

# TP (Express.JS)

## 1. What is Express.js and What Can We Make with It?

**Express.js** is a minimalist and flexible web application framework for **Node.js**. It provides a robust set of features for building web and mobile applications, allowing developers to create both APIs and full-stack applications. Express simplifies tasks such as routing, handling HTTP requests and responses, managing middleware, and more.

You can use **Express.js** to:

- Create **RESTful APIs**
- Build **web applications**
- Serve **static files** (like HTML, CSS, JS)
- Manage real-time data with frameworks like **Socket.io**
- Build **single-page applications** (with tools like React or Angular)

## 2. What are Middlewares and How Are They Used in Express.js?

**Middleware** functions in **Express.js** are functions that have access to the request (`req`), response (`res`), and the next middleware function in the application's request-response cycle. They can modify `req` and `res` objects, execute any code, and end the request-response cycle or call the next middleware function.

*Two Examples of Middleware in Express.js:*

1. **Custom Middleware Example:** A middleware function that logs every request made to the server.

```
2. const express = require('express');
3. const app = express();
4.
5. // Custom middleware to log each request
6. app.use((req, res, next) => {
7.   console.log(`${req.method} request for '${req.url}'`);
8.   next(); // Call the next middleware or route handler
9. });
10.
11. app.get('/', (req, res) => {
12.   res.send('Hello, World!');
13. });
14.
15. app.listen(3000, () => {
```

```
16. console.log('Server running on port 3000');
17.});
18.
```

**2. Built-in Middleware Example:** Using the built-in middleware `express.json()` to parse JSON request bodies.

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

// Middleware to parse JSON request bodies
app.use(express.json());

app.post('/data', (req, res) => {
  console.log(req.body); // Access the parsed JSON body
  res.send('Data received');
});

app.listen(3000, () => {
  console.log('Server running on port 3000');
});
```

## Creating a Simple CRUD Application with Express.js

1. Create a Project Directory
2. Initialize a Node.js Project
3. Install Express

```
C:\Users\Nessrine\Desktop\S5\JS Mobile\TP2>mkdir express-crud-app
C:\Users\Nessrine\Desktop\S5\JS Mobile\TP2>cd express-crud-app
C:\Users\Nessrine\Desktop\S5\JS Mobile\TP2\express-crud-app>npm init -y
Wrote to C:\Users\Nessrine\Desktop\S5\JS Mobile\TP2\express-crud-app\package.json:

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

C:\Users\Nessrine\Desktop\S5\JS Mobile\TP2\express-crud-app>npm install express
added 65 packages, and audited 66 packages in 3s

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

found 0 vulnerabilities
```

## 4- Set Up Express: Run the Server Using *“app.listen”*

```
const express = require('express');
const app = express();
const PORT = 3000;

// Middleware to parse JSON request bodies
app.use(express.json());

// Start the server
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

## 5- Create a POST Endpoint (Add an Item)

```
4- let items = []; // Local variable to store items
5-
6- // POST endpoint to add an item
7- app.post('/items', (req, res) => {
8-   const newItem = req.body;
9-   items.push(newItem);
10-   res.status(201).send('Item added');
11- });
```

## 6. Create a GET Endpoint (Retrieve All Items)

```
// GET endpoint to retrieve all items
app.get('/items', (req, res) => {
  res.json(items);
});
```

## 7. Create a GET Endpoint by ID (Retrieve a Specific Item)

```
// GET endpoint to retrieve an item by ID
app.get('/items/:id', (req, res) => {
  const id = parseInt(req.params.id);
  const item = items.find(i => i.id === id);
  if (item) {
    res.json(item);
  } else {
    res.status(404).send('Item not found');
  }
});
```

## 8. Create a PUT Endpoint (Update an Item)

```
// PUT endpoint to update an item by ID
app.put('/items/:id', (req, res) => {
  const id = parseInt(req.params.id);
  const index = items.findIndex(i => i.id === id);
  if (index !== -1) {
    items[index] = req.body;
    res.send('Item updated');
  } else {
    res.status(404).send('Item not found');
  }
});
```

