Full Stack Development - 22CS2305

MODULE 5 – EXPRESS JS

Theory - Modules

Module	Chapter	Chapter Name
1	1	Markup Language (HTML5)
2	2	CSS3
3	3	JavaScript
4	4	Node JS
5	5	**Express JS**, React JS Basics

Module 5 – Express JS

Topics
1. Introducing Express: Basics of Express 2. Express JS Middleware 3. Serving Static Pages 4. Request and Response

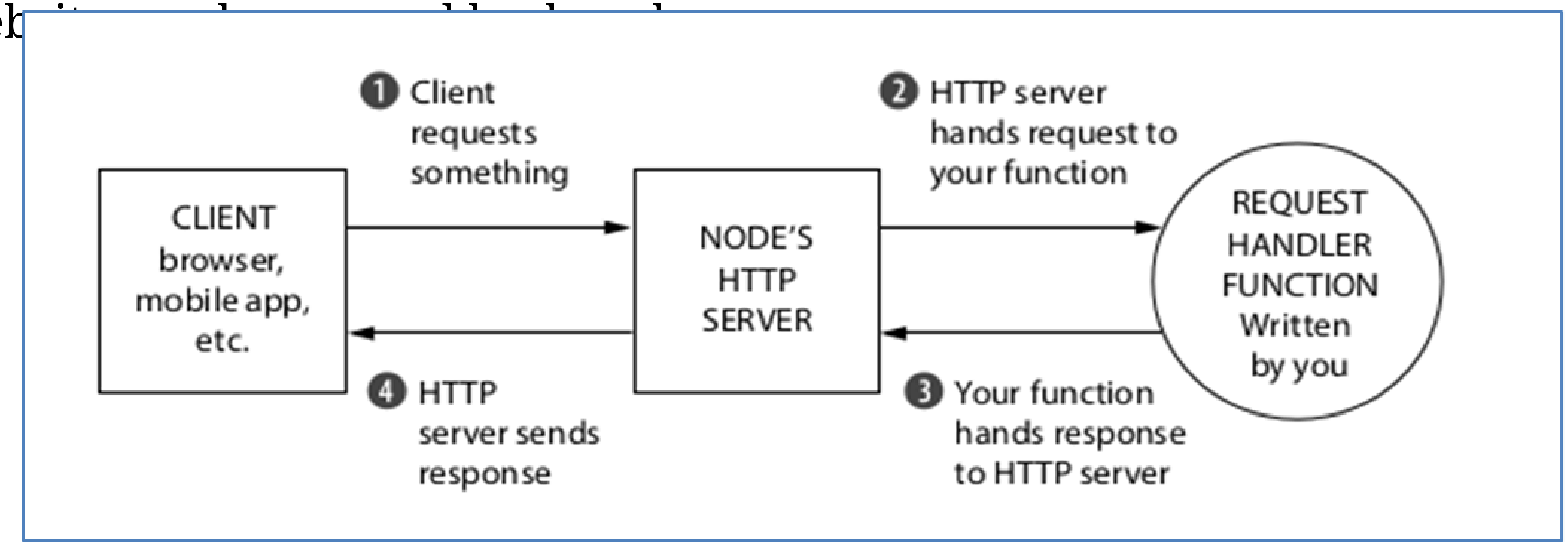
1. Introducing Express

- **Express.js (Express)** is a light web framework which sits on top of Node.js and it adds functionality like

  (middleware, routing, etc.) and simplicity to Node.js.

  - **Express.js** is a Node.js framework. It's the most popular framework .

  - **ExpressJS** is a web application framework that provides you with a simple API to build web



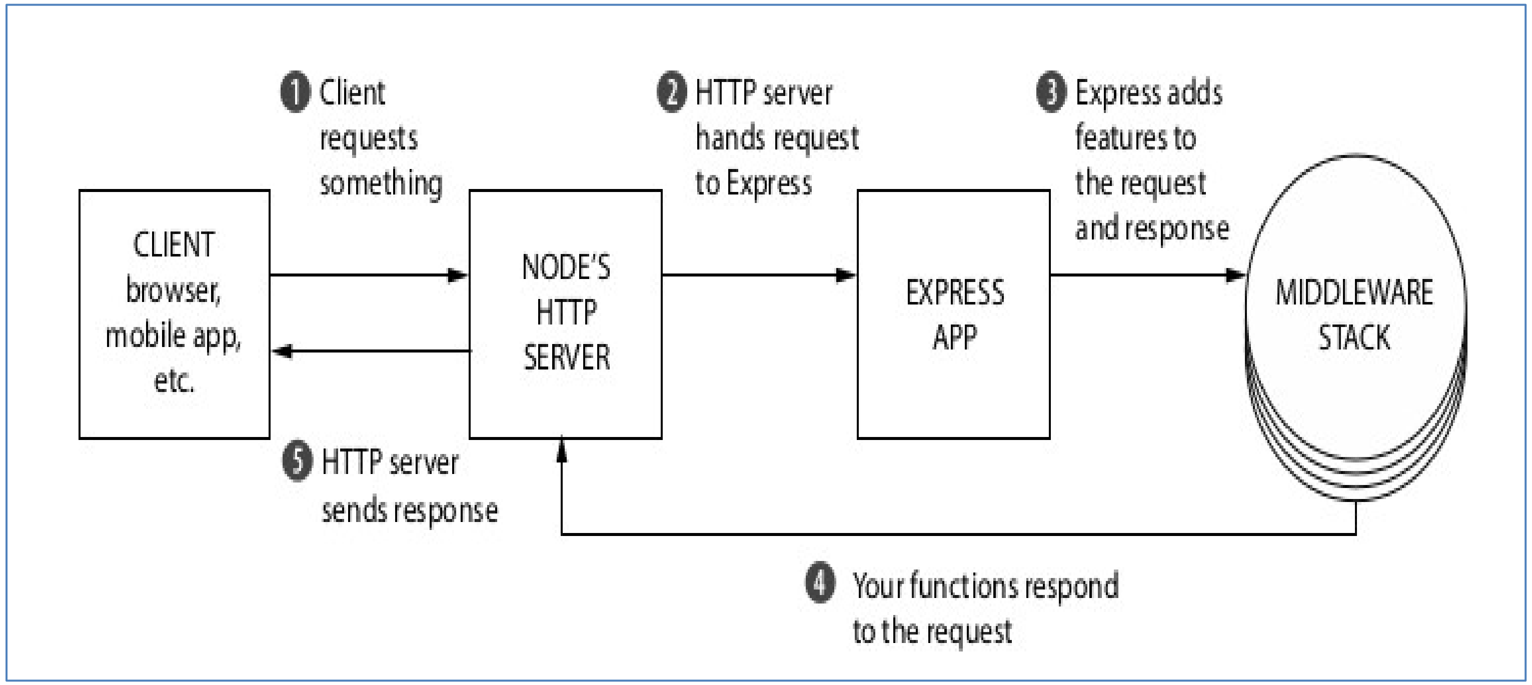
1 Client requests something
2 HTTP server hands request to your function
CLIENT browser, mobile app, etc.
NODE'S HTTP SERVER
REQUEST HANDLER FUNCTION Written by you
4 HTTP server sends response
3 Your function hands response to HTTP server

