware to add additional functionality to requests and responses.

1.2 What are Middlewares in Express.js?

Middlewares in Express.js are functions that have access to the request object, the response object, and the next middleware function in the application's request-response cycle. They are used to modify the request and response objects, terminate the request-response cycle, or pass control to the next middleware in the stack.

In Express.js, middlewares can serve various purposes such as logging, authentication, parsing, error handling, and more.

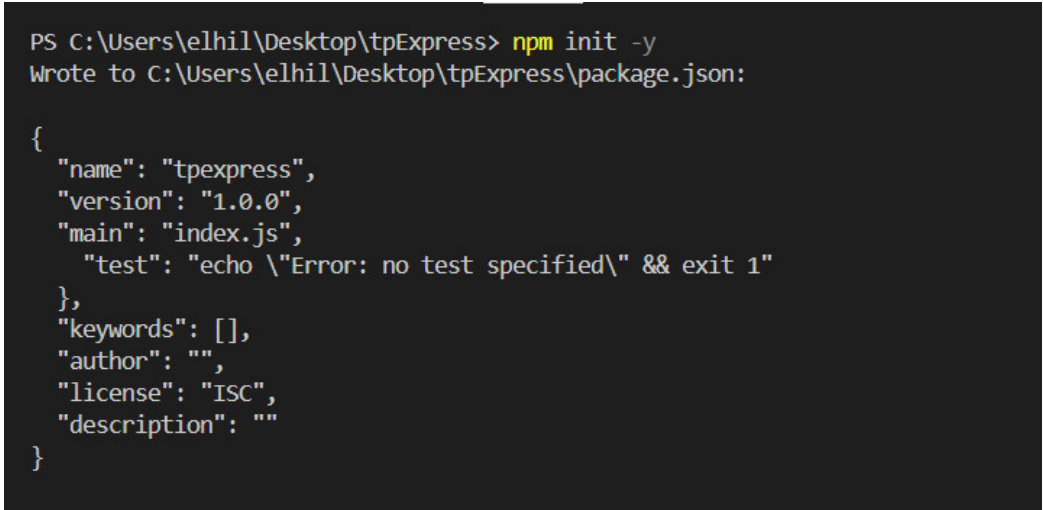
Two examples of middlewares:

- **Body Parsing Middleware:** This is used to parse incoming request bodies in middleware before your handlers. A common example is `express.json()` which parses JSON request bodies and makes the data available via `req.body`.
- **Static File Middleware:** `express.static()` is used to serve static files such as images, CSS, and JavaScript files. It allows Express.js to serve static resources directly without having to create specific routes for them.

2 Initialization of Node.js and Installing Express

2.1 Node.js Initialization

To start a Node.js project, we initialize it with the ‘npm init’ command. This command creates a ‘package.json’ file, which holds metadata relevant to the project and lists dependencies and scripts. Here’s the initialization process:



```
PS C:\Users\elhil\Desktop\tpExpress> npm init -y
Wrote to C:\Users\elhil\Desktop\tpExpress\package.json:

{
  "name": "tpexpress",
  "version": "1.0.0",
  "main": "index.js",
  "test": "echo \"Error: no test specified\" && exit 1",
},
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}
```

In this image, you can see how we set up the basic project structure. After running the command, we fill out the necessary fields for the package configuration.

2.2 Installing Express.js

After initializing the Node.js project, we install Express.js using the ‘npm install express’ command.

```
PS C:\Users\elhil\Desktop\tpExpress> npm install express

added 65 packages, and audited 66 packages in 23s

13 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS C:\Users\elhil\Desktop\tpExpress> |
```

The second image shows the result of successfully installing Express.js. It adds the Express package to 'package.json' and creates the required files for developing our Express application.

2.3 Example of code

```
server.js  X
server.js > ...
1  const express = require('express');
2  const app = express();
3  const port = 5001;
4
5  // Middleware to parse incoming JSON requests and convert them to JavaScript objects
6  app.use(express.json());
7
8  app.listen(port, () => {
9    console.log(`Node.js server started on port : ${port}`);
10 });
```

