TP2 : Express.js

Realized by
ASSAG Khadija

Questions :

1- Express.js is a web application framework for Node.js, designed for building web applications and APIs. It simplifies the process of handling HTTP requests, routing, and middleware integration, allowing developers to create robust and scalable server-side applications quickly. With its minimalist structure, Express provides essential features such as routing, request and response handling, and support for various middleware, making it a popular choice for web development.

2- In Express.js, middleware are functions that have access to the request object (req), the response object (res).

Example 1: Body Parser Middleware

The principal purpose of the Body Parser middleware in Express.js is to parse the body of incoming HTTP requests, making it easier to access the data sent by clients.

Example 2 : Blogger

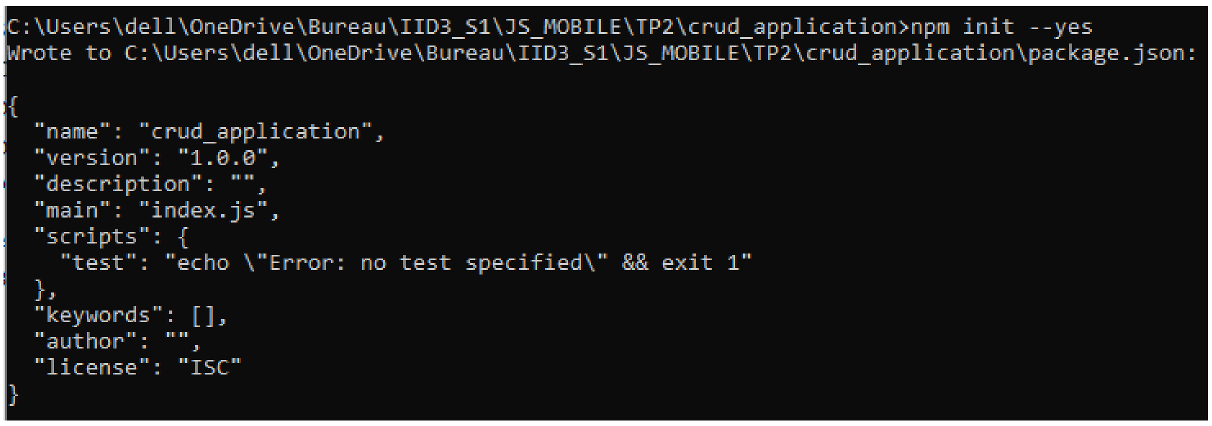
Logger middleware in Express.js is a custom middleware function that records details about incoming requests. This can be useful for debugging, monitoring, and analyzing how your application is being used.

Crud Application :

1- Create the folder crud_application

2-Create a Node.js project

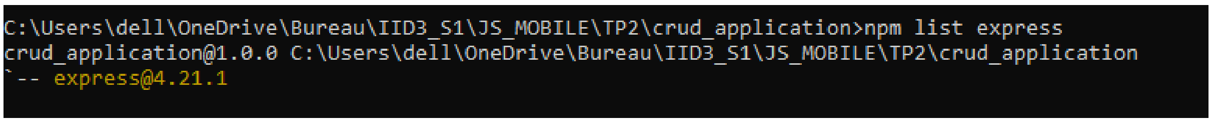
Using the command ligne npm init --yes we create a Node.js project named



```
C:\Users\dell\OneDrive\Bureau\IID3_S1\JS_MOBILE\TP2\crud_application>npm init --yes
Wrote to C:\Users\dell\OneDrive\Bureau\IID3_S1\JS_MOBILE\TP2\crud_application\package.json:

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

3- Install Express

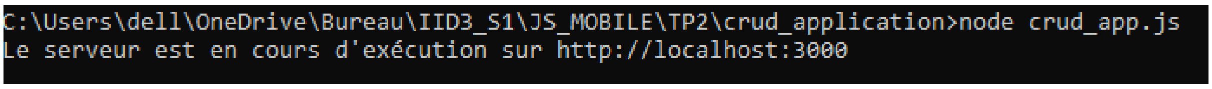
Use npm list express to verify the version of Express installed



