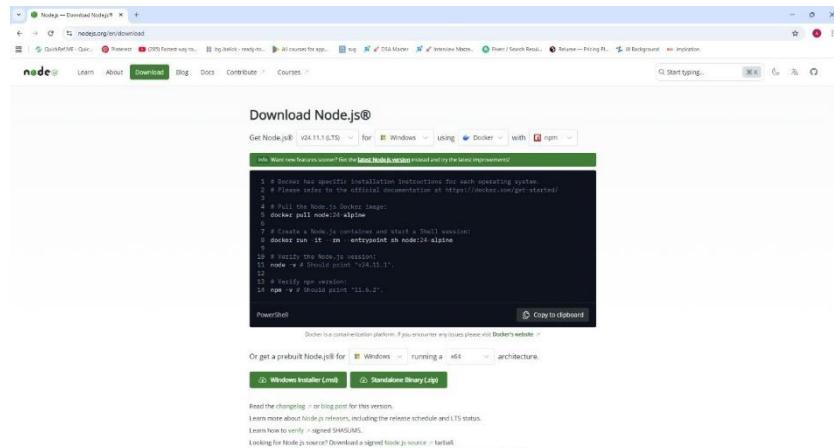


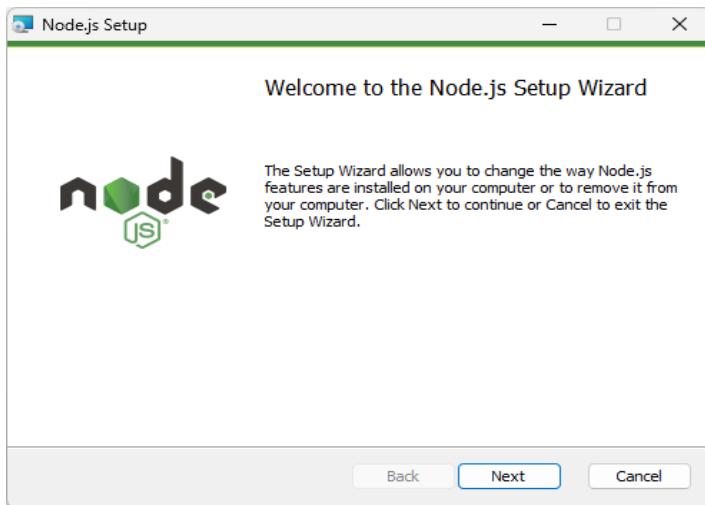
Practical No 1

Aim : Installation and initialization of Node.js

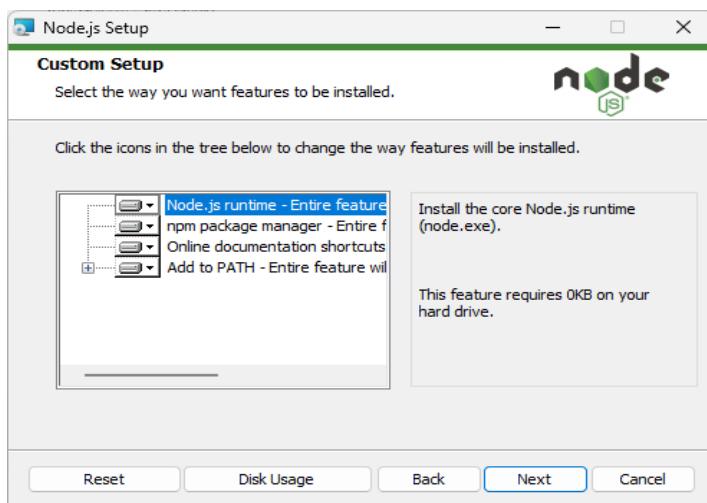
Step 1



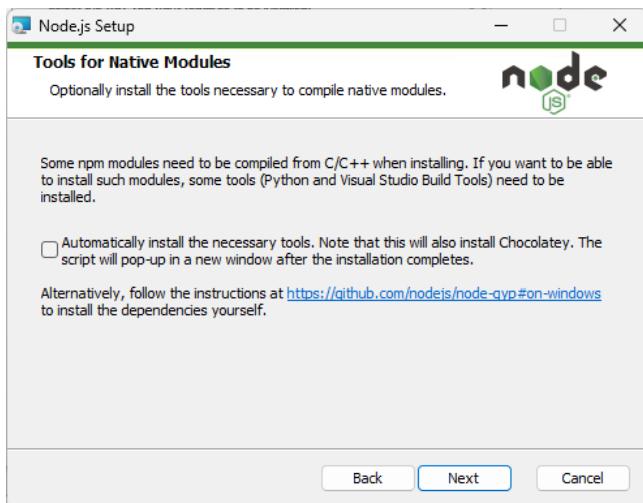
Step 2



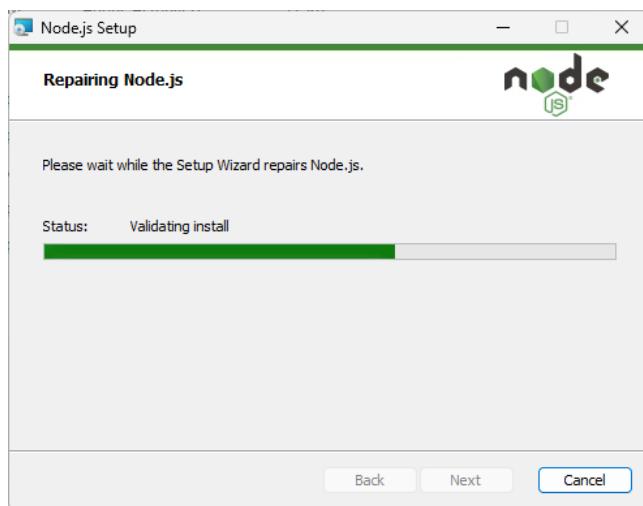
Step 3



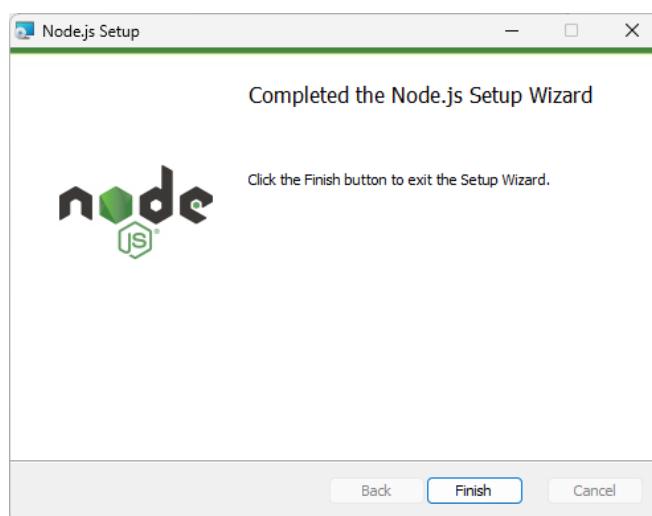
Step 4



Step 5



Step 6



Step 7

```
C:\Users\Administrator>node --version  
v25.0.0
```

Practical No: 2

AIM: Write a Program to demonstrate creation and understanding of package.json and usage of npm

```
$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (js(node)) crud-operation
version: (1.0.0)
description: project
entry point: (first.js)
test command:
git repository:
keywords:
author: Habiba
license: (ISC)
type: (commonjs)
About to write to C:\Clg\JS(Node)\package.json:

{
  "name": "crud-operation",
  "version": "1.0.0",
  "description": "project",
  "main": "first.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Habiba",
  "license": "ISC",
  "type": "commonjs"
}
```

Package.json

```
{
  "name": "crud-operation",
  "version": "1.0.0",
  "description": "project",
  "license": "ISC",
  "author": "Habiba",
  "type": "commonjs",
  "main": "first.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  }
}
```