Introduction

- Node.js APIs can get complex and writing how to handle a single request can end up being over 50 lines of

  code.

- Express makes it easier to write Node.js web applications.


1 Client requests something
2 HTTP server hands request to Express
3 Express adds features to the request and response
CLIENT browser, mobile app, etc.
NODE'S HTTP SERVER
EXPRESS APP
MIDDLEWARE STACK
5 HTTP server sends response
4 Your functions respond to the request

Advantages

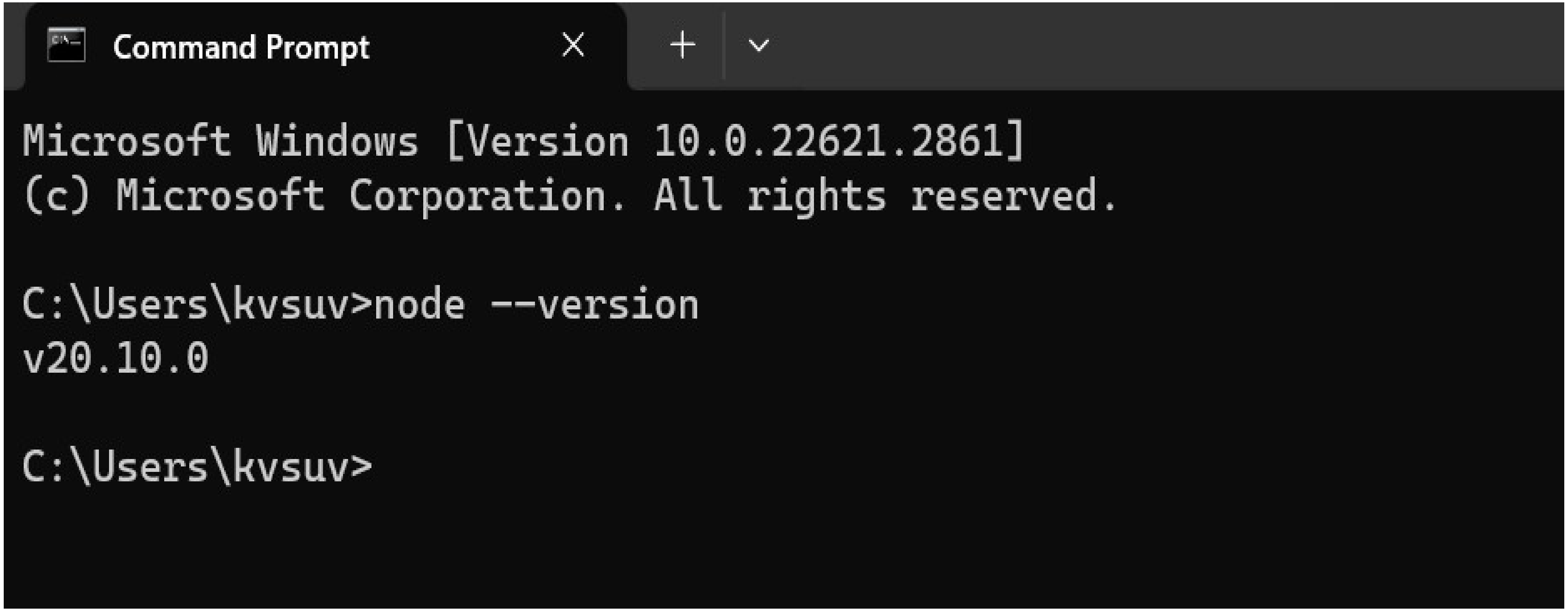
- Develops Node.js web applications ***quickly and easily.***
- It's ***simple to set up and personalize.***
- Allows you to ***define application routes*** using HTTP methods and URLs.
- Includes a number of ***middleware modules*** that can be used to execute

  additional requests and responses activities.