```
C:\Users\dell\OneDrive\Bureau\IID3_S1\JS_MOBILE\TP2\crud_application>npm list express
crud_application@1.0.0 C:\Users\dell\OneDrive\Bureau\IID3_S1\JS_MOBILE\TP2\crud_application
`-- express@4.21.1
```

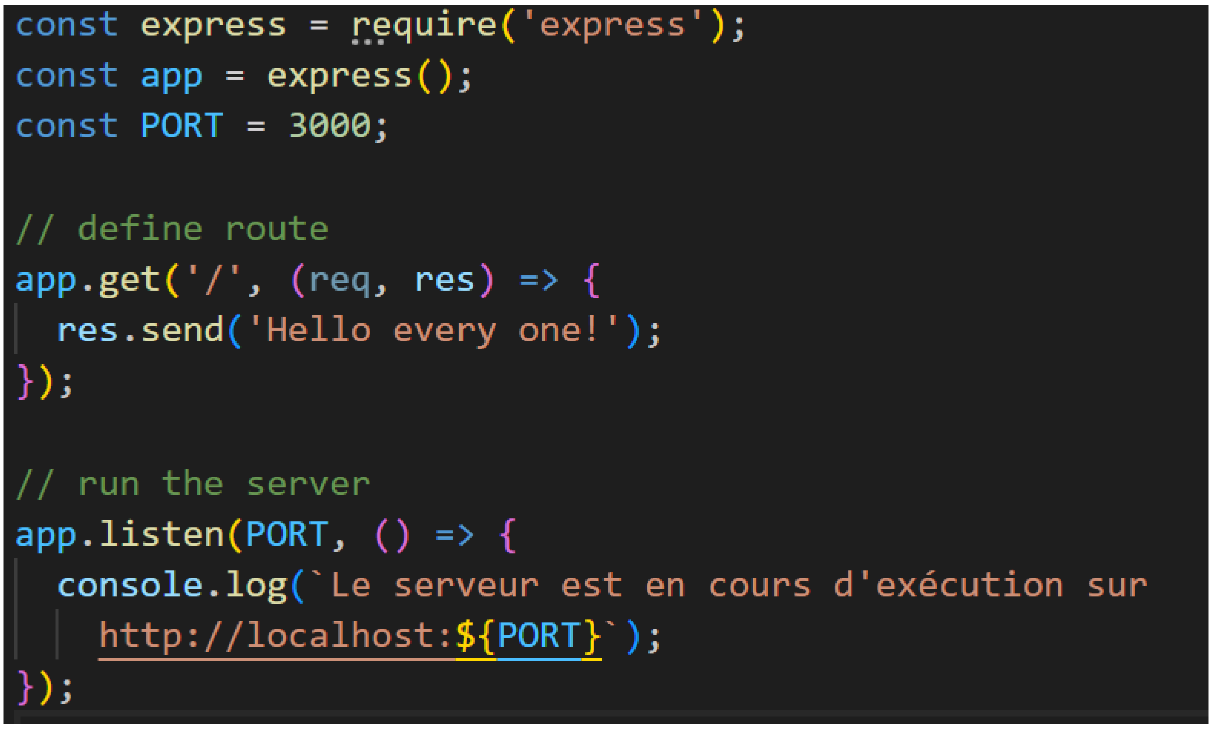
4- Set Up Express: run the server using app.listen

- Run the file "crud_app.js" using node crud_app.js



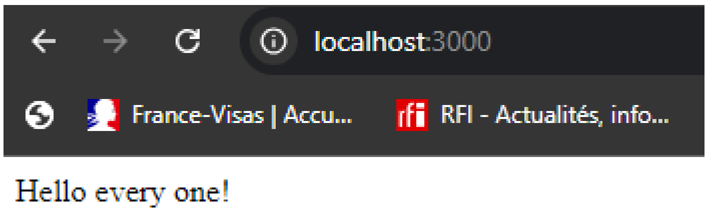
```
C:\Users\dell\OneDrive\Bureau\IID3_S1\JS_MOBILE\TP2\crud_application>node crud_app.js
Le serveur est en cours d'exécution sur http://localhost:3000
```

- We configure this file as below for start the server using crud_app.listen to define the port on which your server will listen.
- Add a route to test



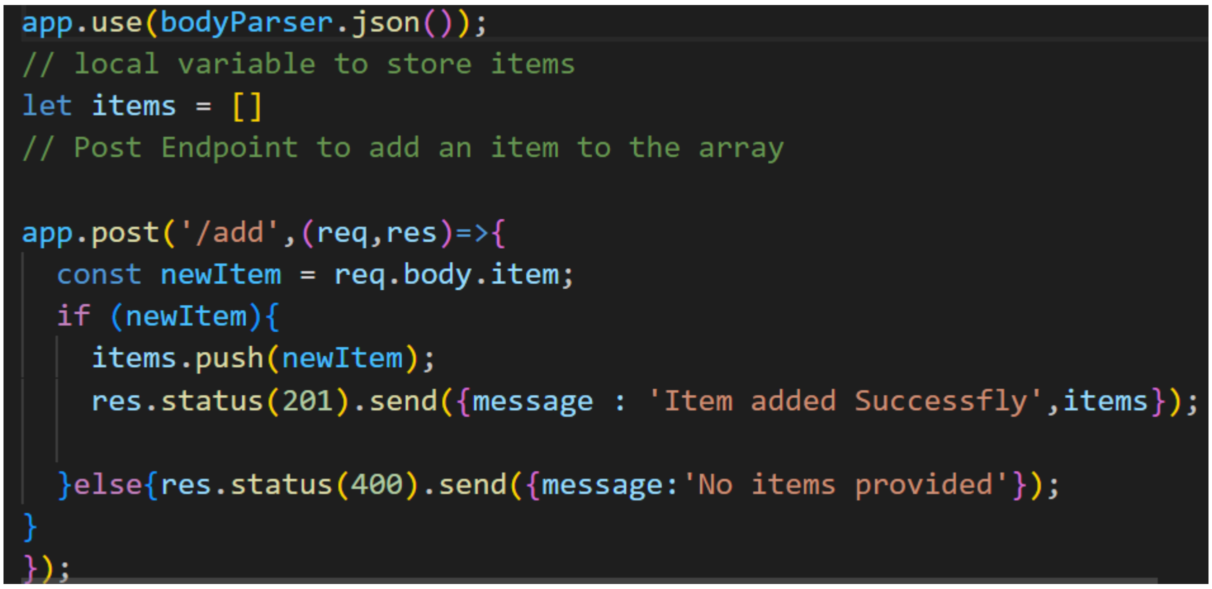
```
const express = require('express');
const app = express();
const PORT = 3000;