3 Create Item

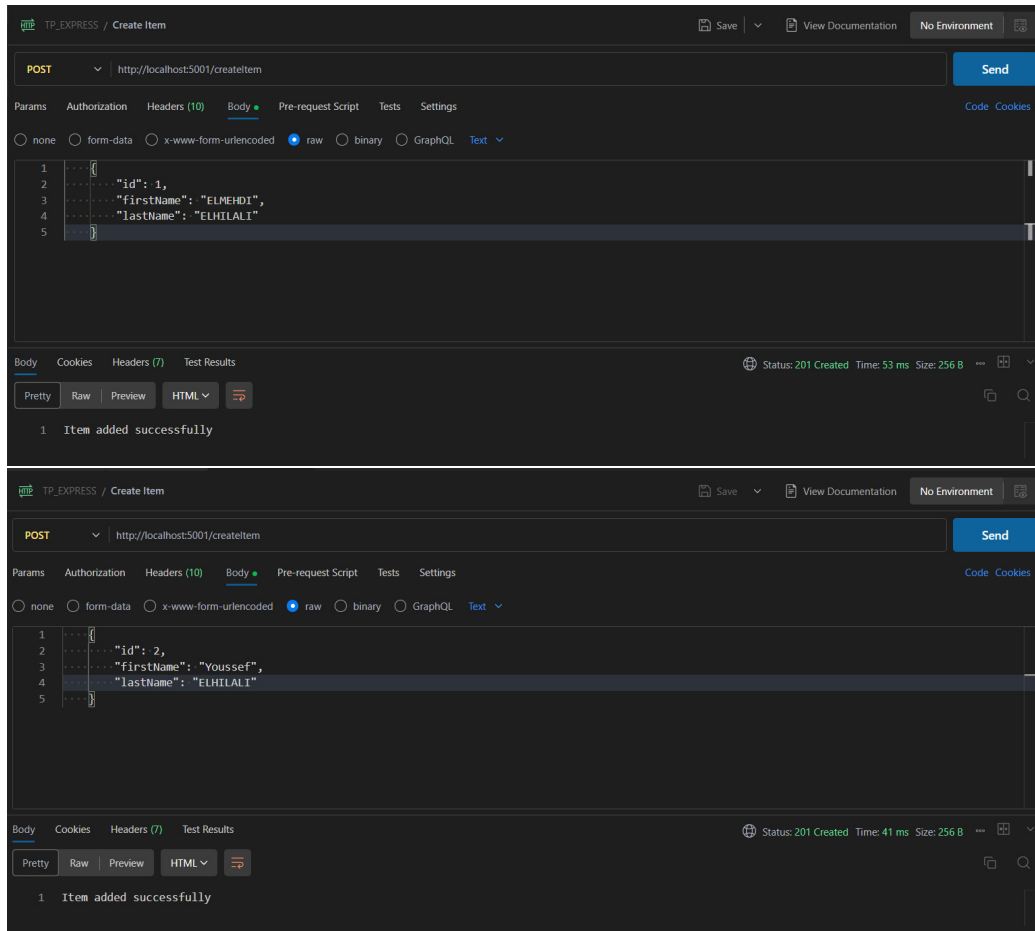
3.1 Create Item Code

```
10 // POST : Create Item
11 app.post('/createItem', (req, res) => {
12   ⚡ const item = req.body;
13
14   // Check if the request body contains the correct data
15   if (!item || Object.keys(item).length === 0) {
16     return res.status(400).send('Invalid item');
17   }
18
19   items.push(item); // Add the item to the items array
20
21   console.log(items); // Log the items array to ensure it contains the new item
22
23   res.status(201).send('Item added successfully');
24 });
25
```

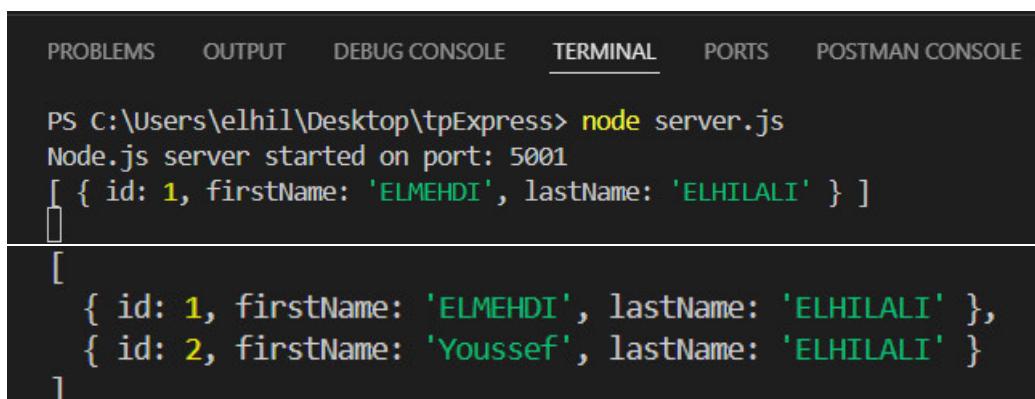
3.2 Code Explanation

The 'POST /createItem' endpoint is responsible for adding a new item to our in-memory array. The request body is parsed using the 'express.json()' middleware, and the data is pushed into the 'items' array. If the request doesn't contain valid data, a 400 status is returned. Otherwise, the item is added, and a success response (status 201) is sent.