- ***Simple to interface*** with a variety of template engines, including Jade, Vash, and EJS.
- Allows you to specify a middleware for ***handling errors.***

Dayananda Sagar University Bengaluru

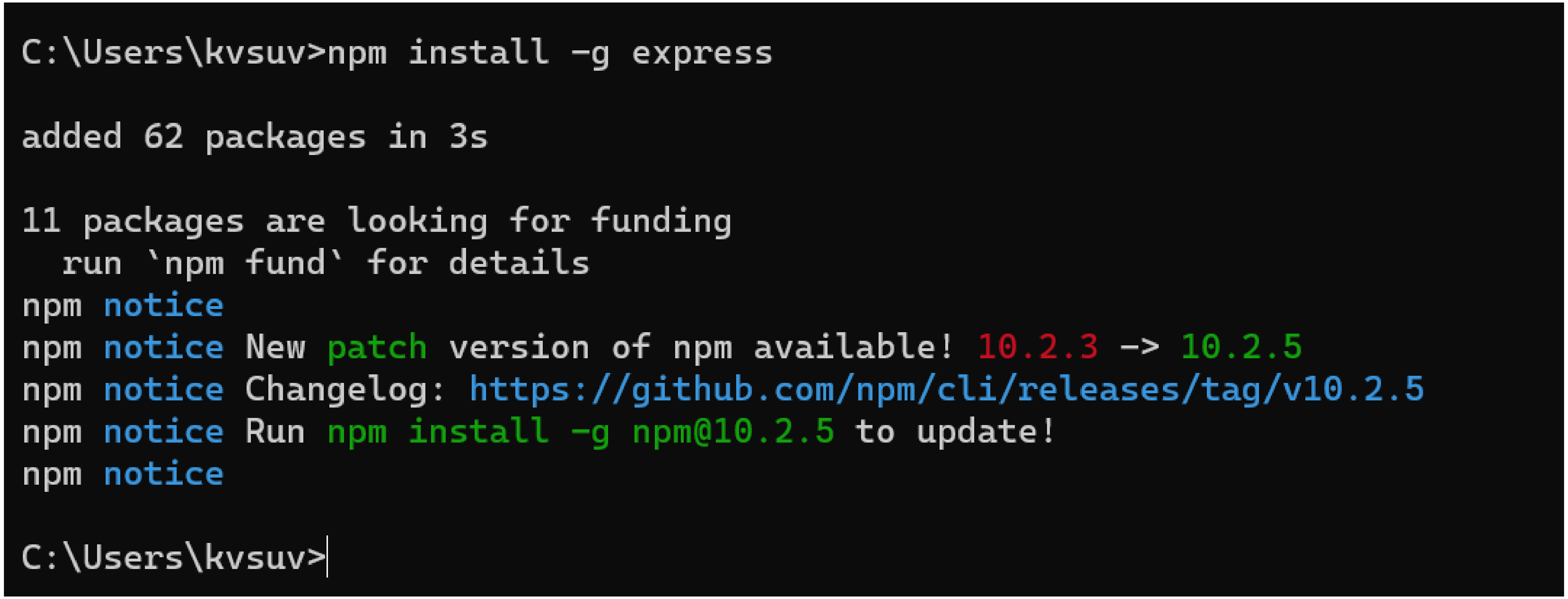
Version


Command Prompt

```
Microsoft Windows [Version 10.0.22621.2861]
(c) Microsoft Corporation. All rights reserved.

C:\Users\kvsuv>node --version
v20.10.0

C:\Users\kvsuv>
```