Practical No: 3

AIM: Write a program to demonstrate creating module and using module .export or require

Input:

1.index.js

```
import add from "./math.js";  
  
console.log(add(3, 4))
```

2.math.js

```
function add(a,b){  
    console.log("addition")  
    return a + b  
}  
export default add;
```

Output:

```
addition
```

```
7
```

```
PS C:\MeanStack>
```

Practical No: 4

AIM: Write a program to demonstrate the node.js in Event Loop

1: Set Timeout

Input:

```
1  console.log("A");
2  setTimeout(() => {
3    |  console.log("Habiba (Run after 2 seconds)");
4  },2000);
5
6  console.log("B");
```

Output:

```
● A
  B
  Habiba (Run after 2 seconds)
```

2: Set Immediate

Input:

```
1  console.log("A");
2
3  setImmediate(() => {
4    |  console.log("Habiba");
5  });
6
7  console.log("B");
```

Output:

```
PS C:\Users\Admin\Documents\NodeJS>
● A
  B
  Habiba
```

3: Set Interval

Input:

```
1  console.log("A");
2  setInterval(() => {
3    |  console.log("Habiba (Run after every 2 seconds)");
4  },2000);
5
6  console.log("B");
```

Output:

```
A  
B  
Habiba (Run after every 2 seconds)  
Habiba (Run after every 2 seconds)  
Habiba (Run after every 2 seconds)
```

4 Process.Next tick

Input:

```
1  console.log("A");  
2  process.nextTick(() => {  
3      console.log("Habiba (Next tick)");  
4  });  
5  console.log("B");
```

Output:

```
PS C:\Users\Admin\Documents\  
▶ A  
B  
Habiba (Next tick)
```

5 Asynchronous Function

Input:

```
1  async function name(params) {  
2      console.log(params);  
3  }  
4  await name("Habiba")
```

Output:

```
PS C:\Users\Admin\Documents\  
▶ Habiba  
▶ PS C:\Users\Admin\Documents\
```

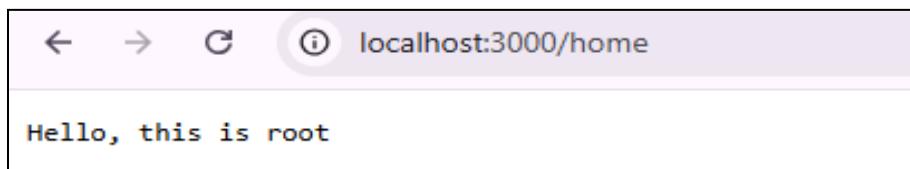
Practical No: 5

AIM: Write a program to demonstrate handling routes and requests in a node.js serve
Input:

```
const http =require("http");
const Server = http.createServer((req, res) => {
    if (req.url === "/home") {
        res.write("Hello, this is root");
        res.end();
    }
    else if (req.url === "/about") {
        res.write("Hello, this is about page");
        res.end();
    }
    else {
        res.write("Hello, this is Else");
        res.end();
    }
});
Server.listen(3000, () => console.log('Server is running at 3000'));
```

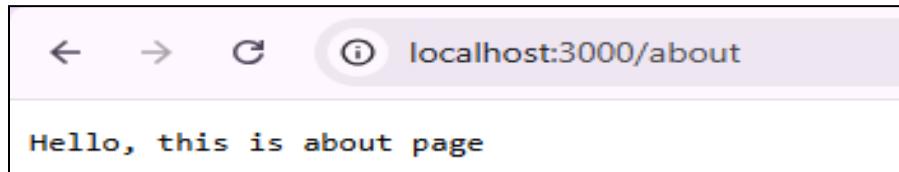
Output:

```
PS C:\Users\Admin\Documents>
> Server is running at 3000
█
```