// define route
app.get('/', (req, res) => {
  res.send('Hello every one!');
});

// run the server
app.listen(PORT, () => {
  console.log(`Le serveur est en cours d'exécution sur
    http://localhost:${PORT}`);
});
```

Exécution :



Hello every one!

5- Create a POST Endpoint: This will allow us to add items to a local variable.

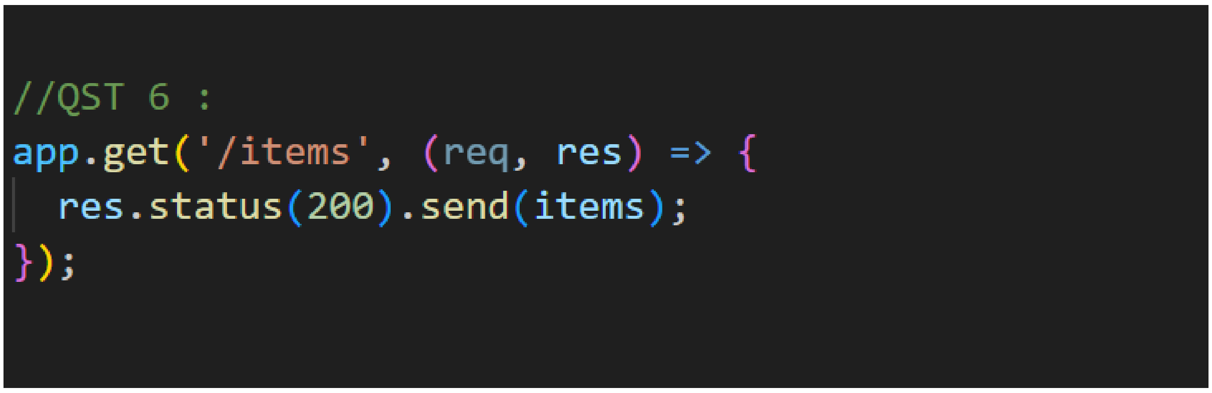


