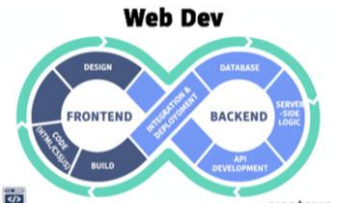
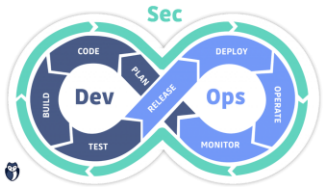



# Day 45

# "Web Development + Security"

## Response, Request and Routers in Express:

**What Are Requests and Responses?**

Every time a client (like your browser or frontend app) interacts with your Express server:

CLIENT → [HTTP REQUEST] → SERVER (Express)

SERVER → [HTTP RESPONSE] → CLIENT

**Request (req)**

It represents data coming into the server (from the client).

It can include:

- URL → /home, /about, etc.
- Method → GET, POST, PUT, DELETE
- Query parameters → ?id=10&name=Aditya
- Body data → JSON or form data in a POST request
- Headers, cookies, etc.

**Response (res)**

This is what the server sends back to the client — can be text, JSON, HTML, or status codes.

Common methods:

- res.send() → Send text, JSON, or HTML
- res.json() → Send JSON data
- res.status(code) → Set HTTP status code
- res.redirect() → Redirect to another URL

**Routers in Express (From Basics)**

When your app grows, you'll have many routes — /, /login, /products, etc. Putting them all in one file makes it messy.
Routers help organize routes into smaller, modular files.

**Summary**

| Concept | Description | Example |
|---|---|---|
| **req** | Request object → contains info sent by client | req.query, req.params, req.body |
| **res** | Response object → used to send data back | res.send("Hello"), res.json({...}) |
| **Router** | Helps organize routes into smaller files | app.use("/users", userRoutes) |

**What Are HTTP Request Methods?**

When a client (browser, frontend, or app) sends data to your Express.js server, it does so using different HTTP methods — each representing a different kind of operation.

Think of it like:

- GET = Ask for info
- POST = Send new info
- PUT = Update existing info
- DELETE = Remove info

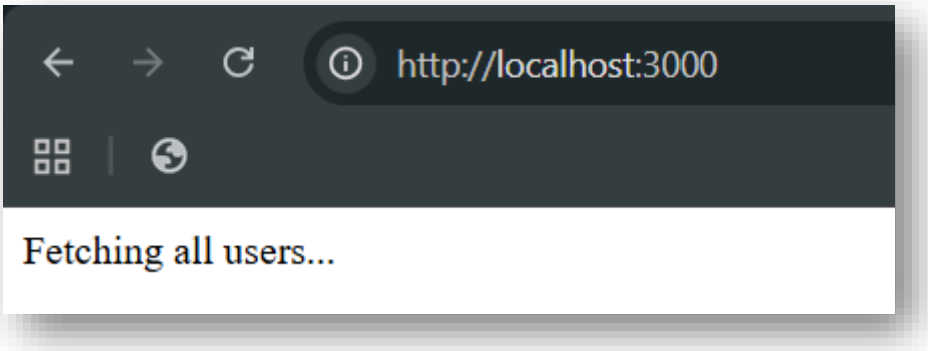
1) GET Request – can be checked by browser

**Purpose:** Used to retrieve data from the server. It does not modify anything — just reads data.

Example: of get request

```
JS main.js > ...
1   const express = require('express')
2   const app = express()
3   const port = 3000
4
5   app.get('/', (req, res) => {
6     res.send('Fetching all users...')
7   })
8
9   app.listen(port, () => {
10    console.log(`Example app listening on port ${port}`)
11  })
12
```

Output:

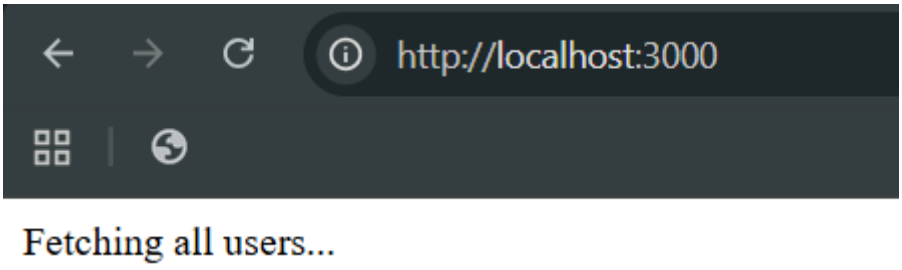


Now, printing something in the terminal we will use the console.log():

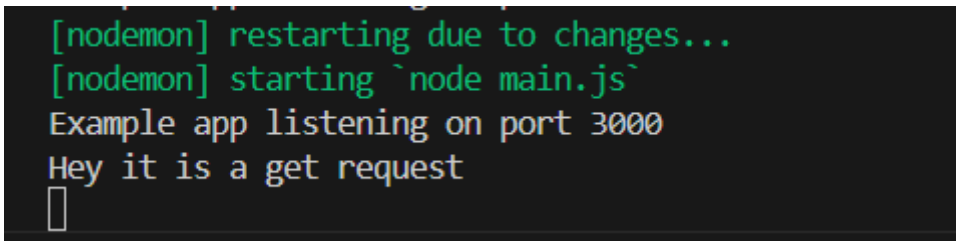
Code:

```
JS main.js > app.get('/') callback
1   const express = require('express')
2   const app = express()
3   const port = 3000
4
5   app.get('/', (req, res) => {
6       console.log("Hey it is a get request")
7       res.send('Fetching all users...')
8   })
9
10  app.listen(port, () => {
11      console.log(`Example app listening on port ${port}`)
12  })
```