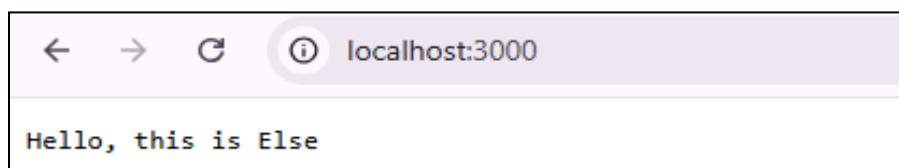
localhost:3000/home

Hello, this is root



localhost:3000/about

Hello, this is about page



localhost:3000

Hello, this is Else

Practical No 6

Aim : Write A Program To Demonstrate Handling Routes And Request In A Node.js Server

```
const http = require("http")
const { addNumber } = require("./add")

const app = http.createServer((req, res) => {
  const url = req.url

  if ("/" === url) {
    res.write("This Is Home Page")
    res.end()
  }

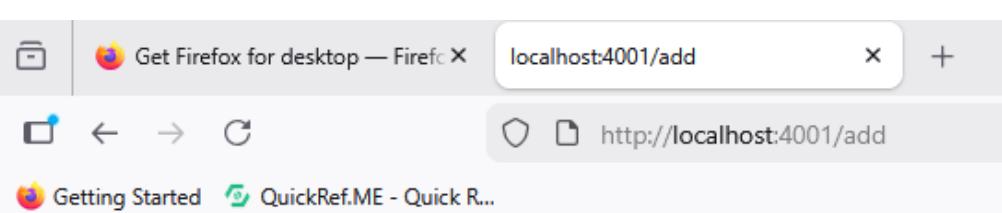
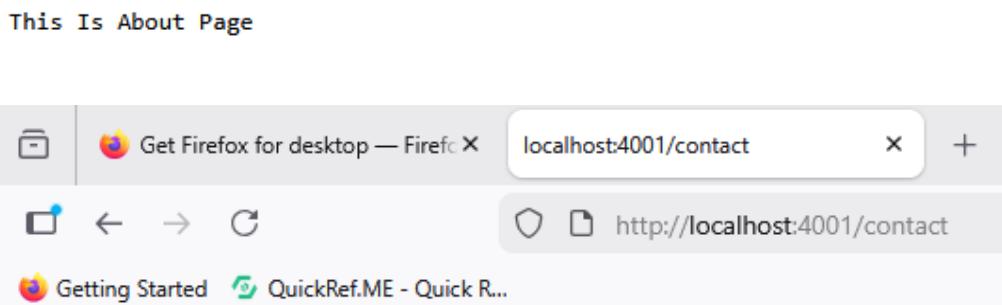
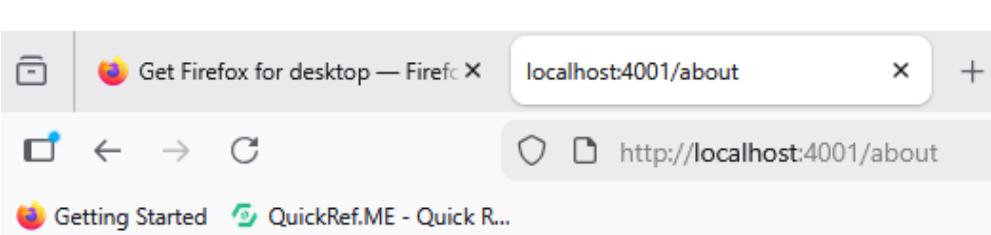
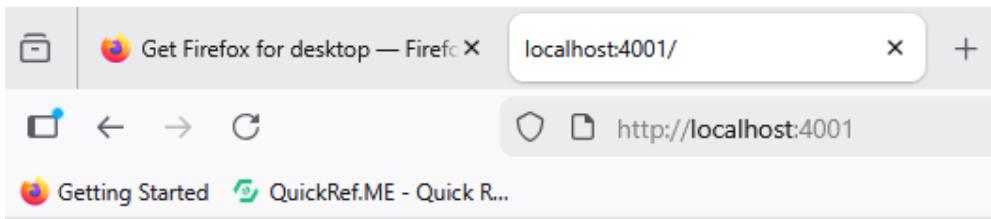
  else if ("/about" === url) {
    res.write("This Is About Page")
    res.end()
  }

  else if ("/add" === url) {
    res.write("This Is About Page")
    res.write("\n")
    res.write(`${addNumber(12, 24)})`)
    res.end()
  }

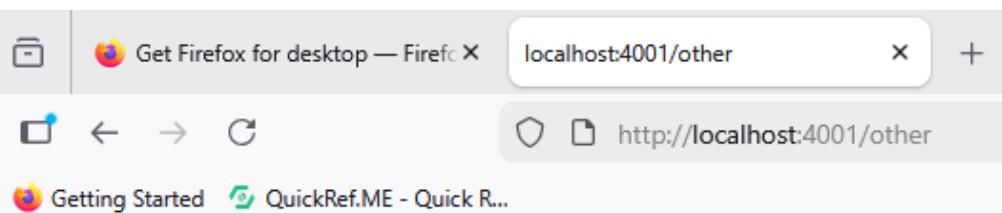
  else if ("/contact" === url) {
    res.write("This Is Contact Page")
    res.end()
  }

  else {
    res.write("Page Not Found")
    res.end()
  }
})

app.listen(4001, () => {
  console.log("Server Started At Port ", 4001)
})
```