```
app.use(bodyParser.json());
// local variable to store items
let items = []
// Post Endpoint to add an item to the array

app.post('/add',(req,res)=>{
  const newItem = req.body.item;
  if (newItem){
    items.push(newItem);
    res.status(201).send({message : 'Item added Successfly',items});

  }else{res.status(400).send({message:'No items provided'});
}
});
```

- Use “bodyParser” to enable our Express application to handle JSON data sent in requests. This is especially important for POST, PUT, and DELETE endpoints where we need to retrieve data from the request body (like a new item to add or an item to update)

- The last part of code defines a POST endpoint /add to add a new item to the items array.Firstly it retrieves the new item from the request body using req.body.item , if newItem exists, it is added to the items array.

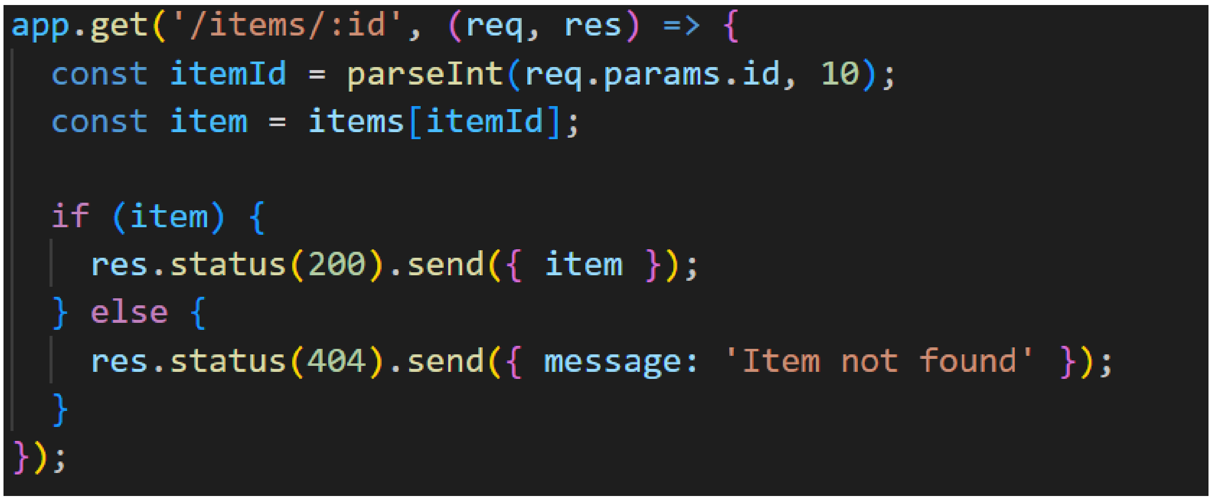
6- Create a GET Endpoint: This will allow us to retrieve all items.