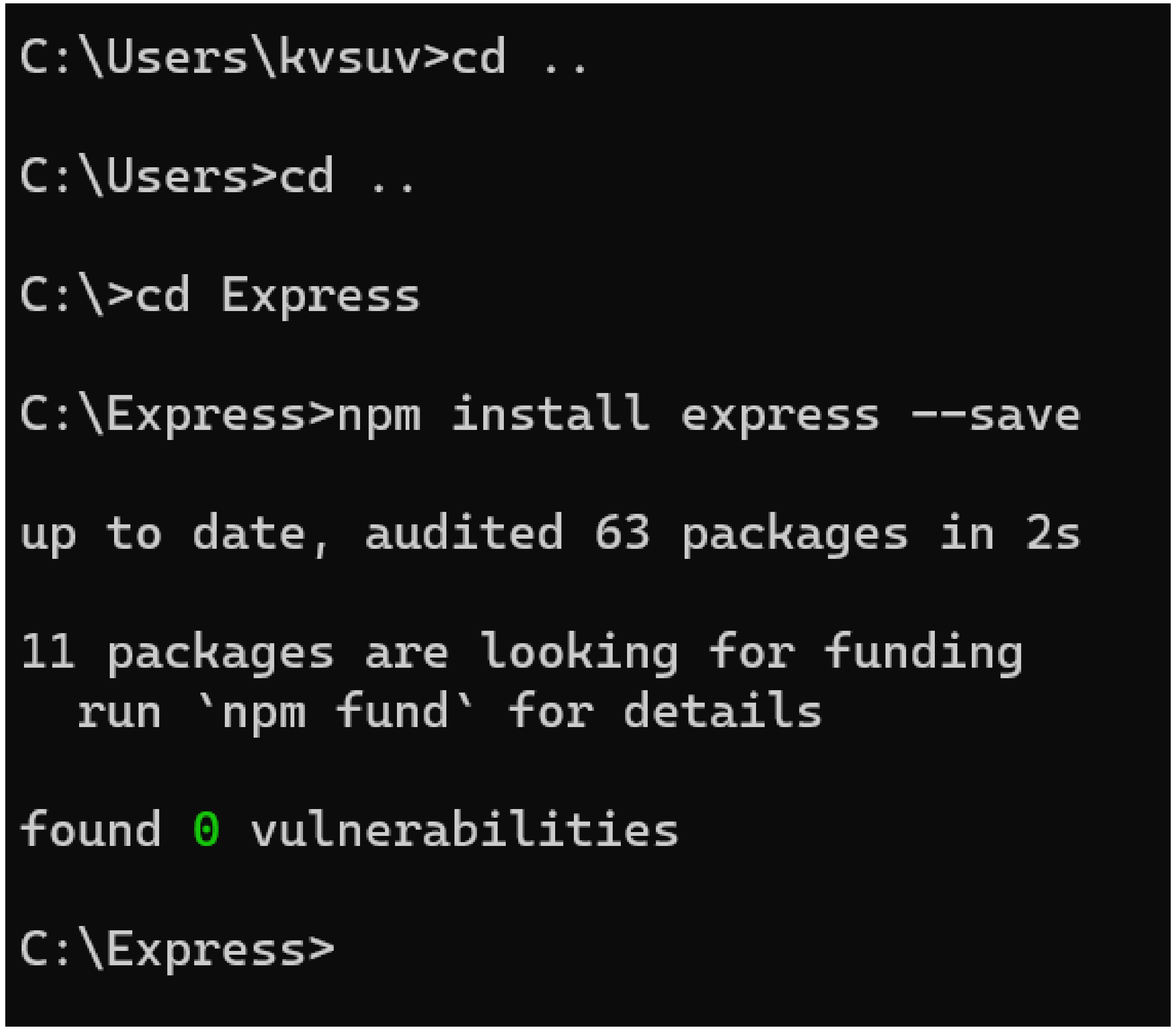
Installing Express

npm install –g express



```
C:\Users\kvsuv>npm install -g express

added 62 packages in 3s

11 packages are looking for funding
  run `npm fund` for details
npm notice
npm notice New patch version of npm available! 10.2.3 -> 10.2.5
npm notice Changelog: https://github.com/npm/cli/releases/tag/v10.2.5
npm notice Run npm install -g npm@10.2.5 to update!
npm notice

C:\Users\kvsuv>
```

Installing Express



```
C:\Users\kvsuv>cd ..

C:\Users>cd ..

C:\>cd Express

C:\Express>npm install express --save

up to date, audited 63 packages in 2s

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

found 0 vulnerabilities

C:\Express>
```

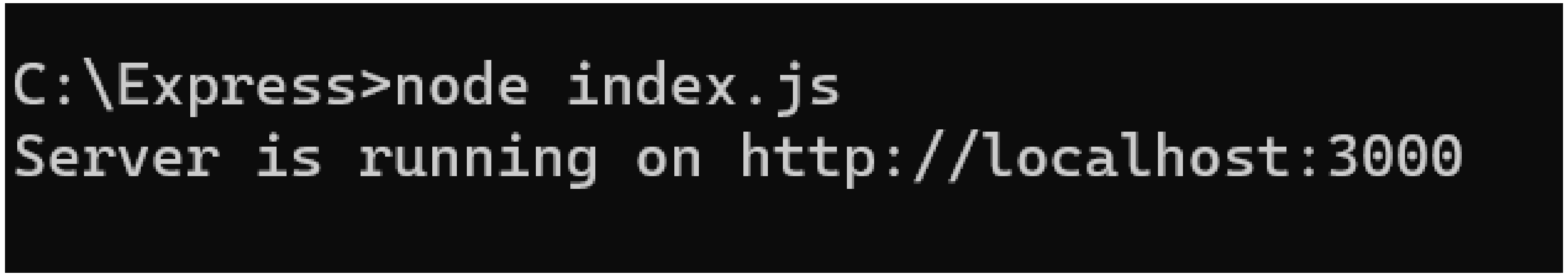
npm install express --save

Dayananda Sagar University Bengaluru

Simplest Express Application - 1

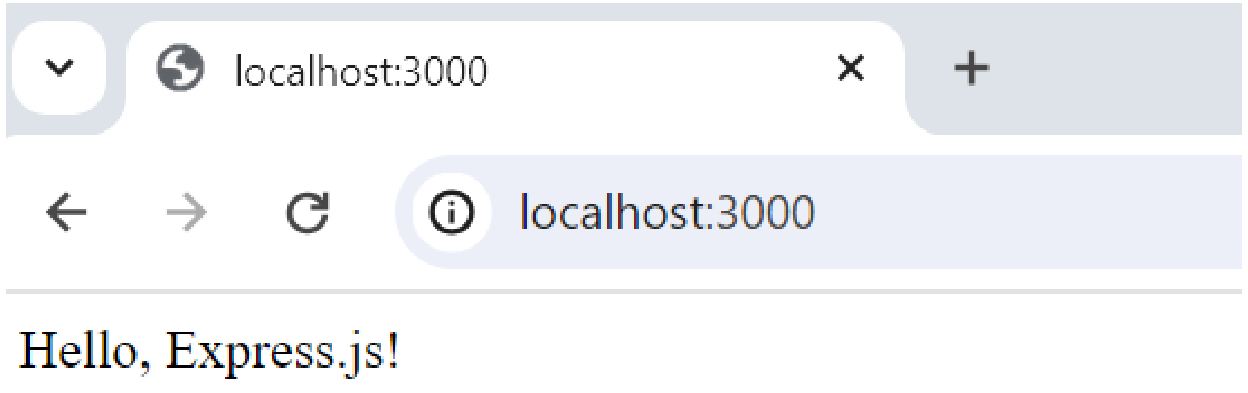
```
const express = require('express'); const app = express();

const port = 3000;


app.get('/', (req, res) => res.send('Hello, Express.js!'));


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

Output – Command Prompt