This Is Add Page
36



Practical No 7

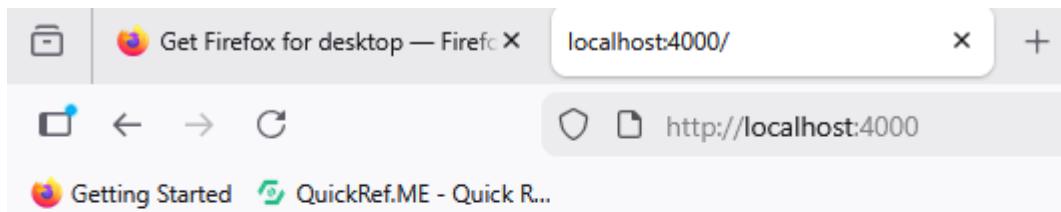
Aim : Write A Program To Demonstrate Introduction And Creation Of Express.js Application.

```
import express from "express"

const port = 4000
const app = express()

app.get("/", (req, res) => {
  res.send("Hello World!, From Server.")
})

app.listen(port, () => {
  console.log(`Server Started On : localhost:${port}`)
})
```



Hello World!, From Server.

Practical No 8

Aim : Write A Program To Demonstrate The Use Of Middleware In Express.js.

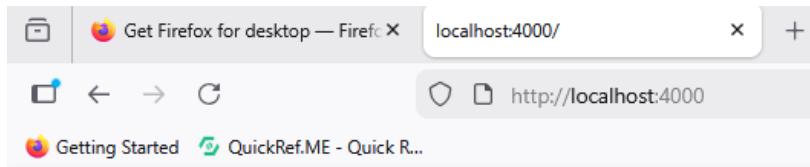
```
import express from "express"
import { logger } from "./middleware/logger.js";

const port = 4000
const app = express()

// Middleware
app.use(express.json())
app.use(logger)

app.get("/", (req, res) => {
  res.send("Hello World!, From Server.")
})

app.listen(port, () => {
  console.log(`Server Started On : localhost:${port}`)
})
```