```
//QST 6 :
app.get('/items', (req, res) => {
  res.status(200).send(items);
});
```

- This code defines a GET endpoint /items that allows clients to retrieve the current list of items stored in the items array

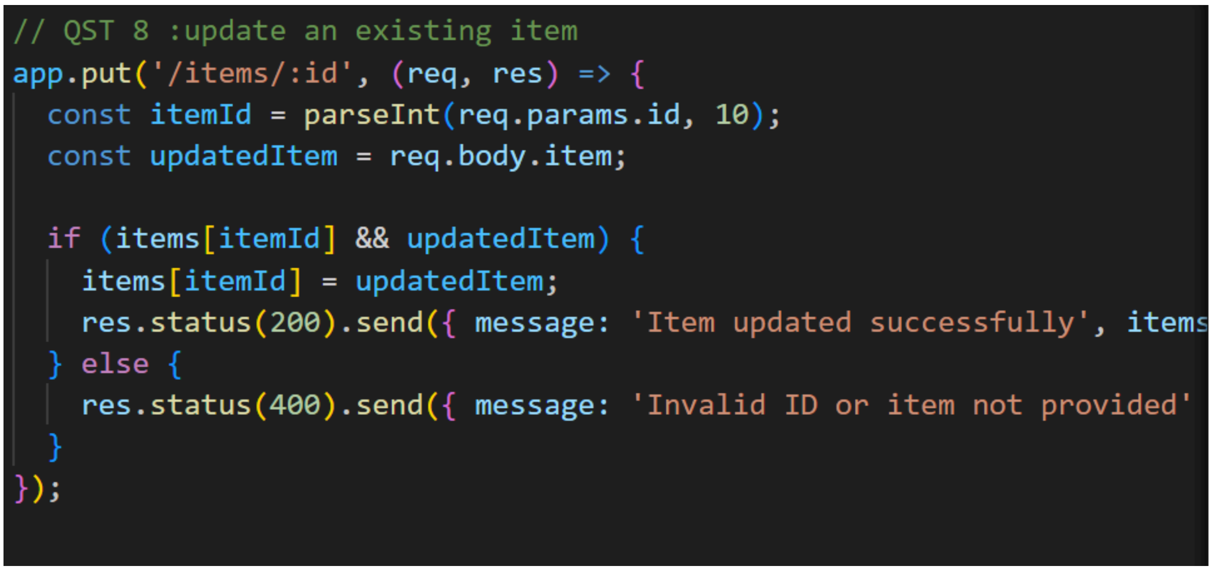
7-. Create a GET Endpoint by ID: This will allow us to get a specific item.



```
app.get('/items/:id', (req, res) => {
  const itemId = parseInt(req.params.id, 10);
  const item = items[itemId];

  if (item) {
    res.status(200).send({ item });
  } else {
    res.status(404).send({ message: 'Item not found' });
  }
});
```

- This code defines a GET endpoint: /items/:id that allows clients to retrieve a specific item from the items array using its ID.
- The :id in the route signifies a dynamic parameter, allowing to pass an item index in the URL
- const itemId = parseInt(req.params.id, 10) : retrieves the ID from the request parameters and converts it to an integer.
- const item = items[itemId]; looks up the item in the items array using the extracted ID.

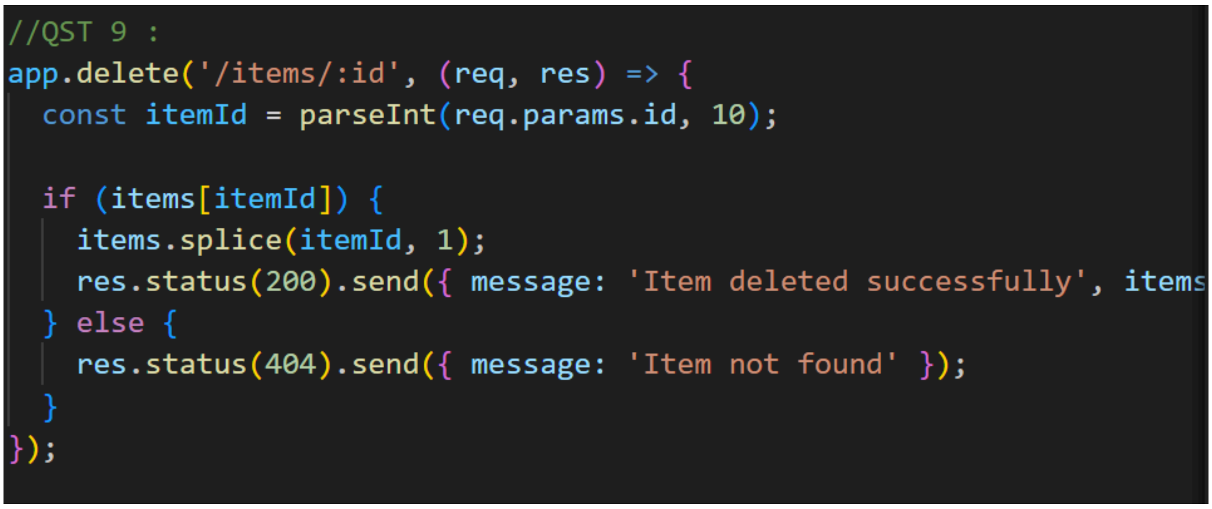
8- Create a PUT Endpoint: This will allow us to update an existing item.