```
C:\Express>node index.js
Server is running on http://localhost:3000
```

Output – Browser



Hello, Express.js!

CRUD Operations

1. CREATE – Create new resource {Adding a new user/ new task/ new transaction}

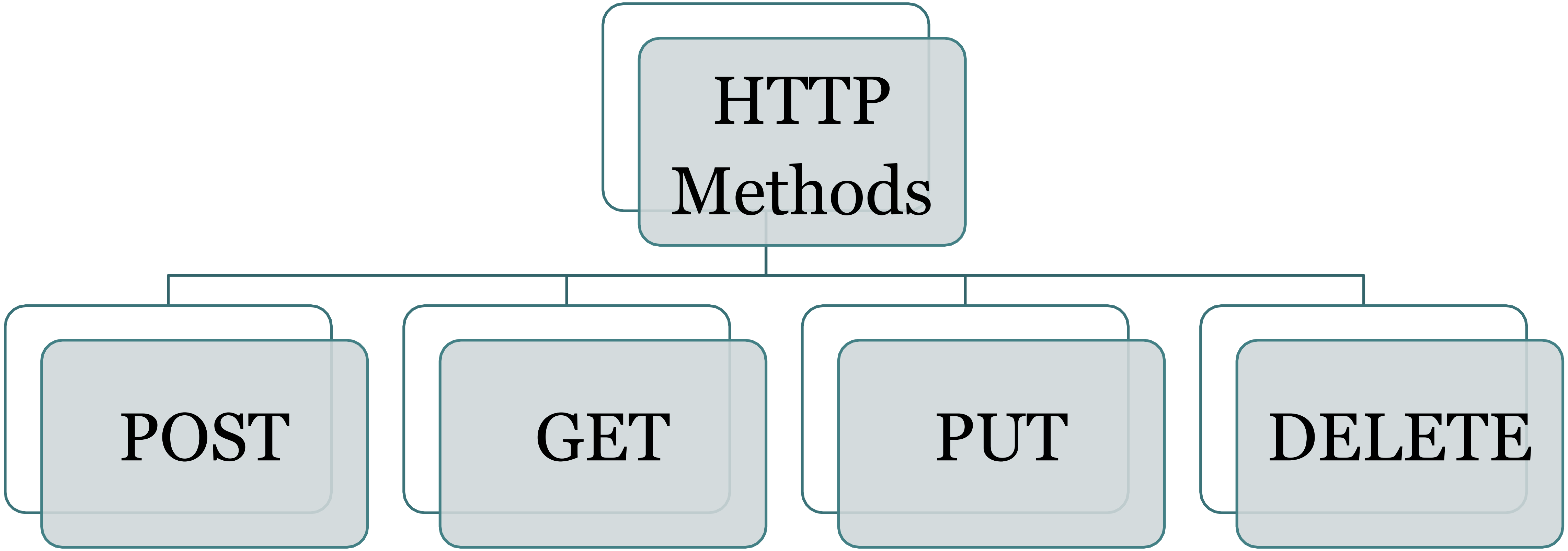
2. READ – Read resource from server {Reading/ Fetching resources – Analytical}

3. UPDATE – Update a resource {Based on some condition – Modifying data}

4. DELETE – Delete a resource

HTTP Methods

- The HTTP method is supplied in the request and specifies the operation that the client has requested.



HTTP Methods

POST

The POST method requests that the server accept the data enclosed in the request as a new object/entity of the resource identified by the URI.

GET

The GET method requests a representation of the specified resource. Requests using GET should only retrieve data and should have no other effect.

HTTP Methods

PUT

The PUT method requests that the server accept the data enclosed in the request as a modification to existing object identified by the URI. If it does not exist then the PUT method should create one.

DELETE

The DELETE method requests that the server delete the specified resource.

2. Express JS Middleware

- Middleware is a set of functions that sit between a raw request and the final intended route.
- Middleware functions have access to *all* the HTTP requests coming to the server.
- Middleware can handle tasks such as **logging, sending static files, authorization, and session**

  management, etc.
- In Express the request and response objects are passed through a set of

  functions, called the middleware stack.

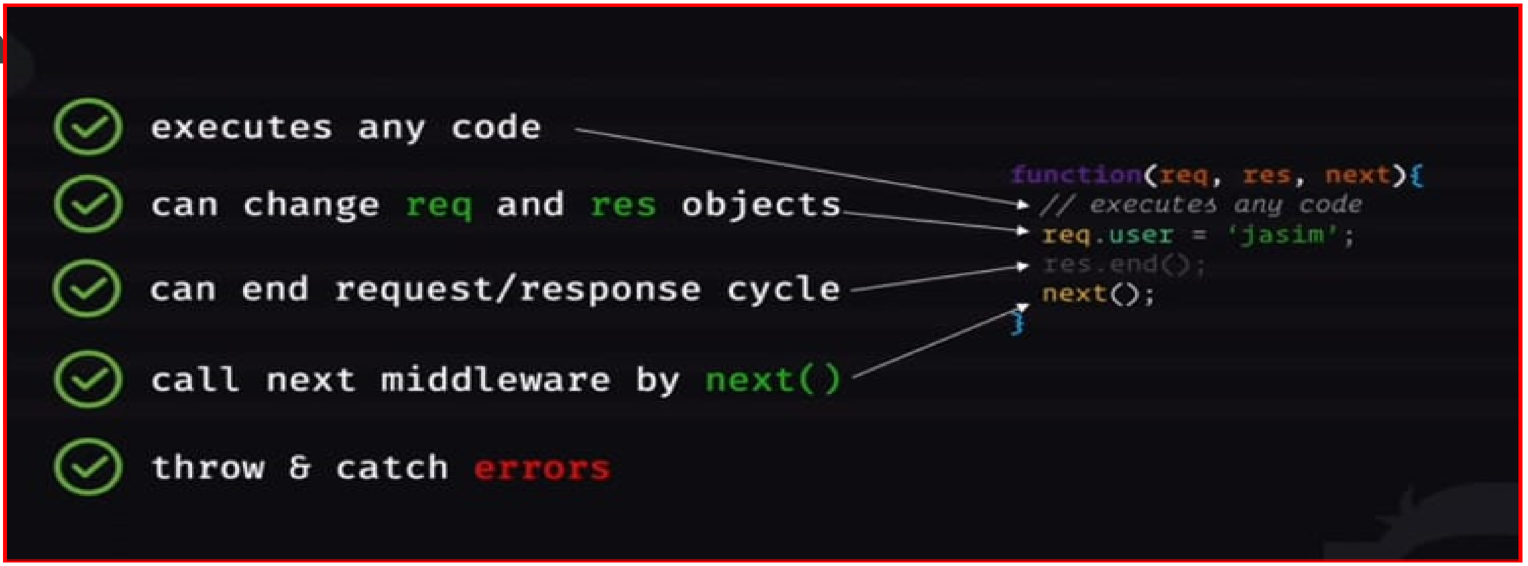
- Every function in the stack takes three arguments request, response and next.

  next is a function, that when called Express executes the next function in the stack.

- This is a subtle difference between middleware and a route handler which we saw above

Middleware Functions

- Middleware functions can perform the following tasks:
  - Execute any code.
  - Make changes to the request and the response objects.
  - End the request-response cycle.
  - Call the next middleware function in the stack.
- If the current middleware function does not end the request-response cycle, it must call next() to pass control to the next middleware function. Otherwise, the request will be left han


executes any code
can change req and res objects
can end request/response cycle
call next middleware by next()
throw & catch errors