Output: Browser



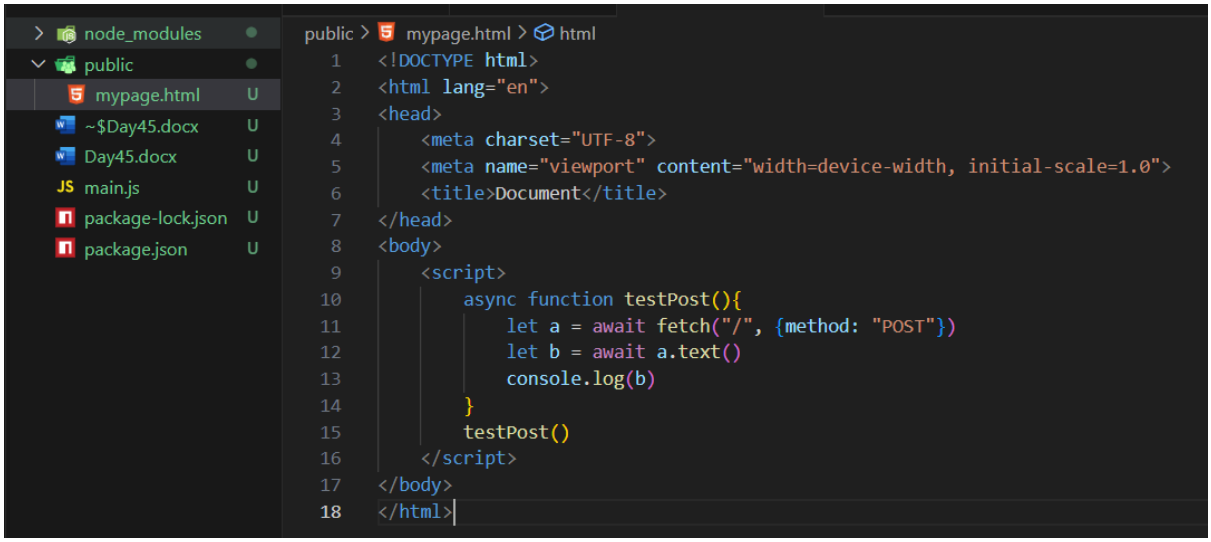

Output: terminal. Clearly, console.log prints on the terminal and res.send prints on the browser window.



2) POST Request – can be checked when by creating a public folder

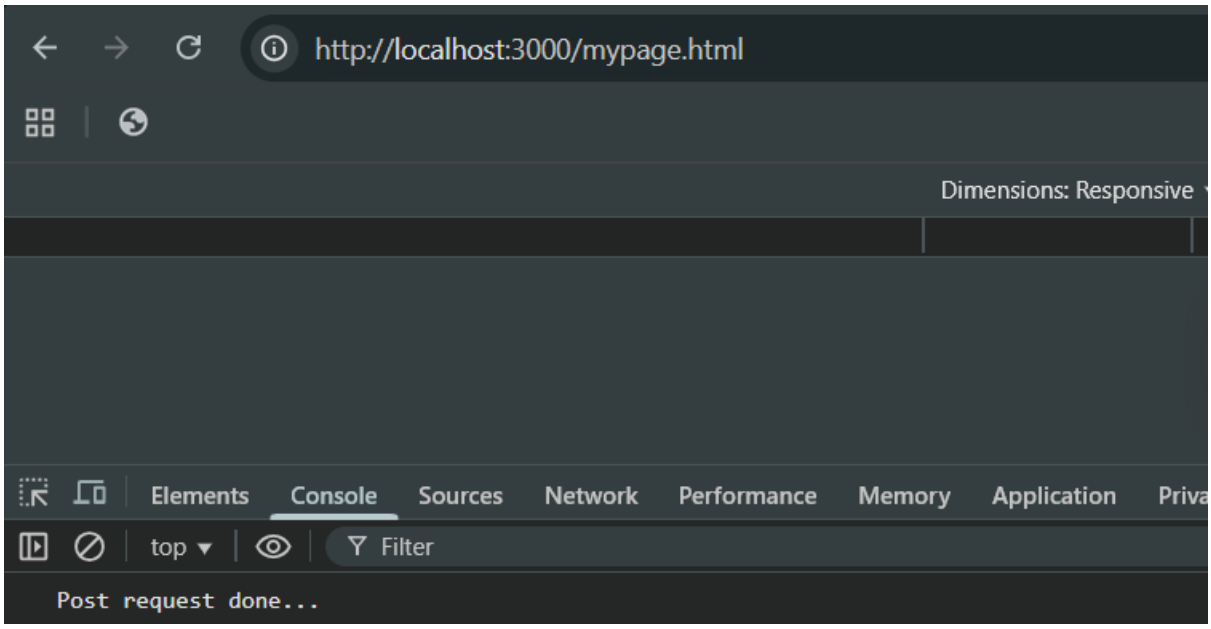
**Purpose:** Used to send or create new data on the server. Usually used with forms or JSON bodies.

Example: of post request. For which we will be creating a public folder and creating mypage.html inside it.

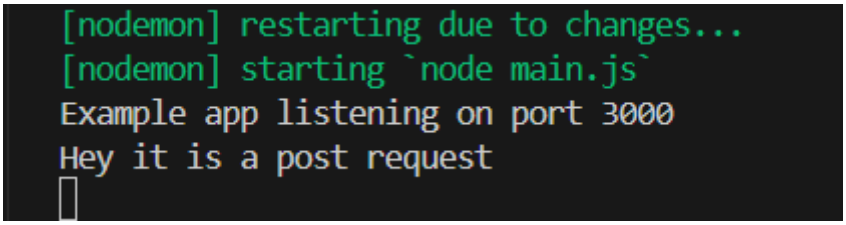



Also, we will be writing the post request code in the main.js:

Output: in console when we visit the mypage.html in the website via local host



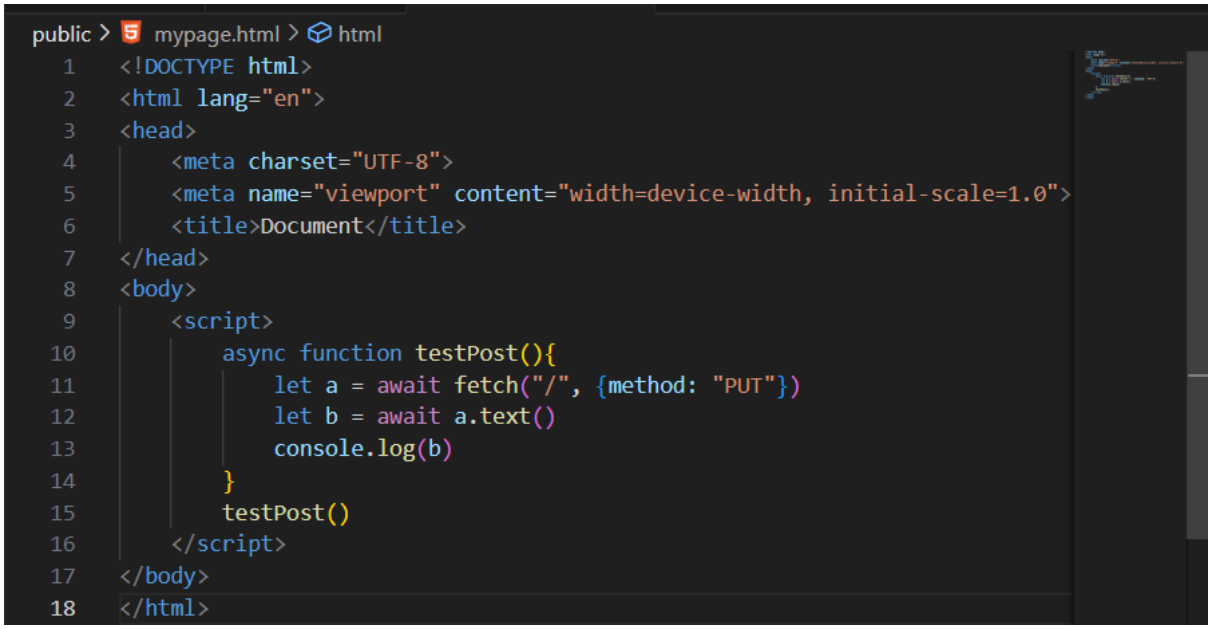

Terminal output:




3) **PUT Request – try to change the value we had given by post**

**Purpose:** Used to update or replace an existing resource completely.

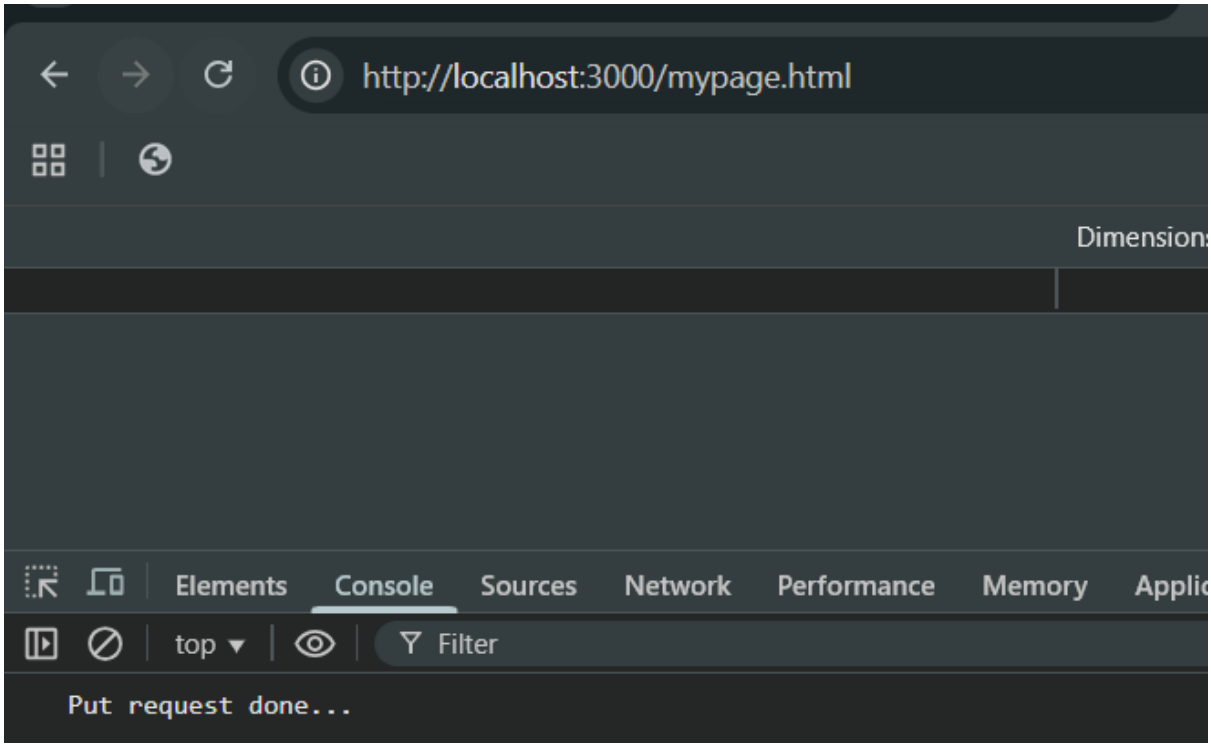
Mypage.html: we wrote “PUT” over the method name.



Main.js: we added put request

```
JS main.js > ...
1   const express = require('express')
2   const app = express()
3   const port = 3000
4
5   app.use(express.static("public"))
6   app.get('/', (req, res) => {
7       console.log("Hey it is a get request")
8       res.send('Fetching all users...')
9   })
10
11  app.post('/', (req, res) => {
12      console.log("Hey it is a post request")
13      res.send('Post request done...')
14  })
15
16  app.put('/', (req, res) => {
17      console.log("Hey it is a put request")
18      res.send('Put request done...')
19  })
20
21  app.listen(port, () => {
22      console.log(`Example app listening on port ${port}`)
23  })
```

Output: console



Output: terminal

```
[nodemon] restarting due to changes...
[nodemon] starting `node main.js`
Example app listening on port 3000
Hey it is a put request
```

Now, we can also make the chain of the request. It is basically the other way of writing the code we wrote till now:

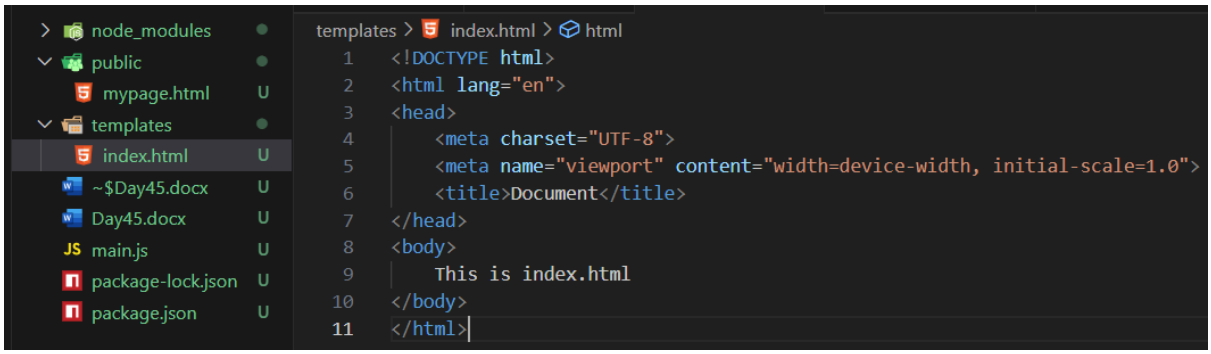
Main.js: we just remove “app” from each .post, .put and made a chain.

```
//Chaining of request
const express = require('express')
const app = express()
const port = 3000

app.use(express.static("public"))
app.get('/', (req, res) => {
    console.log("Hey it is a get request")
    res.send('Fetching all users...')
}).post('/', (req, res) => {
    console.log("Hey it is a post request")
    res.send('Post request done...')
}).put('/', (req, res) => {
    console.log("Hey it is a put request")
    res.send('Put request done...')
})

app.listen(port, () => {
    console.log(`Example app listening on port ${port}`)
})
```

Now, suppose **we want to serve the HTML**:

Then to do so, we should first create a template folder, and inside it we will create the index.html.

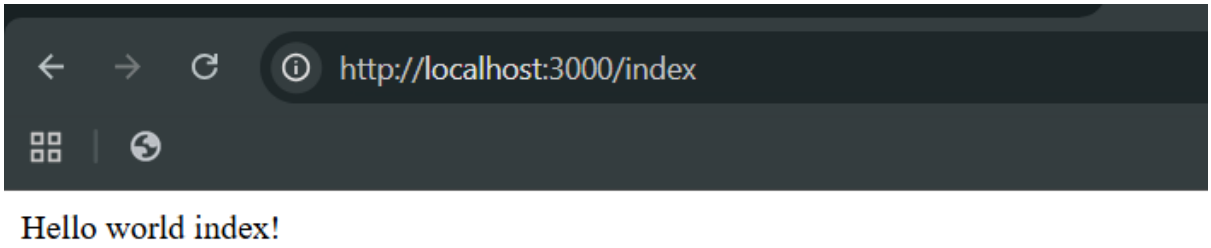
Index.html:



Main.js:

```
JS main.js > ...
47  //Serving the index.html
48
49  const express = require('express')
50  const app = express()
51  const port = 3000
52
53  app.use(express.static("public"))
54  app.get('/', (req, res) => {
55      console.log("Hey it is a get request")
56      res.send('Fetching all users...')
57  })
58
59  app.post('/', (req, res) => {
60      console.log("Hey it is a post request")
61      res.send('Post request done...')
62  })
63
64  app.put('/', (req, res) => {
65      console.log("Hey it is a put request")
66      res.send('Put request done...')
67  })
68
69  //to get the index.html
70  app.get('/index', (req, res) => {
71      console.log("Hey its index")
72      res.send('Hello world index!')
73  })
74  app.listen(port, () => {
75      console.log(`Example app listening on port ${port}`)
76  })
```

Output: at browser when we visited index




Output: console

```
[nodemon] restarting due to changes...
[nodemon] starting `node main.js`
Example app listening on port 3000
Hey its index
```

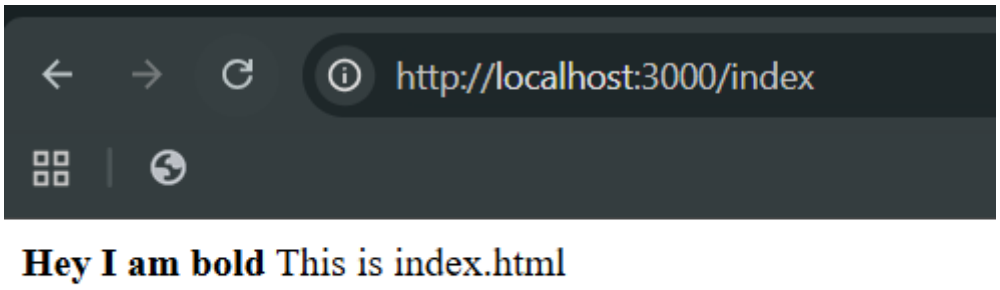
Now, suppose we want to render the HTML properly, then we will use .sendFile as shown below:

Index.html:

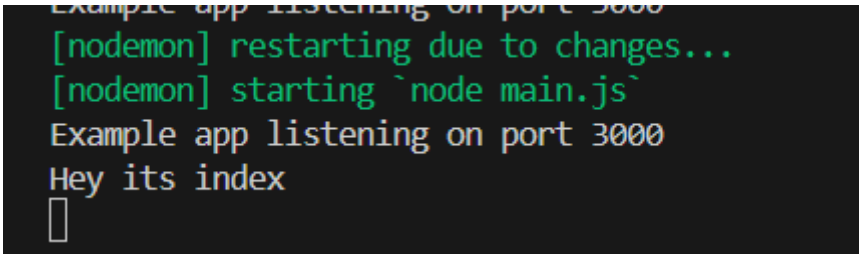

Main.js:

```
47  //Serving the index.html
48
49  const express = require('express')
50  const app = express()
51  const port = 3000
52
53  app.use(express.static("public"))
54  app.get('/', (req, res) => {
55      console.log("Hey it is a get request")
56      res.send('Fetching all users...')
57  })
58
59  app.post('/', (req, res) => {
60      console.log("Hey it is a post request")
61      res.send('Post request done...')
62  })
63
64  app.put('/', (req, res) => {
65      console.log("Hey it is a put request")
66      res.send('Put request done...')
67  })
68
69  //to get the index.html properly rendered
70  app.get('/index', (req, res) => {
71      console.log("Hey its index")
72      res.sendFile('templates/index.html', {root: __dirname})
73  })
74  app.listen(port, () => {
75      console.log(`Example app listening on port ${port}`)
76  })
```