Hello World!, From Server.

```
_workers: [],
_unref: false,
_listeningId: 2,
allowHalfOpen: true,
pauseOnConnect: false,
noDelay: true,
keepAlive: false,
keepAliveInitialDelay: 0,
highWaterMark: 16384,
httpAllowHalfOpen: false,
timeout: 0,
maxHeadersCount: null,
maxRequestsPerSocket: 0,
_connectionKey: '6:::4000',
Symbol(IncomingMessage): [Function: IncomingMessage],
Symbol(ServerResponse): [Function: ServerResponse],
Symbol(shapeMode): false,
Symbol(kCapture): false,
Symbol(async_id_symbol): 8,
Symbol(kUniqueHeaders): null,
Symbol(http.server.connections): ConnectionsList {},
Symbol(http.server.connectionsCheckingInterval): Timeout {
  _idleTimeout: 30000,
  _idlePrev: [Timerslist],
  _idleNext: [Timerslist],
  _idleStart: 172,
  _onTimeout: [Function: bound checkConnections],
  _timerArgs: undefined,
  _repeat: 30000,
  _destroyed: false,
  Symbol(refed): false,
  Symbol(kHasPrimitive): false,
  Symbol(asyncId): 10,
  Symbol(triggerId): 9,
  Symbol(kAsyncContextFrame): undefined
},
},
```

Practical No 9

Aim : Write A Program To Demonstrate Routing In Express.js.

```
import express from "express"

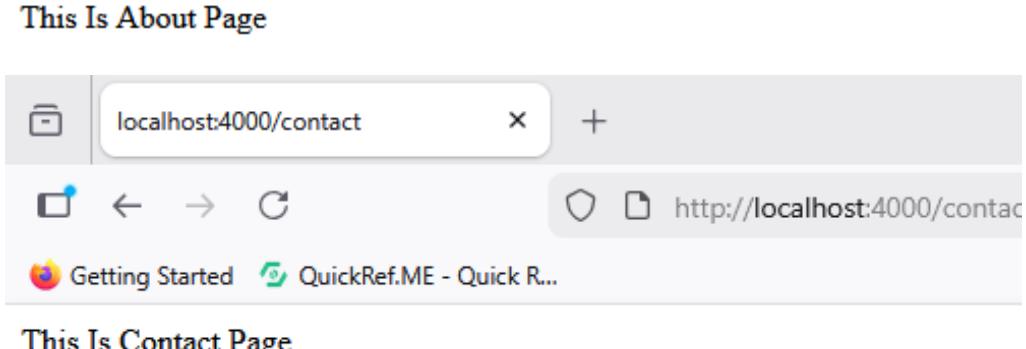
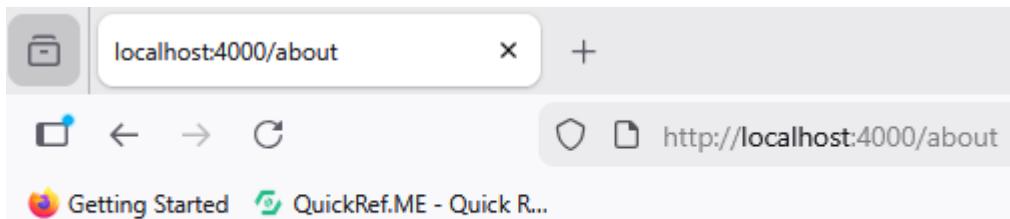
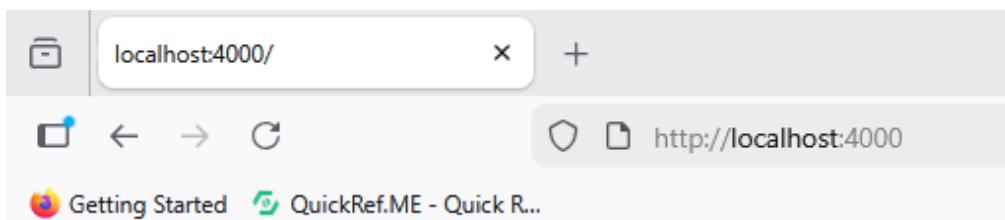
const port = 4000
const app = express()

app.get("/", (req, res) => {
  res.send("Hello World!, From Server.")
})

app.get("/about", (req, res) => {
  res.send("This Is About Page")
})

app.get("/contact", (req, res) => {
  res.send("This Is Contact Page")
})

app.listen(port, () => {
  console.log(`Server Started On : localhost:${port}`)
})
```



