UNIVERSITÉ SULTAN MOULAY SLIMANE

École Nationale des Sciences Appliquées de Khouribga

Département Mathématiques et Informatique

Filière : Génie informatique

TP2 : Express JS

Réalisé par : IDM'BARK Loubna

ENSAK 2024/2025

Table des matières

Table des figures

Chapitre 1

What is Express.js and What Can We Do With It ?

Express.js is a fast, unopinionated, and minimalist web framework for Node.js. It is designed to simplify the process of building web applications and APIs by providing a lightweight layer of web functionalities built on top of Node.js's core features. **Key Features of Express.js :**

- **Routing :** Helps define how an application responds to different HTTP requests (GET, POST, PUT, DELETE, etc.) on different URLs or paths.

- **Middleware Support :** Enables you to manage the request-response cycle by inserting middleware functions to handle requests, responses, and any processing in between.

- **Templates :** Can render HTML pages based on templates, such as Pug, EJS, or Handlebars.

- **Middleware-based pipelines** for handling requests before they are completed

.

- **Easy integration with databases :** Works seamlessly with various databases such as MongoDB, MySQL, PostgreSQL, etc

- **Static File Serving :** Can serve static files (like images, CSS files, and JavaScript files) easily

.

Chapitre 2

What Can We Build with Express.js ?

1. **Web Applications :** Express.js simplifies the development of web apps, whether they are single-page applications (SPAs) or complex multi-page web applications.
2. **RESTful APIs** : Express.js makes it easy to create REST APIs to interact with databases or serve data to front-end applications.
3. **Real-Time Chat Applications :** Paired with libraries like Socket.io, Express.js can help in building real-time chat applications.
4. **Middleware Pipelines :** Express allows you to structure middleware pipelines to handle user authentication, session management, and other HTTP request-related tasks.
5. **Microservices :** Using Express.js, you can build microservices that are fast, lightweight, and scalable.

In summary, Express.js provides the tools to create anything from a simple website to a fully functional API or complex web app with real-time features. It enables developers to rapidly build and deploy powerful web services while maintaining flexibility and control over application design.

Figure 2.1 – Express js

Chapitre 3

What Are Middlewares in Express.js ?

A **middleware in Express.js** is a function that has access to the request object (req), the response object (res), and the next middleware function in the applications request-response cycle. Middleware functions are used to modify the request and response objects, run any code, make changes to the application, or end the request-response cycle

Middleware can perform tasks like :

- Logging every request made to the server
- Validating and parsing incoming requests
- Handling authentication and authorization
- Managing sessions or cookies

In Express.js, middleware functions are executed in sequence. They can either pass control to the next middleware function using next(), or terminate the cycle by sending a response to the client.

3.1 Types of Middlewares :

1. **Application-level middleware :** Applied to every request made to the application or to specific routes.
2. **Router-level middleware :** Used for specific router instances.
3. E**rror-handling middleware :** Specifically designed to handle errors.
4. **Built-in middleware :** Provided by Express, such as express.json() for parsing JSON.
5. **Third-party middleware :** Middleware from external libraries like body-parser, morgan (for logging), etc.

Chapitre 4

Creating a simple CRUD application

4.1 Step 1 : Create a Project Directory

4.2 Step 2 : Install Express

4.3 Step 3 : Set Up Express

Create an index.js file, and set up the Express server :

```
const express = require("express");
const app = express();
const port = 3000;

app.use(express.json());

app.listen(port, () => {
  console.log(`Server is running on http://localhost:${port}`);
});
let items = [];
```

FIGURE 4.1 – Set Up Express

4.4 Step 5 : Create Crud Endpoints

Create

```
//Create
app.post("/items", (req, res) => {
  const item = req.body;
  items.push(item);
  res.status(201).send(`Item added: ${JSON.stringify(item)}`);
});
```

FIGURE 4.2 – create

Get

```
//update
app.put("/items/:id", (req, res) => {
  const id = parseInt(req.params.id, 10);
  const index = items.findIndex((i) => i.id === id);

  if (index !== -1) {
    items[index] = { ...items[index], ...req.body };
    res.status(200).json(items[index]);
  } else {
    res.status(404).send("Item not found");
  }
});
```