## 9. Create a DELETE Endpoint (Delete an Item)

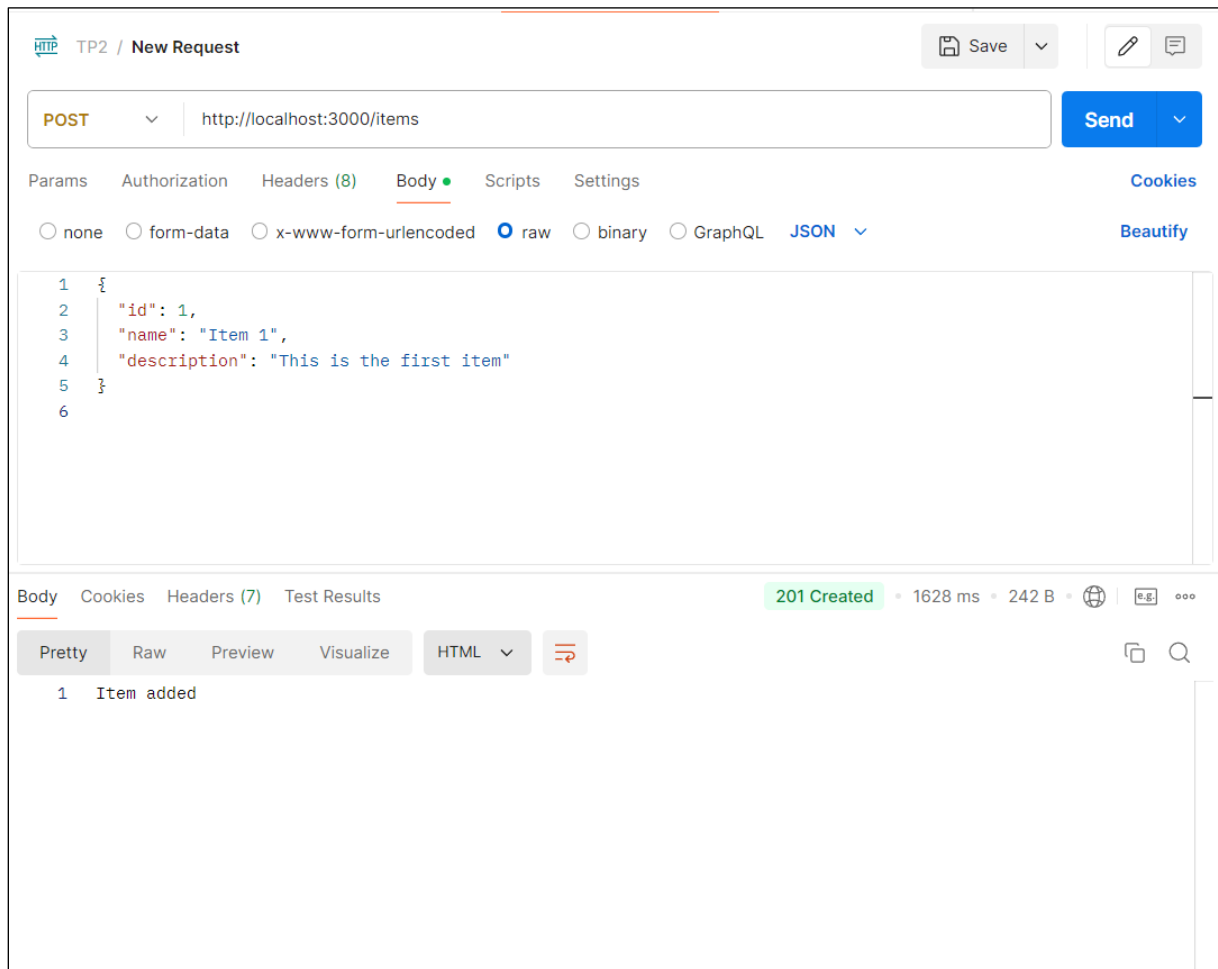
```
// DELETE endpoint to delete an item by ID
app.delete('/items/:id', (req, res) => {
  const id = parseInt(req.params.id);
  const index = items.findIndex(i => i.id === id);
  if (index !== -1) {
    items.splice(index, 1);
    res.send('Item deleted');
  } else {
    res.status(404).send('Item not found');
  }
});
```