Output: browser




Output: terminal




Also,

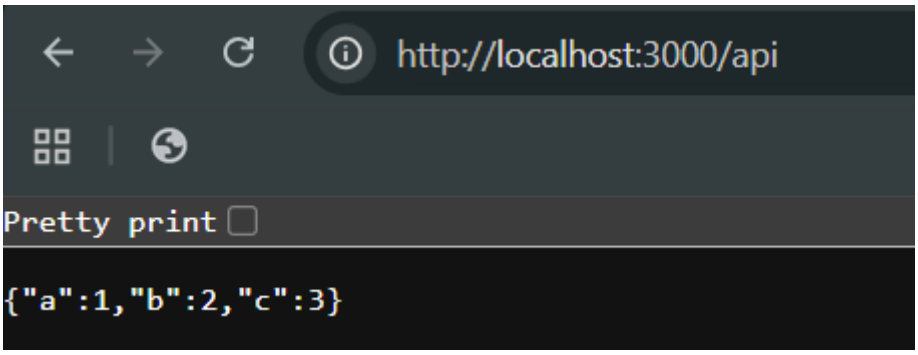
| Method | Description |
|---|---|
| res.download() | Prompt a file to be downloaded. |
| res.end() | End the response process. |
| res.json() | Send a JSON response. |
| res.jsonp() | Send a JSON response with JSONP support. |
| res.redirect() | Redirect a request. |
| res.render() | Render a view template. |
| res.send() | Send a response of various types. |
| res.sendFile() | Send a file as an octet stream. |
| res.sendStatus() | Set the response status code and send its string representation as the response body. |

Now, suppose we want that “/api” should return the JSON.

Main.js:

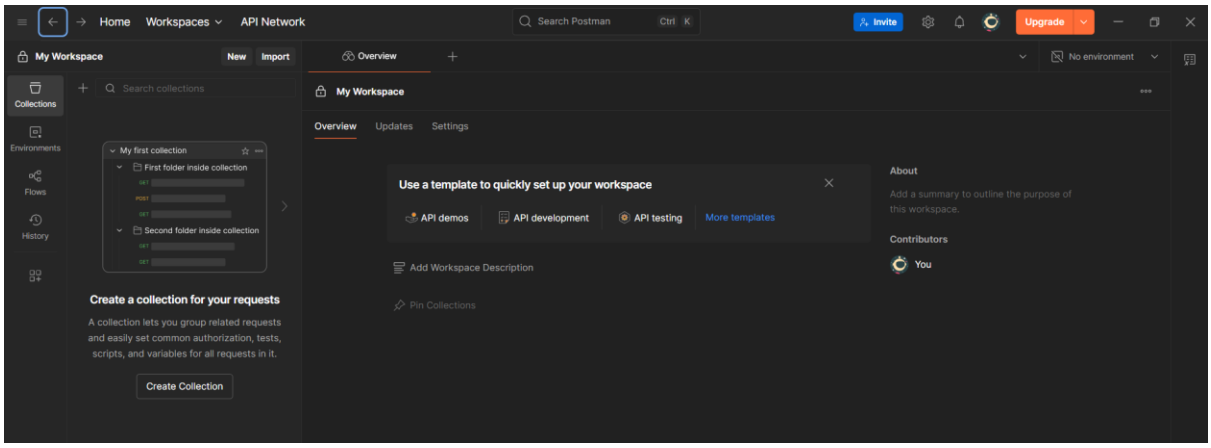
```
const express = require('express')
const app = express()
const port = 3000

app.use(express.static("public"))
app.get('/', (req, res) => {
    console.log("Hey it is a get request")
    res.send('Fetching all users...')
})

app.post('/', (req, res) => {
    console.log("Hey it is a post request")
    res.send('Post request done...')
})

app.put('/', (req, res) => {
    console.log("Hey it is a put request")
    res.send('Put request done...')
})

//to get the index.html properly rendered
// app.get('/index', (req, res) => {
//     console.log("Hey its index")
//     res.sendFile('templates/index.html', {root: __dirname})
// })

// for api adn JSON
app.get('/api', (req, res) => {
    console.log("Hey its index")
    res.json({a:1, b:2, c:3}, {root: __dirname})
})
app.listen(port, () => {
    console.log(`Example app listening on port ${port}`)
})
```

Output: browser

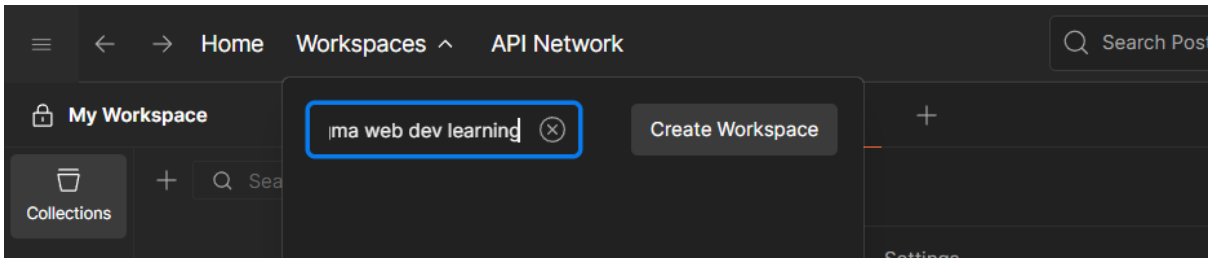



Now, postman:

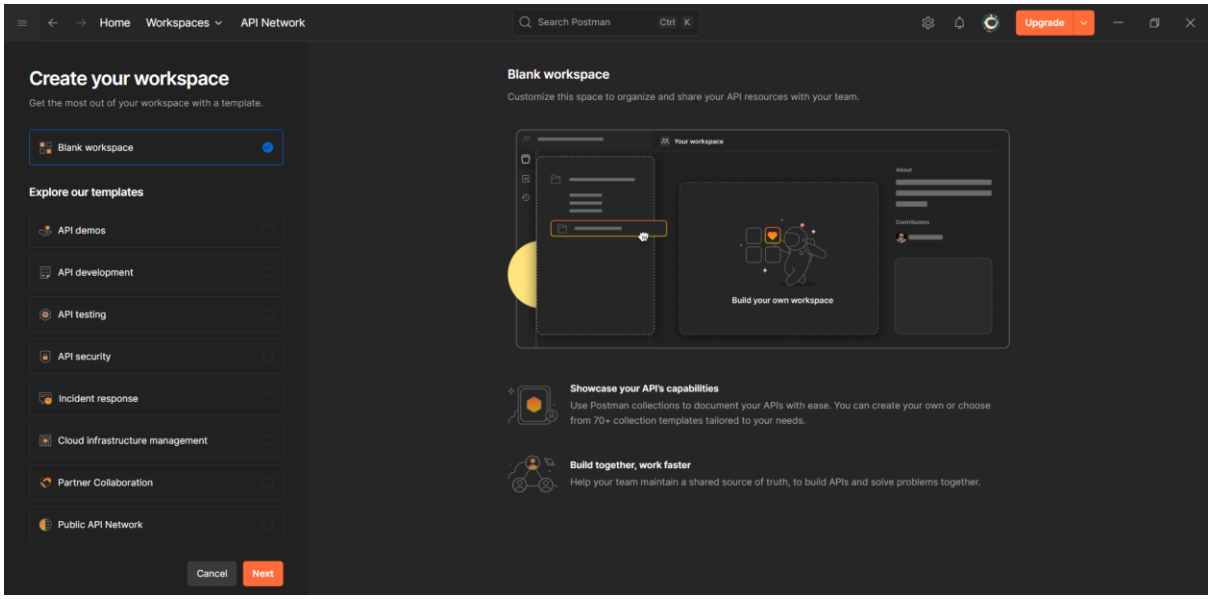
Creating a new workspace in the Postman:



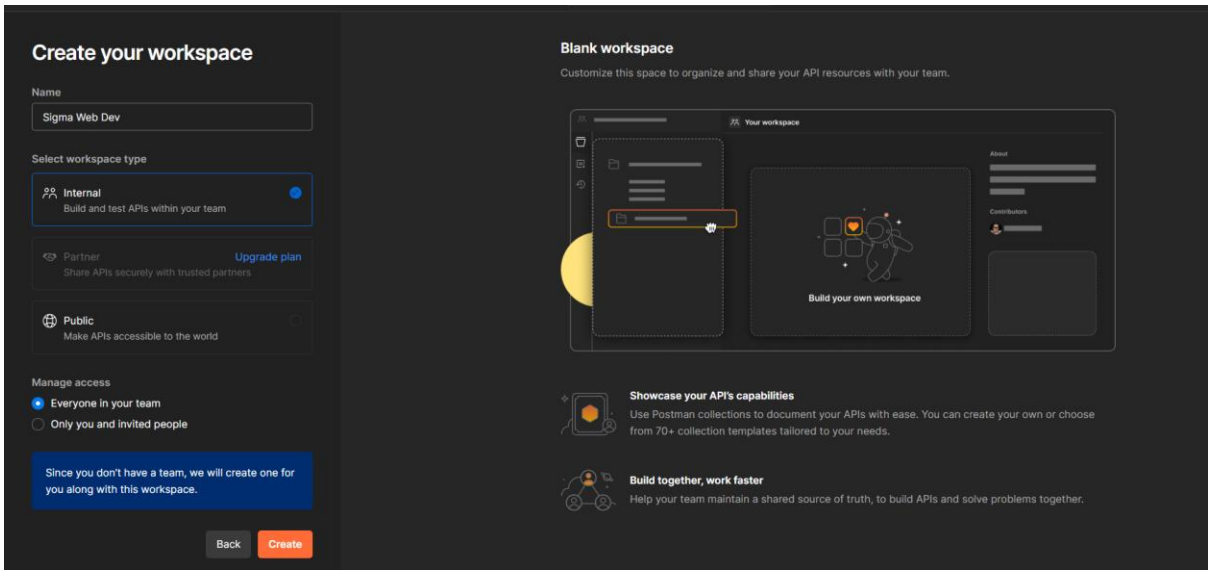

Type the name:



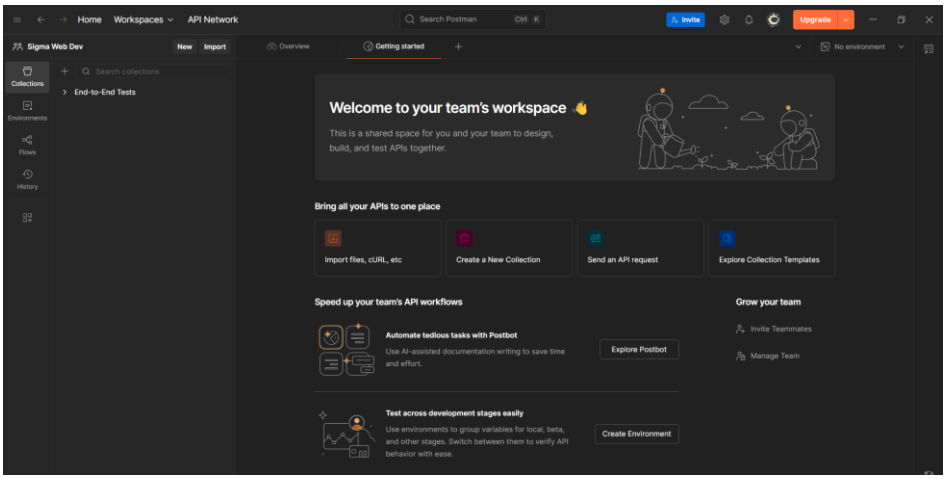
Following window will appear: click on the “blank workspace” option.