FIGURE 4.3 – get

Update

```
//getAll
app.get("/items", (req, res) => {
  res.status(200).json(items);
});

//getById
app.get("/items/:id", (req, res) => {
  const id = parseInt(req.params.id, 10);
  const item = items.find((i) => i.id === id);
  if (item) {
    res.status(200).json(item);
  } else {
    res.status(404).send("Item not found");
  }
});
```

FIGURE 4.4 – Update

Delete

```
//delete
app.delete("/items/:id", (req, res) => {
  const id = parseInt(req.params.id, 10);
  const index = items.findIndex((i) => i.id === id);

  if (index !== -1) {
    items.splice(index, 1);
    res.status(200).send("Item deleted");
  } else {
    res.status(404).send("Item not found");
  }
});
```

FIGURE 4.5 – Delete

Chapitre 5

Testing with Postman

5.1 Testing POST

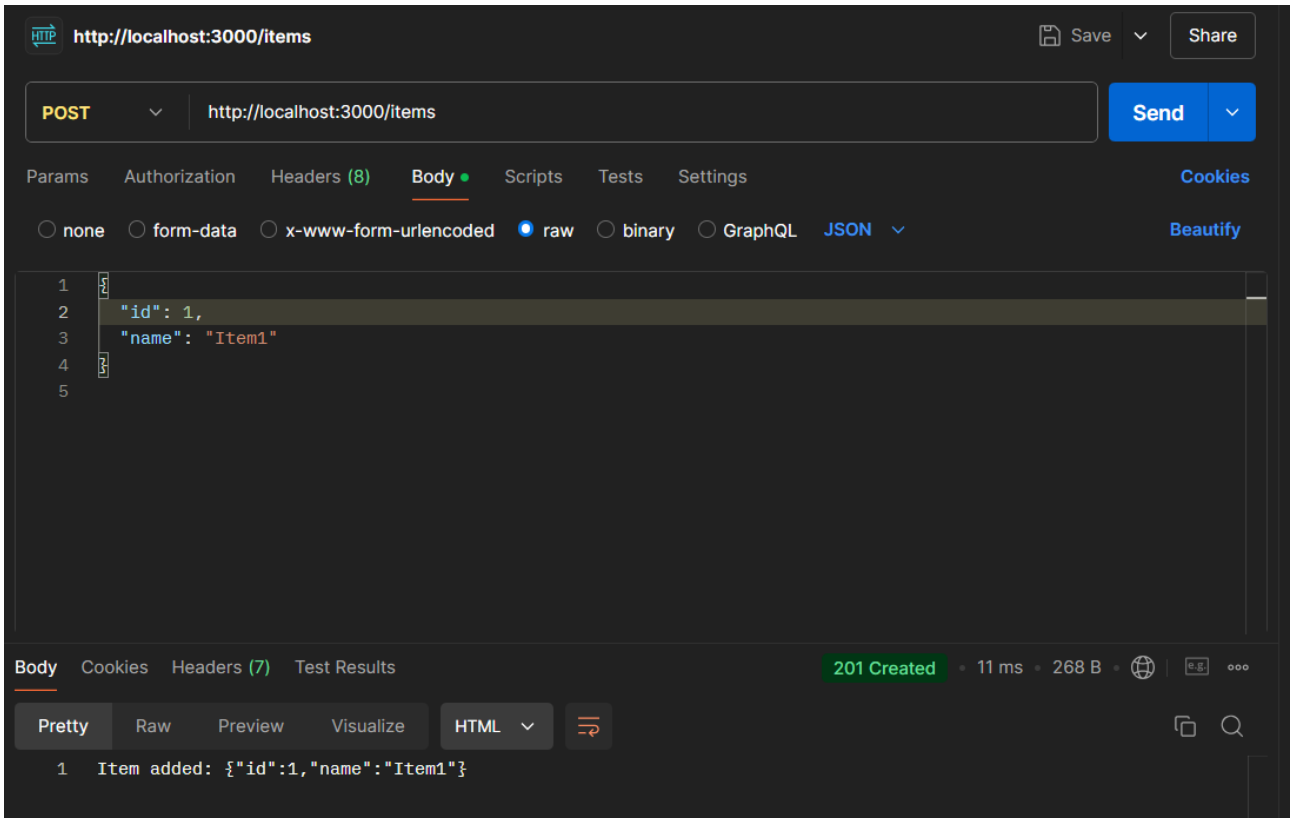

FIGURE 5.1 – postman post

5.2 Testing GET

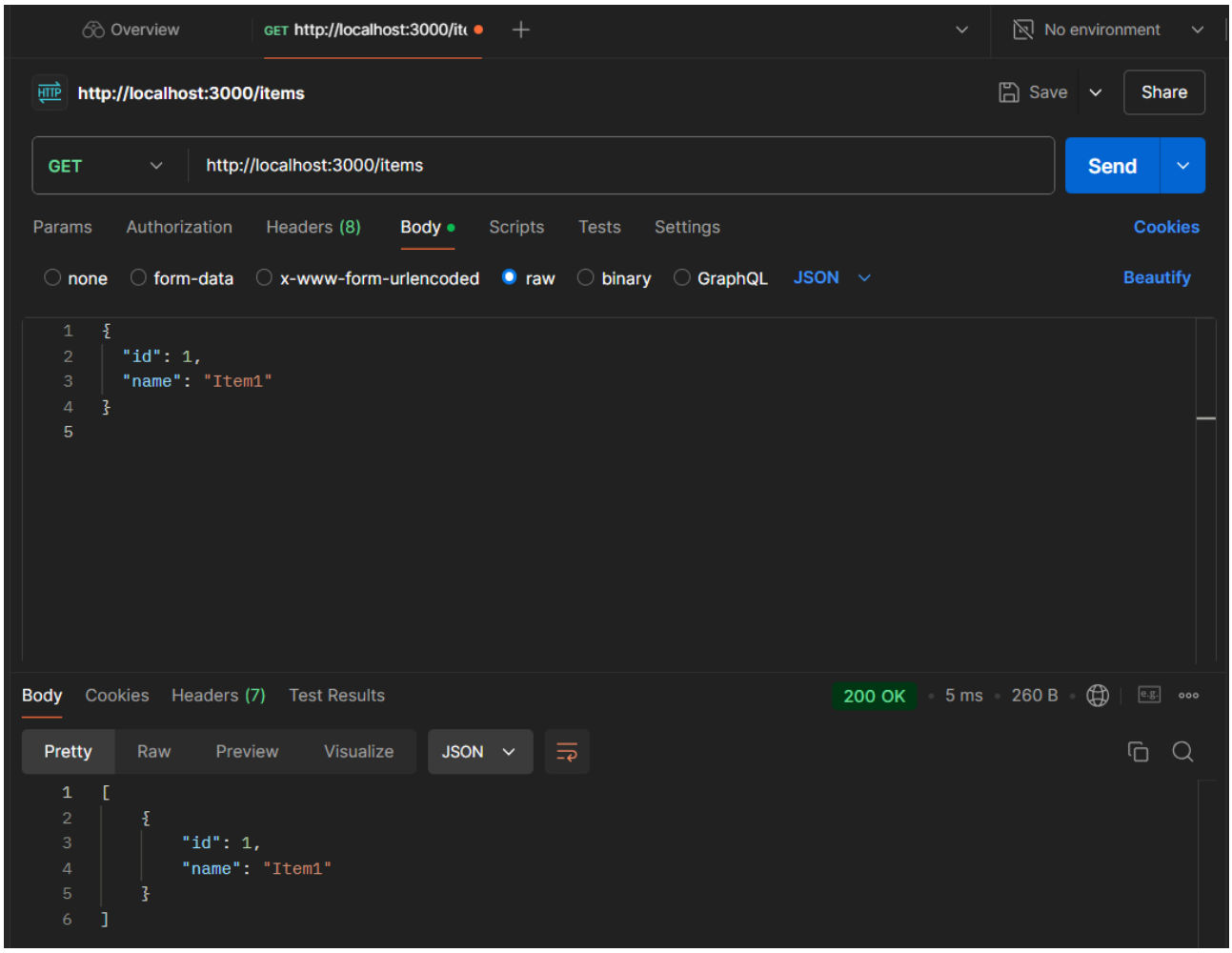

FIGURE 5.2 – postman get

5.3 Testing GET by ID