## 10. Start the Server

```
C:\Users\Nessrine\Desktop\S5\JS Mobile\TP2\express-crud-app>node index.js
Server running on port 3000
```

## 11. Test the Endpoints Using Postman

### - Tester le POST Endpoint



## Tester le GET Endpoint pour récupérer tous les items

The screenshot displays a REST client interface with the following components:

- Top Bar:** Includes tabs for 'Overview', 'TP2', and 'GET New Request'. The 'No environment' dropdown is also visible.
- Request Section:**
  - Method:** 'GET' (indicated by a green icon).
  - URL:** 'http://localhost:3000/items'.
  - Buttons:** 'Save' and 'Send' (blue button).
- Request Body Section:**
  - Params:** 'none' (selected).
  - Authorization:** Not set.
  - Headers:** 6 headers (indicated by a green icon).
  - Body:** 'raw' (selected).
  - Scripts:** Not set.
  - Settings:** Not set.
  - Buttons:** 'Cookies' and 'Beautify'.
- Response Section:**
  - Status:** '200 OK' (green background).
  - Time:** '58 ms'.
  - Size:** '300 B'.
  - Buttons:** 'Pretty', 'Raw', 'Preview', 'Visualize', 'JSON' (dropdown), and a refresh icon.
  - Body:** A JSON array containing one object:

```
1 [
2   {
3     "id": 1,
4     "name": "Item 1",
5     "description": "This is the first item"
6   }
7 ]
```

## 5. Tester le GET Endpoint pour récupérer un item par ID

The screenshot displays a REST client interface with the following components:

- Header:** Shows the HTTP method **GET** and the URL **http://localhost:3000/items/1**. A **Send** button is located to the right.
- Body Tab:** The **Body** tab is selected, showing the request body as the string **1**.
- Response Section:** Below the request body, the response status is **200 OK** with a response time of **59 ms** and a size of **298 B**. The response body is displayed in **JSON** format, showing a successful retrieval of an item.
- Response Body (JSON):**

```
1 {  
2   "id": 1,  
3   "name": "Item 1",  
4   "description": "This is the first item"  
5 }
```



## 6. Tester le PUT Endpoint pour mettre à jour un item

The screenshot shows a REST client interface with the following components:

- Top Bar:** Includes tabs for 'Overview', 'TP2', and 'PUT New Request'. The 'PUT New Request' tab is active. On the right, it says 'No environment'.
- Request Bar:** Shows the HTTP method 'PUT' and the URL 'http://localhost:3000/items/1'. A 'Send' button is on the right.
- Request Body:** The 'Body' tab is selected. It shows a JSON object: 

```
{  "id": 1,  "name": "Updated Item 1",  "description": "This is the updated first item"}
```

. The format is set to 'JSON'.
- Response Bar:** Shows the status '200 OK', response time '44 ms', and size '239 B'. It also includes icons for cookies, headers, and test results.
- Response Body:** The 'Body' tab is selected. It shows the response text: '1 Item updated'.

## 7. Tester le DELETE Endpoint pour supprimer un item

The screenshot shows a REST client interface with the following details:

- Overview** tab selected.
- TP2** environment selected.
- DEL New Request** button.
- HTTP** method selected.
- TP2 / New Request** title.
- Save** button.
- Send** button.
- DELETE** method selected.
- http://localhost:3000/items/1** URL.
- Params**, **Authorization**, **Headers (8)**, **Body** (selected), **Scripts**, **Settings** tabs.
- Body** tab selected.
- JSON** format selected.
- Body** content: 

```
1 {
2   "id": 1,
3   "name": "Updated Item 1",
4   "description": "This is the updated first item"
5 }
6
```
- Body**, **Cookies**, **Headers (7)**, **Test Results** tabs.
- 200 OK** status.
- 20 ms** time.
- 239 B** size.
- HTML** format selected.
- Item deleted** response body.