```
// QST 8 :update an existing item
app.put('/items/:id', (req, res) => {
  const itemId = parseInt(req.params.id, 10);
  const updatedItem = req.body.item;

  if (items[itemId] && updatedItem) {
    items[itemId] = updatedItem;
    res.status(200).send({ message: 'Item updated successfully', items
  } else {
    res.status(400).send({ message: 'Invalid ID or item not provided'
  }
});
```

- This code defines a PUT endpoint /items/:id that allows clients to update a specific item in the items array.
- The :id in the route signifies a dynamic parameter, allowing to pass an item index in the URL
- If the item at items[itemId] exists and updatedItem is provided, it updates the item in the array with the new value.

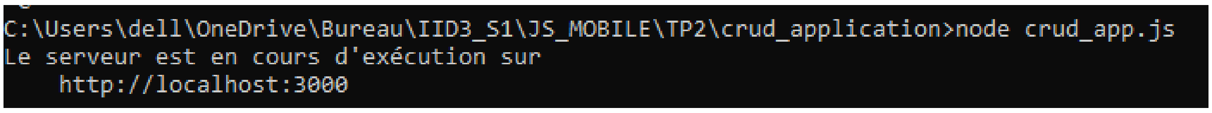
9- Create a DELETE Endpoint: This will allow us to delete an item.



```
//QST 9 :
app.delete('/items/:id', (req, res) => {
  const itemId = parseInt(req.params.id, 10);