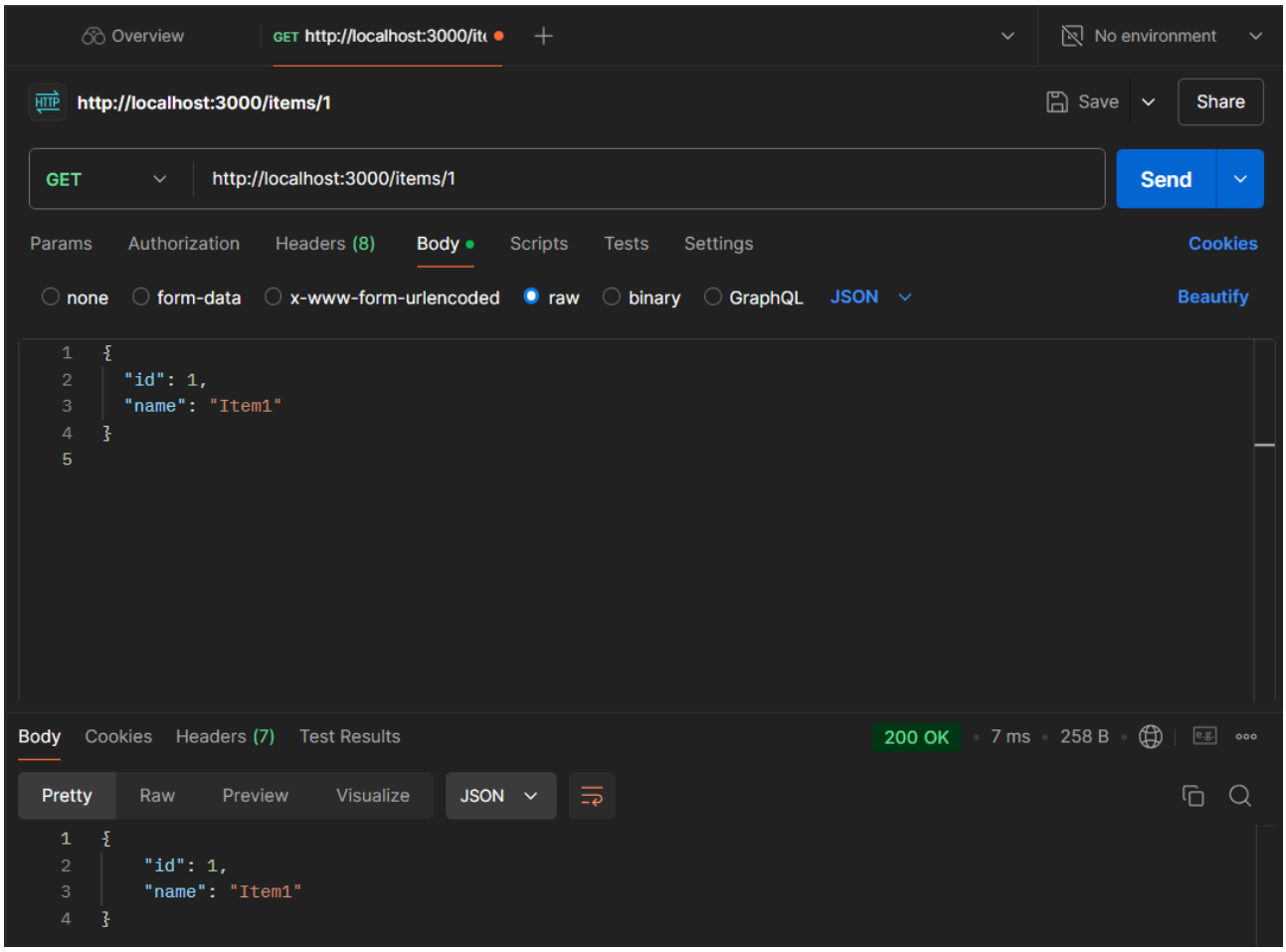

FIGURE 5.3 – postman get by id

5.4 Testing PUT

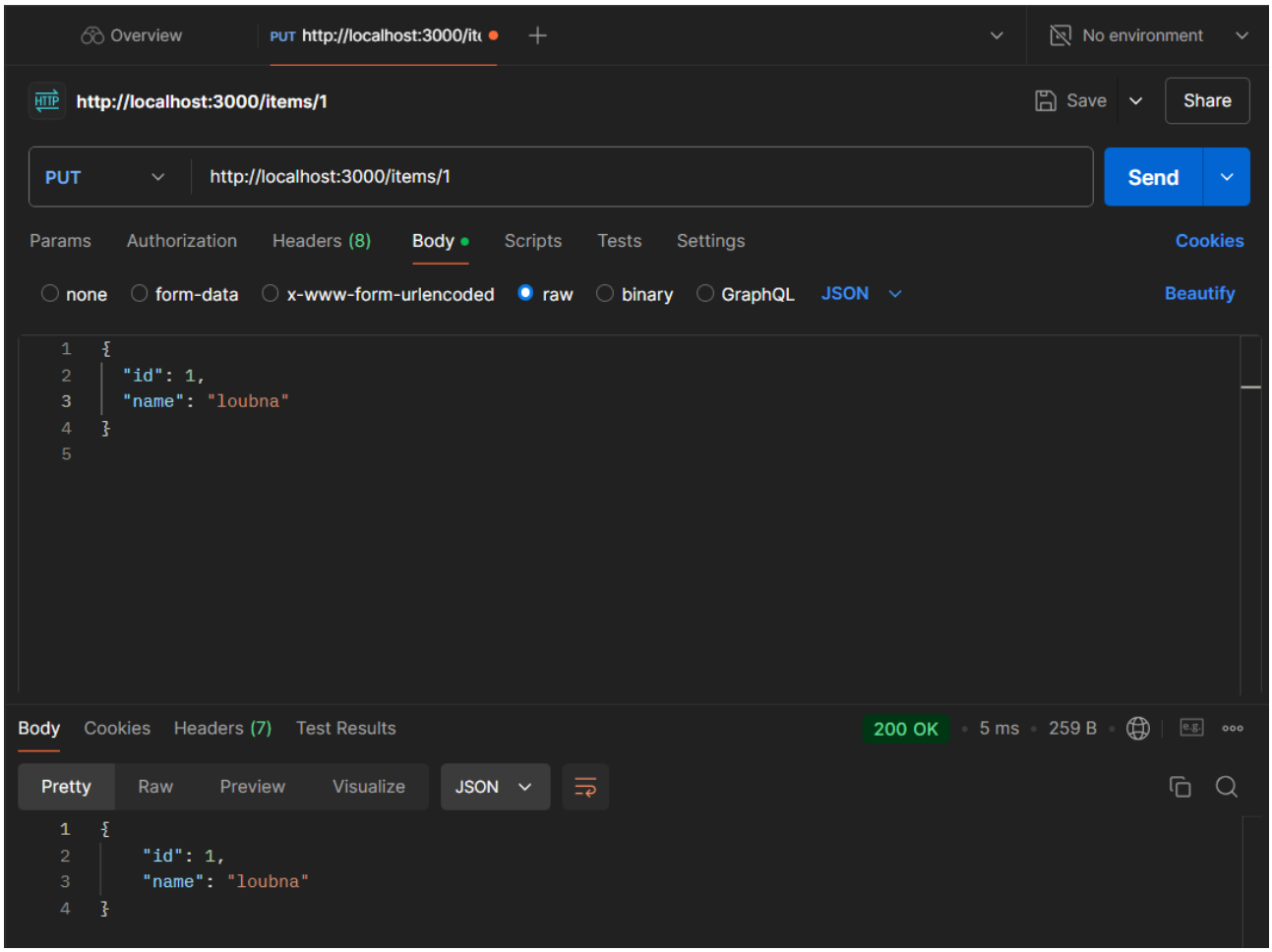

FIGURE 5.4 – postman Update

Testing DELETE

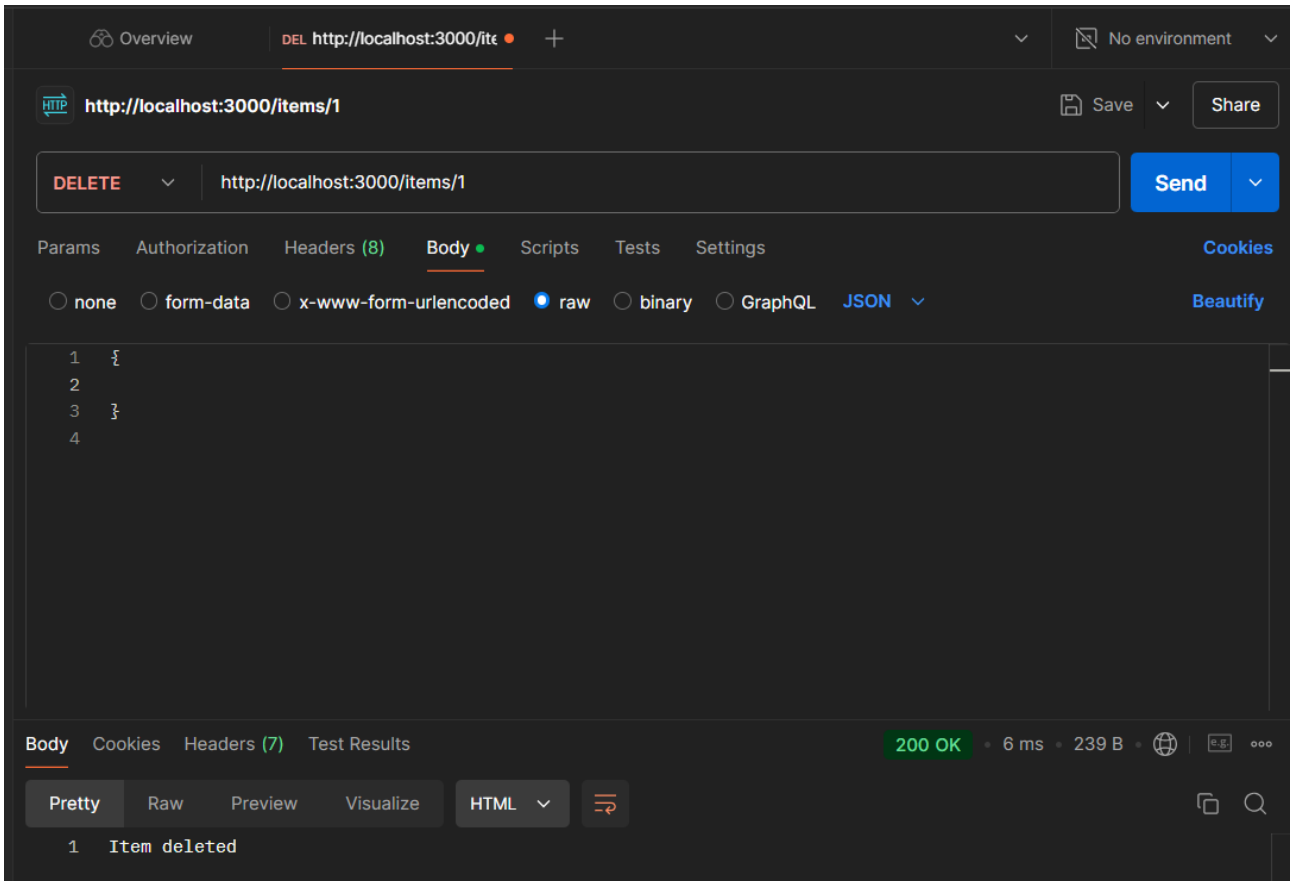

FIGURE 5.5 – postman Delete