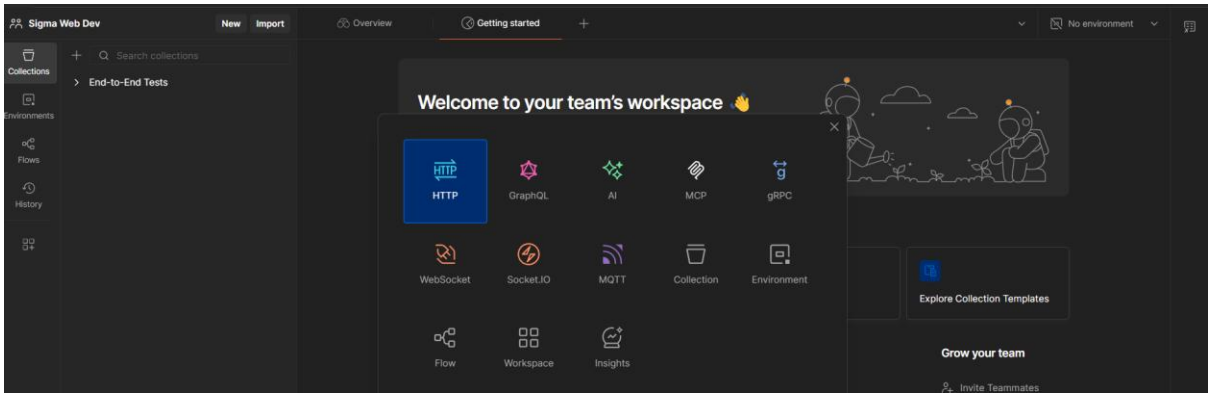

Specify the name:



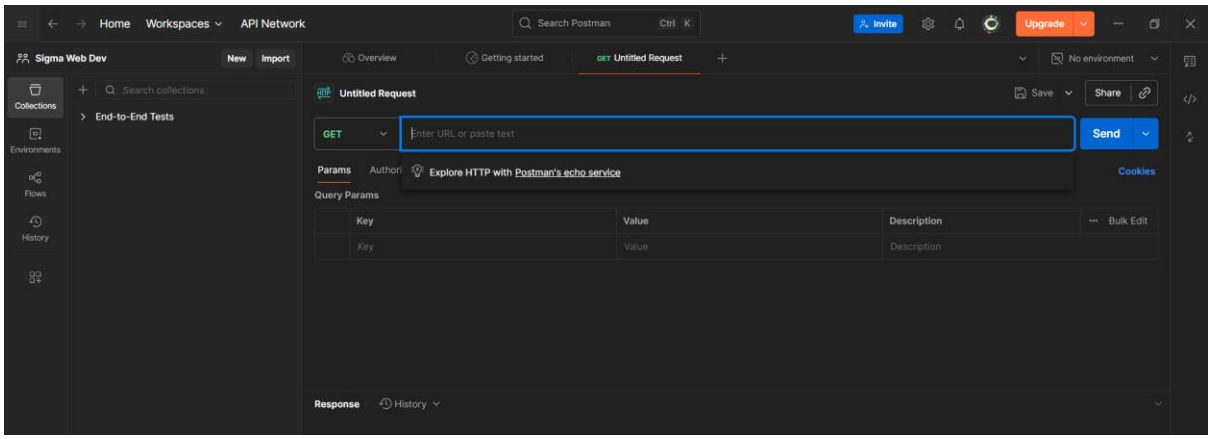

Following window confirms that workspace is created.



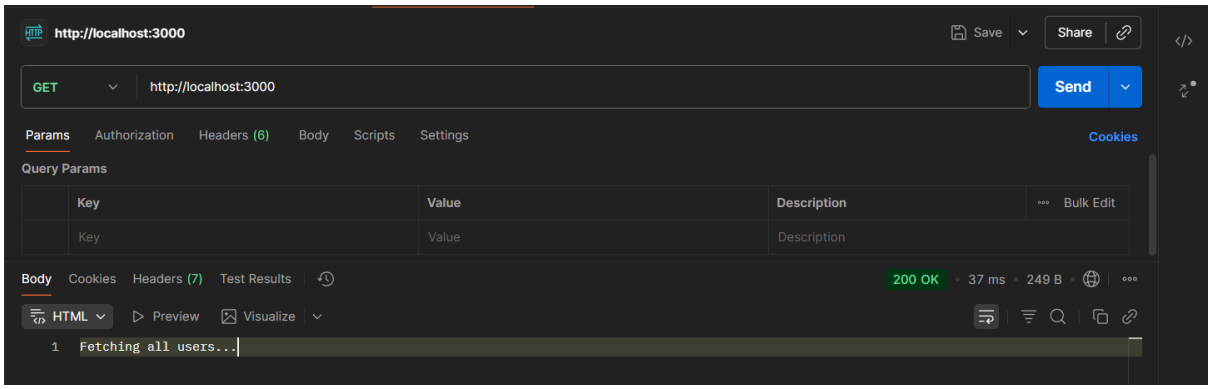
Now, click on the “New”, following options will occur:



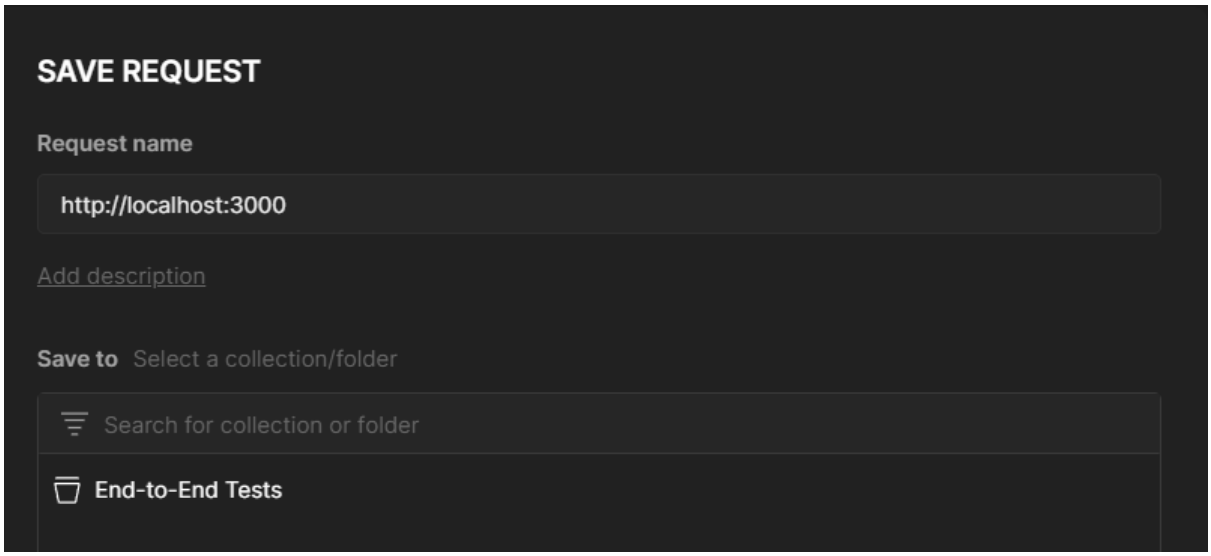

Click on the “HTTP”: following window will appear



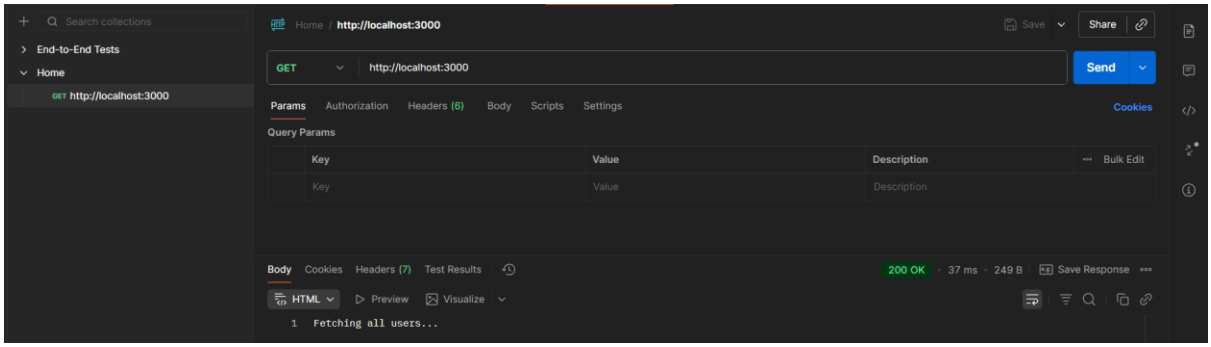

Now, just type the local host address in the search box: clearly, it worked



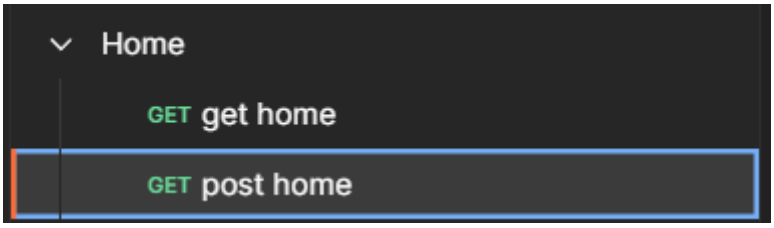
Now, we can also collect and arrange the requests in the postman, just click on “save” button: following screen will appear



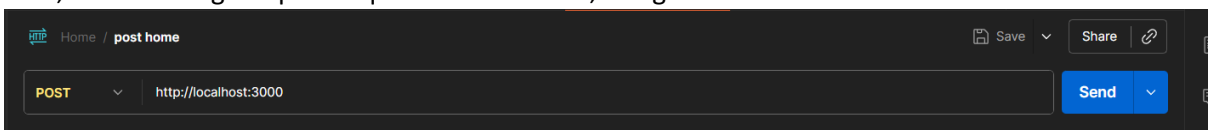

Suppose we create “home” named workspace and saved the content.



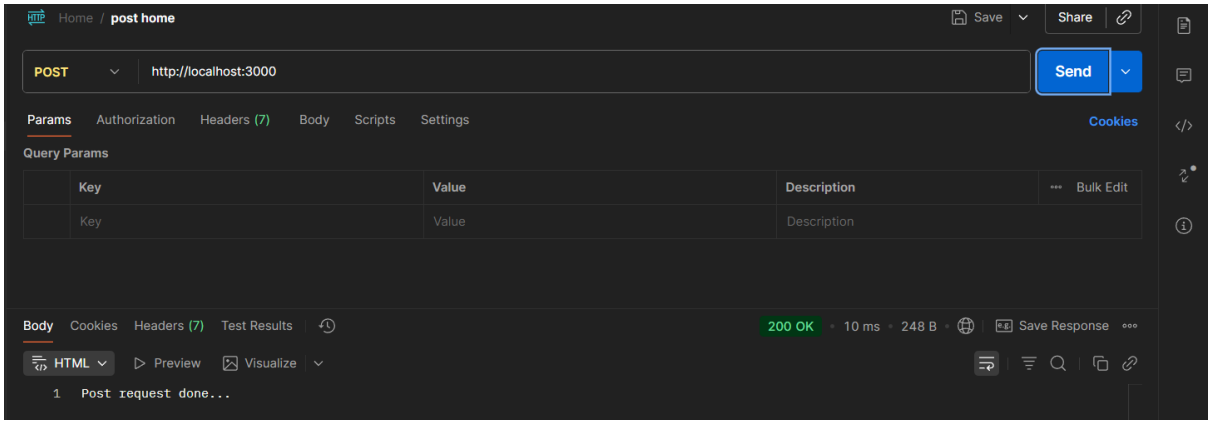

We can also rename and duplicate:




Now, we are doing the post request on the Home, using the



Just click on the “Send” button to test: clearly, it worked




--The End--