  if (items[itemId]) {
    items.splice(itemId, 1);
    res.status(200).send({ message: 'Item deleted successfully', items
  } else {
    res.status(404).send({ message: 'Item not found' });
  }
});
```

- The app.delete() function defines a DELETE endpoint that allows clients to remove a specific item from the items array.
- If the item at the specified index exists (if (items[itemId])), it removes the item using items.splice(itemId, 1), which removes one item at that index.

10- Start the Server:



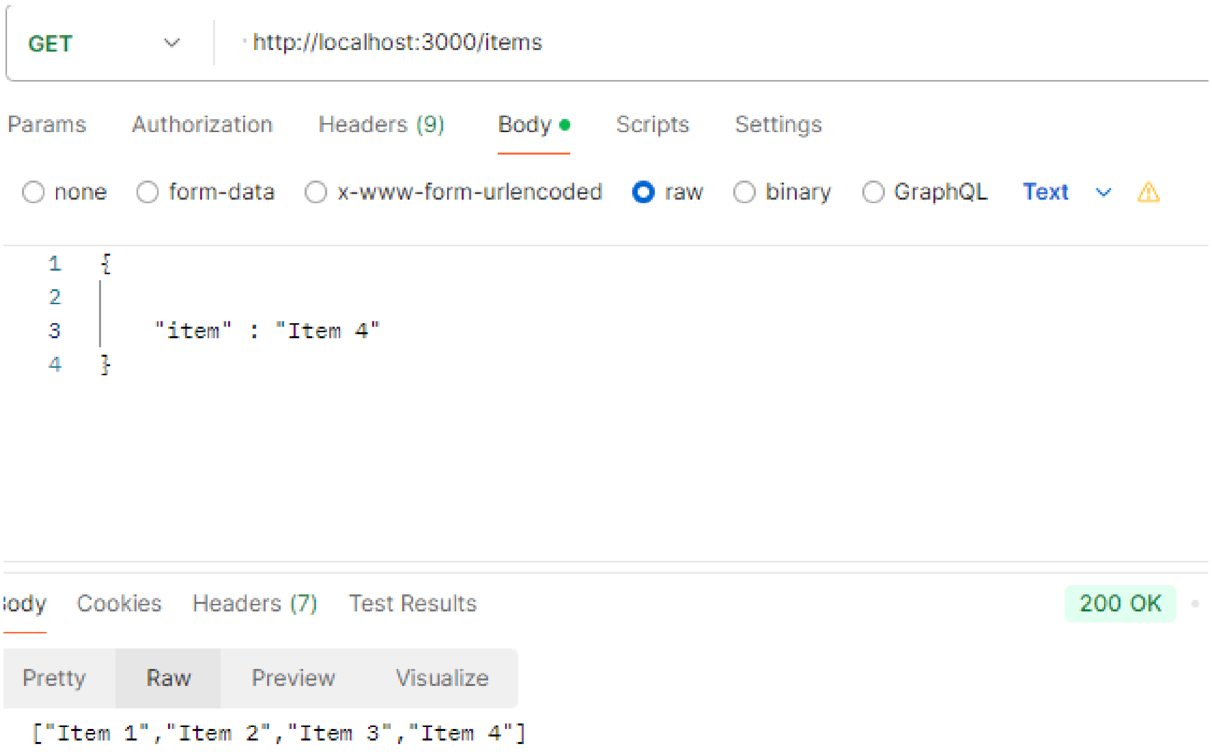
```
C:\Users\dell\OneDrive\Bureau\IID3_S1\JS_MOBILE\TP2\crud_application>node crud_app.js
Le serveur est en cours d'exécution sur
    http://localhost:3000
```

11- Test the Endpoints using Postman :

- **Test the POST Endpoint**
  Using the URL http://localhost:3000/add with POST in the example below we add 4 items

- **Test the GET Endpoint**

  Using th URL http://localhost:3000/items with GET we list all items



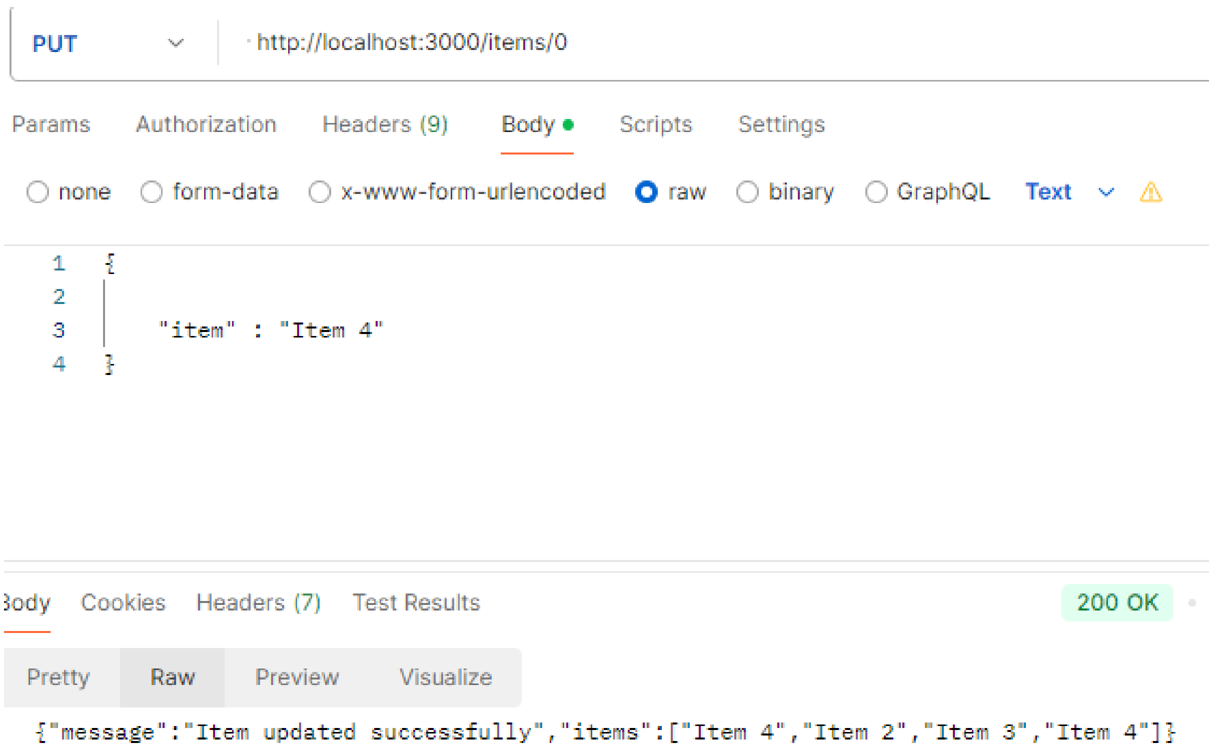
GET http://localhost:3000/items