3.3 Create Item Postman Request



3.4 Create Item Terminal Output



4 Get All Items

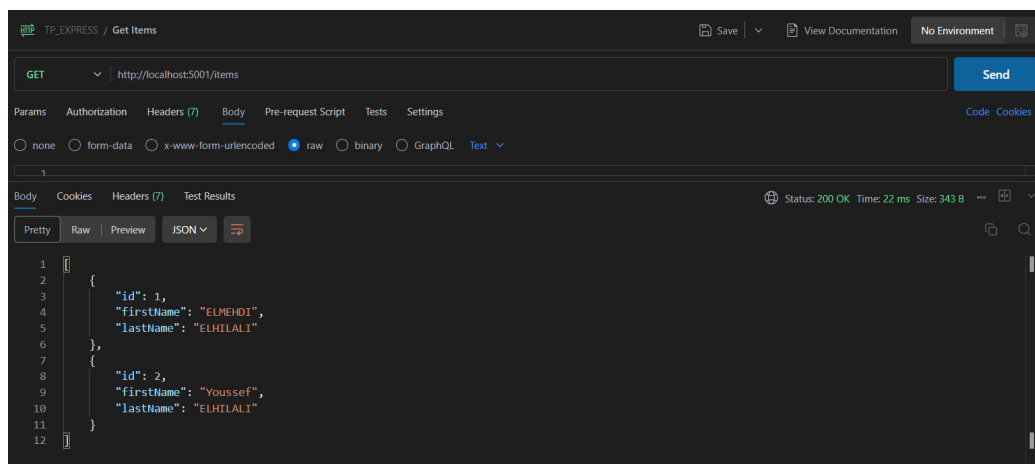
4.1 Get All Items Code

```
26 // GET : retrieve all items
27 app.get('/items', (req, res) => {
28   console.log('GET request received for /items');
29   res.json(items); // Return all items as a JSON response
30 });
31
```

4.2 Code Explanation

The 'GET /AllItems' endpoint retrieves all the items stored in the 'items' array. It simply returns the array in JSON format, allowing clients to view all the items in the current session. This is useful for displaying a list of all available items.

4.3 Get All Items Postman Request



5 Get Item By ID

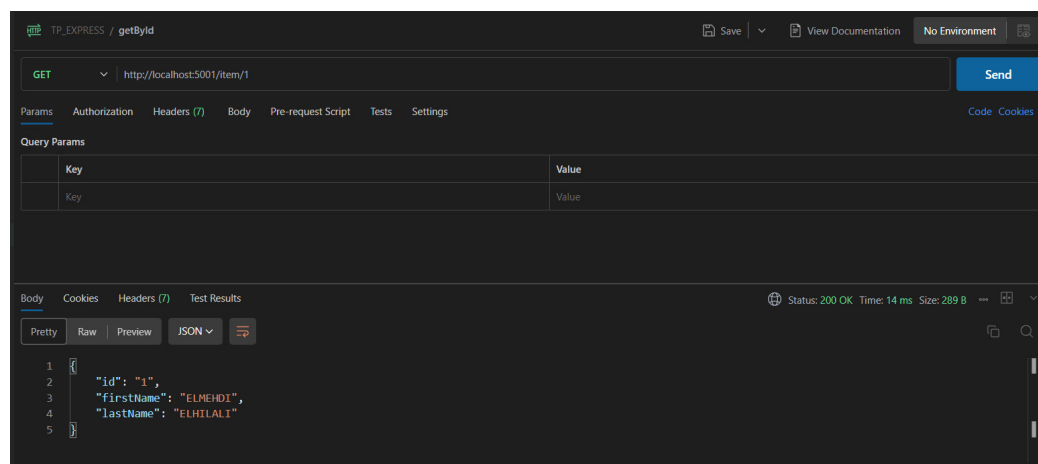
5.1 Get Item By ID Code

```
32 // GET : Endpoint by ID (Retrieve a Specific Item)
33 app.get('/item/:id', (req, res) => {
34   const { id } = req.params;
35   const item = items.find(i => i.id === id);
36   if (item) {
37     res.json(item);
38   } else {
39     res.status(404).send('Item not found');
40   }
41 });
42
43
44 app.listen(port, () => {
45   console.log(`Node.js server started on port: ${port}`);
46 });
```

5.2 Code Explanation

The 'GET /item/:id' endpoint retrieves a specific item by its ID. It searches the 'items' array for an item that matches the given ID. If found, the item is returned as a JSON response. If not, a 404 error is returned, indicating that the item was not found.