```
function(req, res, next){
  // executes any code
  req.user = 'jasim';
  res.end();
  next();
}
```

Middleware Advantages

Optimization and better performance
Can manipulate request object before reaching

the server Can perform various functions on

the request object

Can improve client-side rendering

performance Setting some specific HTTP

headers

Middleware Types

Application level middleware	**app.use**
Router level middleware	**router.use**
Built-in middleware	**express.static, express.json, express.urlencoded**
Error handling middleware	**app.use(err, req, res, next)**
Third party middleware	**bodyparser, cookie-parser**

Middleware Working

- ***Middleware*** functions are functions that have access to the request object (req), the response object (res),

  and the next middleware function in the application’s request-response cycle.

- The next middleware function is commonly denoted by a variable named next.

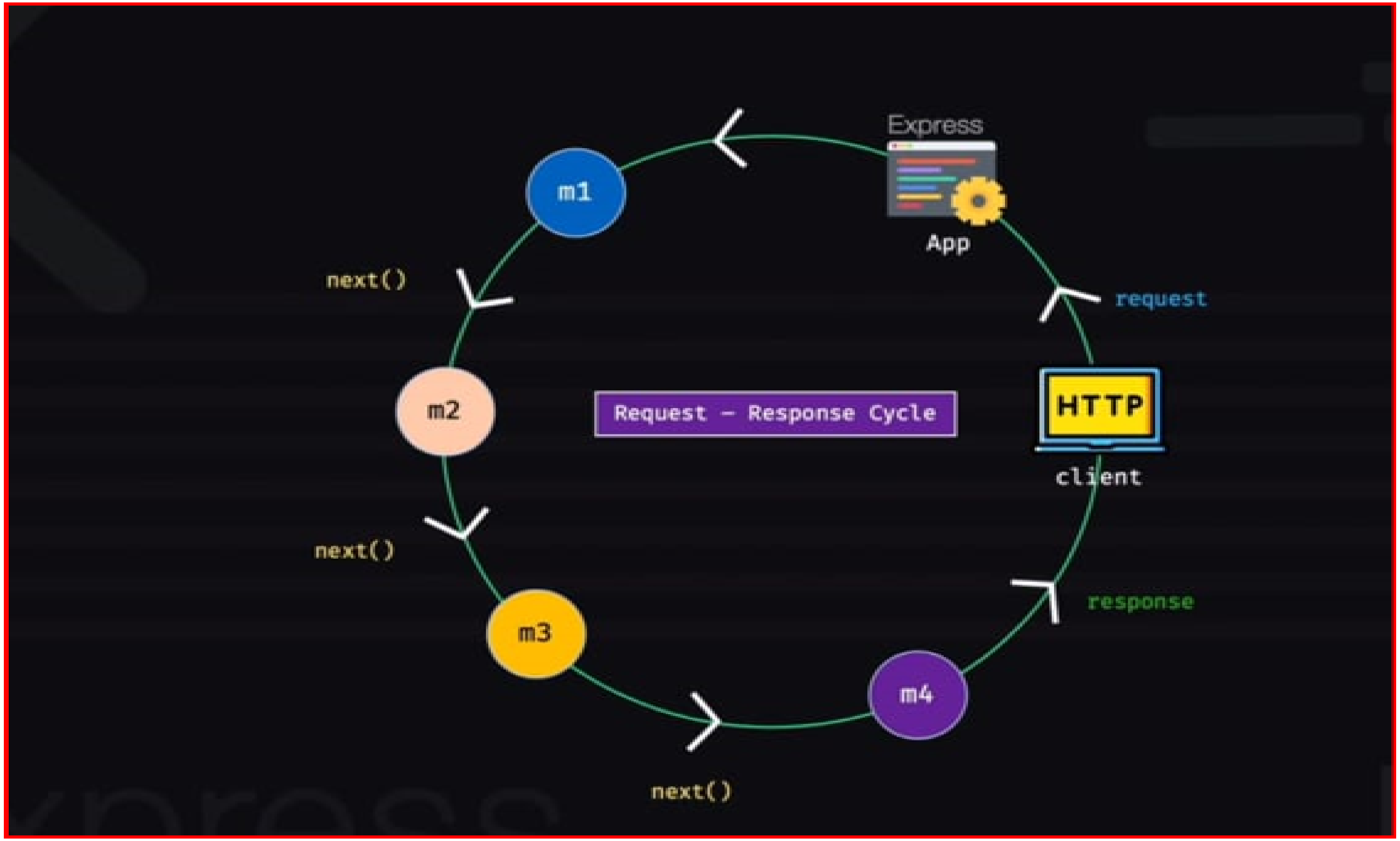
- As name suggests it comes in middle of something and that is request and response cycle:

  1. Middleware has access to request and response object.

  2. Middleware has access to next function of request-response life cycle.