Practical No 10

**Aim : Write A Program To Demonstrate RESTFul API In Express.js
(GET, POST, PUT, DELETE)**

```
const express = require("express");
const app = express()
app.use(express.json());
let students = [
  { id: 1, name: "Habiba" },
  { id: 2, name: "Mahek" },
  {id : 3, name: "Zaberiya" }
];
app.get("/students", (req, res) => {
  res.send(students);
});
app.get("/students/:id", (req, res) => {
  const id = Number(req.params.id);
  const student = students.find(s => s.id === id);
  if (!student) {
    return res.status(404).send("Student not found");
  }
  res.send(student);
});
app.post("/students", (req, res) => {
  const newStudent = {
```

```

    id: students.length + 1,
    name: req.body.name
};

students.push(newStudent);

res.status(201).send(newStudent);

});

app.put("/students/:id", (req, res) => {

const id = Number(req.params.id);

const student = students.find(s => s.id === id);

if (!student) {

    return res.status(404).send("Student not found");

}

student.name = req.body.name;

res.send(student);

});

app.delete("/students/:id", (req, res) => {

const id = Number(req.params.id);

students = students.filter(s => s.id !== id);

res.send("Student deleted");

});

app.listen(4000, () => {

    console.log("Server is running at http://localhost:4000");

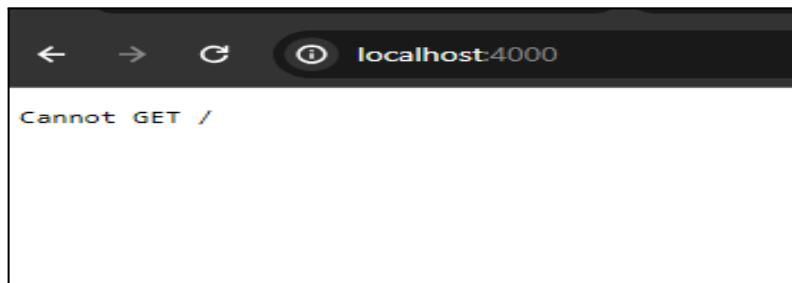
});

```

```

PS C:\Users\Admin\Documents\SEM-1V\MeanStack\Module 1> node "c:\Users\Admin\Documents\SEM-1V\MeanStack\Module 1\Express\data.js"
Server is running at http://localhost:4000

```



GET

The screenshot shows the POSTMAN interface with a "GET" request selected. The URL is set to "http://localhost:4000/students".

The screenshot shows the "Body" tab of the POSTMAN interface displaying the JSON response. The response is a list of three student objects:

```
1 [  
2   {  
3     "id": 1,  
4     "name": "Habiba"  
5   },  
6   {  
7     "id": 2,  
8     "name": "Mahek"  
9   },  
10  {  
11    "id": 3,  
12    "name": "Zaberiya"  
13  }  
14 ]
```

POST

The screenshot shows the POSTMAN interface with a "POST" request selected. The URL is "http://localhost:4000/students". The "Body" tab is active, showing the JSON payload:

```
1 {  
2   "id": 4,  
3   "name": "Muskan"  
4 }
```

The screenshot shows the "Body" tab of the POSTMAN interface displaying the JSON response. The response is a single student object with id 4 and name Muskan.

```
1 {  
2   "id": 4,  
3   "name": "Muskan"  
4 }
```

PUT

PUT http://localhost:4000/students/3

Params Authorization Headers (8) Body Pre-request Script

Body Type: raw (JSON)

```
1 {  
2   "name": "zabbi"  
3 }
```

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```
1 {  
2   "id": 3,  
3   "name": "zabbi"  
4 }
```

DELETE

DELETE http://localhost:4000/students/4

Params Authorization Headers (8) Body Pre-request Script

Body Type: raw (JSON)

```
1 {  
2   "name": "Muskan"  
3 }
```

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize

```
1 Student deleted
```