5.3 Get Item By ID Postman Request



6 Update Item

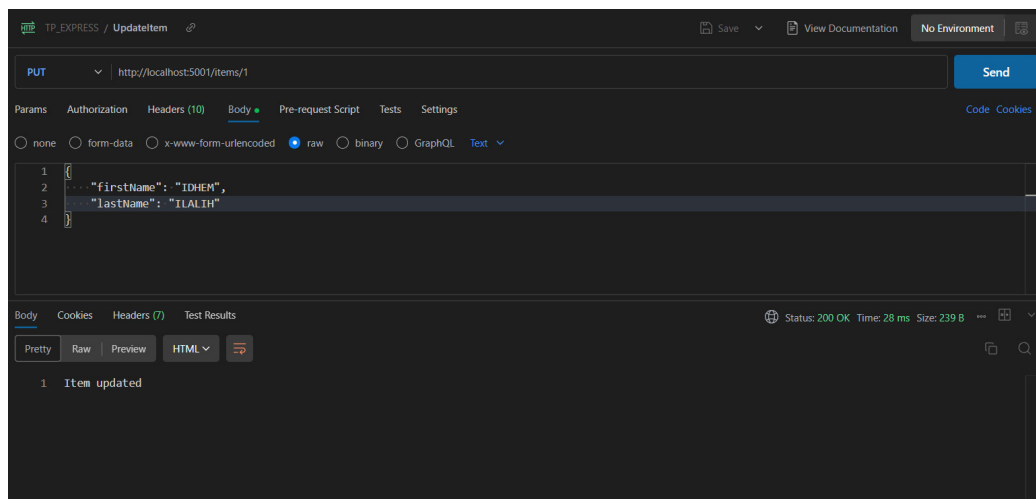
6.1 Update Item Code

```
43 // PUT : Endpoint (Update an Item)
44 app.put('/items/:id', (req, res) => {
45   const { id } = req.params;
46   const index = items.findIndex(i => i.id === id); // Compare as string
47
48   if (index !== -1) {
49     // Merge existing item with updated data
50     items[index] = { ...items[index], ...req.body };
51     res.send('Item updated');
52   } else {
53     res.status(404).send('Item not found');
54   }
55 });
56
```

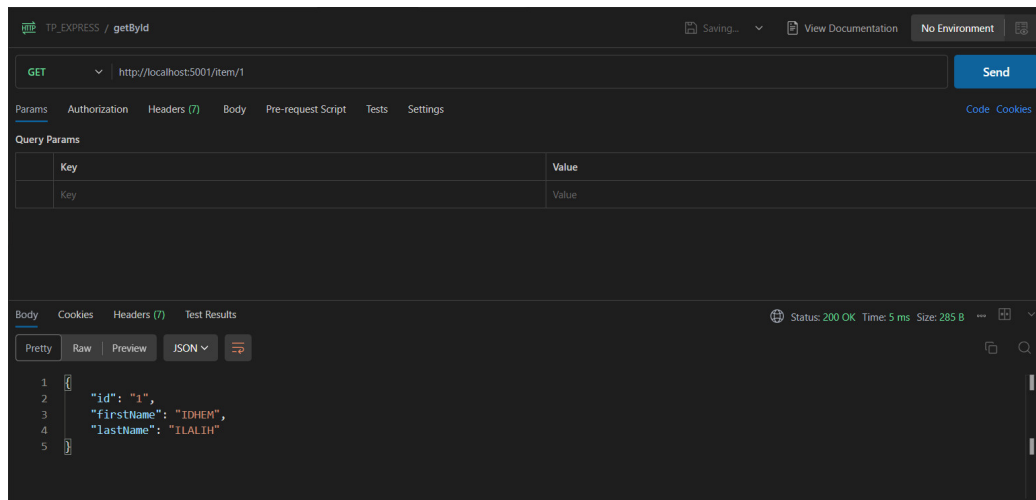
6.2 Code Explanation

The 'PUT /items/:id' endpoint allows the update of a specific item. It searches for the item by ID, and if found, it merges the existing item with the new data provided in the request body. If the item is updated successfully, a confirmation message is sent. If not, a 404 error is returned.