Dayananda Sagar University Bengaluru

Middleware Working



Middleware Working – next( )

- A middleware is basically a function that will the receive the Request and Response objects, just like your route Handlers do.

- As a third argument you have another function which you should call once your middleware code completed.
- This means you can wait for asynchronous database or network operations to finish before proceeding to the next step.

- This might look like the following: If the current middleware function does not end the request-response cycle, it must call next() to pass control to the next middleware function.

- Otherwise, the request will be left hanging.

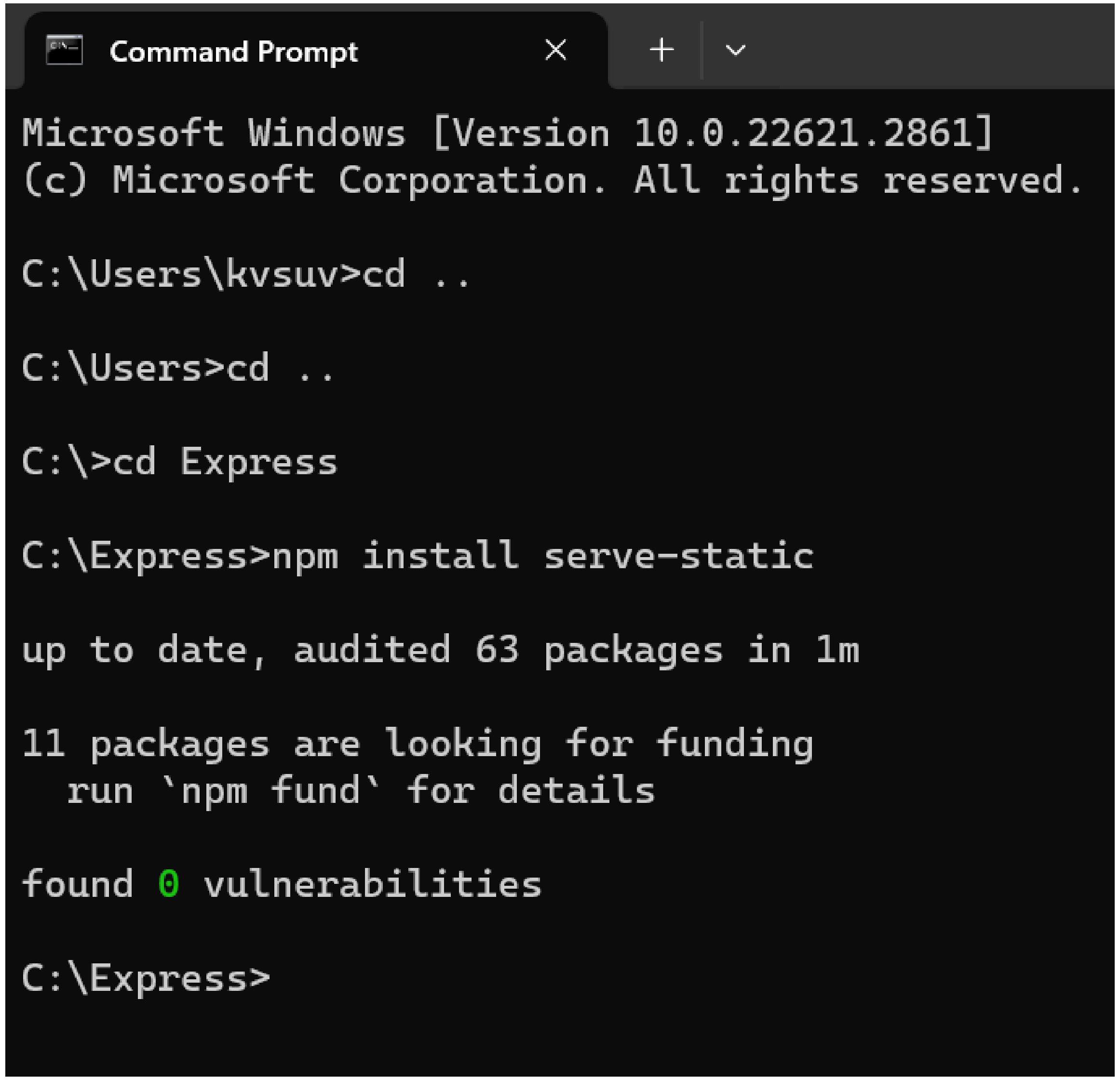
3. Serve Static Pages

- "Serving static pages" refers to the practice of delivering web pages that do not change content dynamically based on user interactions or database queries.
- Instead, the content of these pages remains fixed or "static" until the webmaster or developer manually updates them.
- One of the most common things to do is serve static web-site content.
- The server-static middleware (npm install serve-static) is designed specifically for that.