```
{

    "item" : "Item 4"
}
```

```
["Item 1","Item 2","Item 3","Item 4"]
```

- **Test PUT Endpoint**

  Using the URL http://localhost:3000/items/0 with PUT in the example below we update the item of id=0 bye the "Item 4"



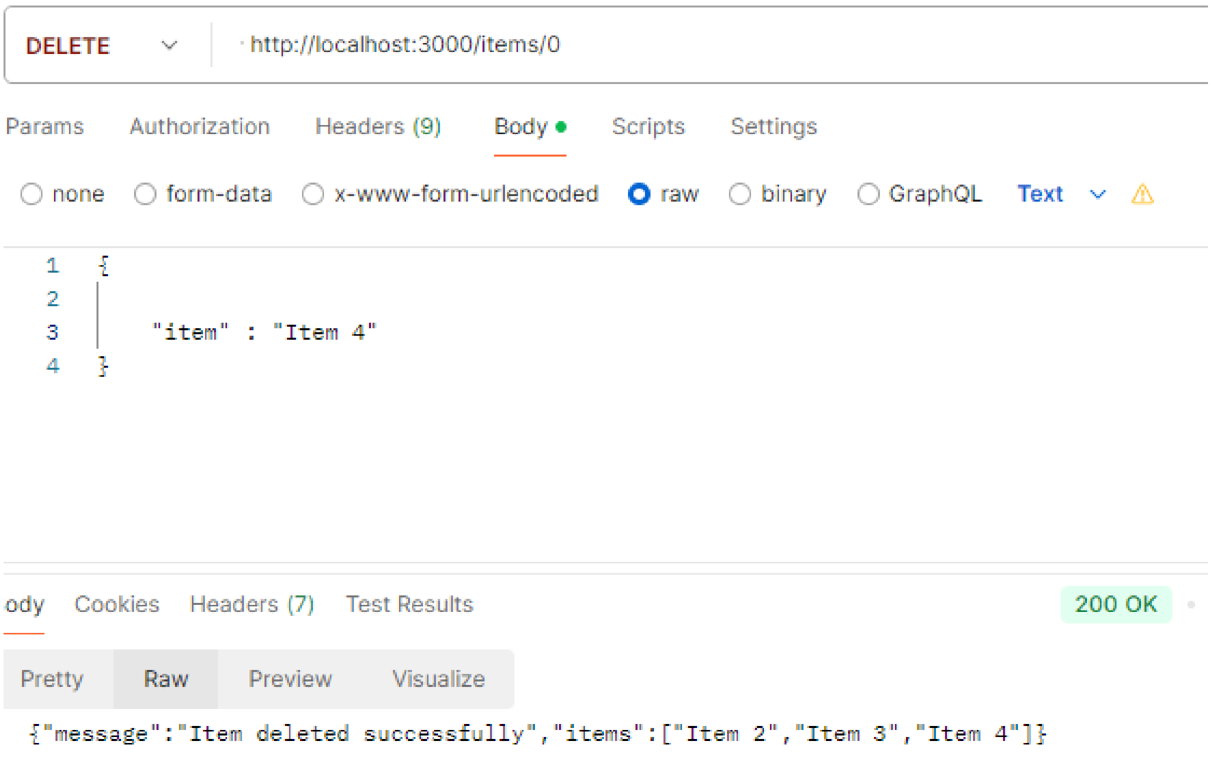
http://localhost:3000/items/0

```
{

    "item" : "Item 4"
}
```

```
{"message":"Item updated successfully","items":["Item 4","Item 2","Item 3","Item 4"]}
```

- **Test Delete Endpoint :**

  Using the URL http://localhost:3000/items/0 with Delete in the example below we delete the item of id=0.



```
{

    "item" : "Item 4"
}
```

```
{"message":"Item deleted successfully","items":["Item 2","Item 3","Item 4"]}
```