6.3 Update Postman Request



6.4 Update Postman Response



7 Delete Item

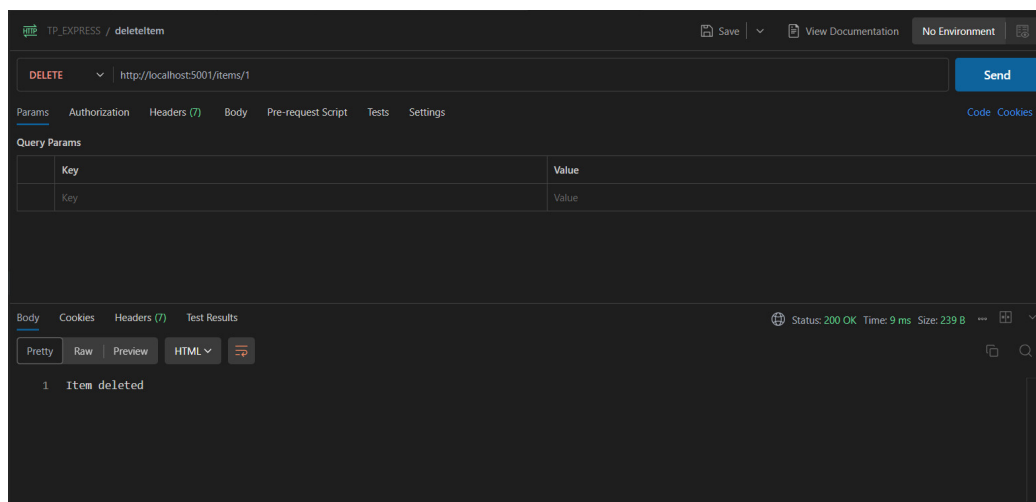
7.1 Delete Item Code

```
57 // DELETE
58 app.delete('/items/:id', (req, res) => {
59   const { id } = req.params;
60   const index = items.findIndex(i => i.id === id);
61   if (index !== -1) {
62     items.splice(index, 1);
63     res.send('Item deleted');
64   } else {
65     res.status(404).send('Item not found');
66   }
67 });
68
```

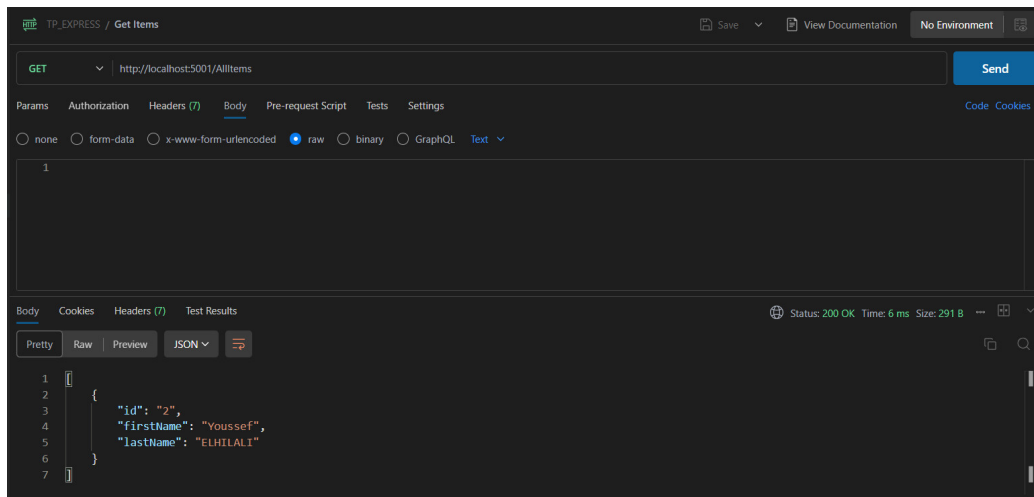
7.2 Code Explanation

The 'DELETE /items/:id' endpoint allows for deleting an item from the 'items' array by ID. It searches for the item, and if found, removes it from the array. A success message is sent if the item is deleted, and a 404 error is returned if the item is not found.

7.3 Delete Postman Request



7.4 Delete Postman Response



8 Conclusion

Through this TP, we gained hands-on experience with the fundamentals of creating a RESTful API using Express.js. We explored various essential features such as handling HTTP methods (GET, POST, PUT, DELETE), setting up routes, and managing data with requests and responses. Additionally, we implemented middleware for tasks like parsing and serving static files, which enhanced our understanding of how middleware works in Express.js.