```
app =Express()
app.use(express.static(public DirectoryPath))
```

Dayananda Sagar University Bengaluru

Installing serve-static



Command Prompt

```
Microsoft Windows [Version 10.0.22621.2861]
(c) Microsoft Corporation. All rights reserved.

C:\Users\kvsuv>cd ..

C:\Users>cd ..

C:\>cd Express

C:\Express>npm install serve-static

up to date, audited 63 packages in 1m

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

found 0 vulnerabilities

C:\Express>
```

npm install serve-static

Serve Static Pages - Code

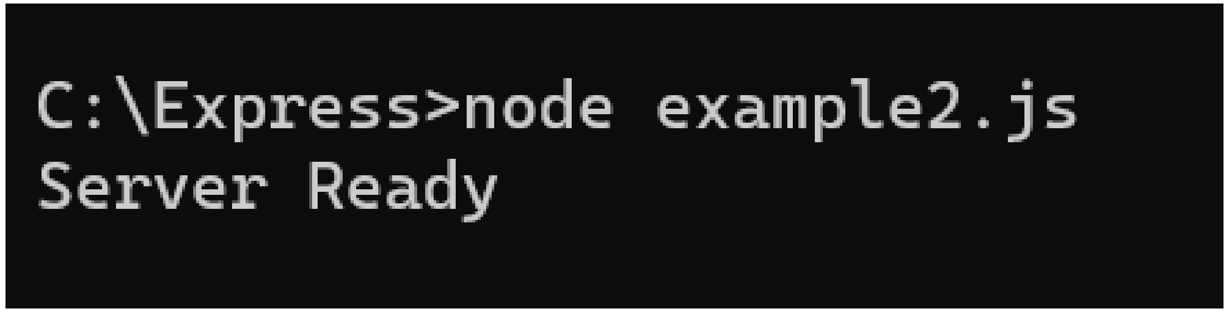
```
const express = require('express');
const app = express(); const port = 3000;


app.use(express.static('public'));

// note the index file that is present in the public
folder will be directly executed on '/'  in the browser
by default

app.listen(port, () => console.log(`Server
Ready`));
```

Code Snippet – HTML(index.html)

```
<!DOCTYPE html>
<html>
        <head>
                <title>Inline</title>
        </head>
        <body>
            <h1 style = "font-family : Lucida Handwriting;
                font-size : 50pt;
                color : red;
                text-align : center;"> Dayananda Sagar University </h1>
        </body>
</html>
```

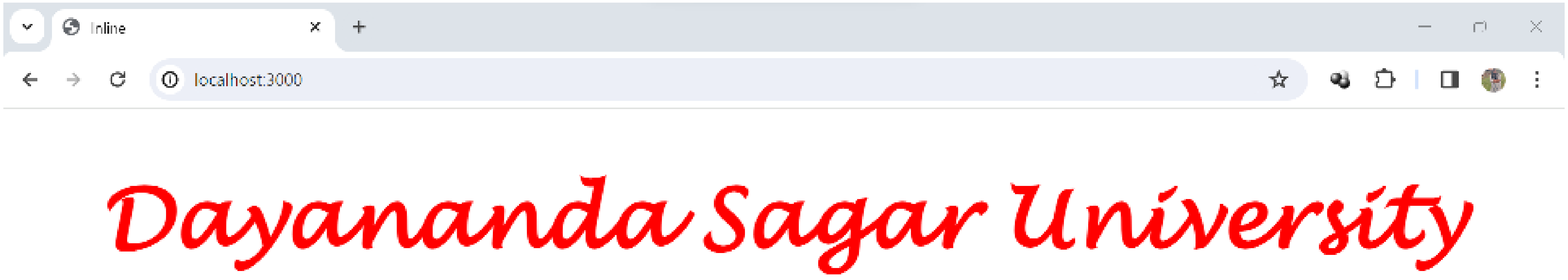
1. Create a folder named as public in your working directory.
2. Create an HTML file inside that directory.**(index.html)**

Output – Command Prompt



```
C:\Express>node example2.js
Server Ready
```

Output – Browser



Dayananda Sagar University

4. Request and Response

- Express application uses a callback function whose parameters are request and response objects

  app.get('/', function (req, res) { // -- })

- ***Request Object*** – The request object represents the HTTP request and has properties for the request query

  string, parameters, body, HTTP headers, and so on.

- ***Response Object*** – The response object represents the HTTP response that an Express app

  sends when it gets an HTTP request.

4. Request and Response

- Express application uses a callback function whose parameters are request and response objects

```
app.get('/', function (req, res) { // -- })
```

- ***Request Object*** – The request object represents the HTTP request and has properties for the request query

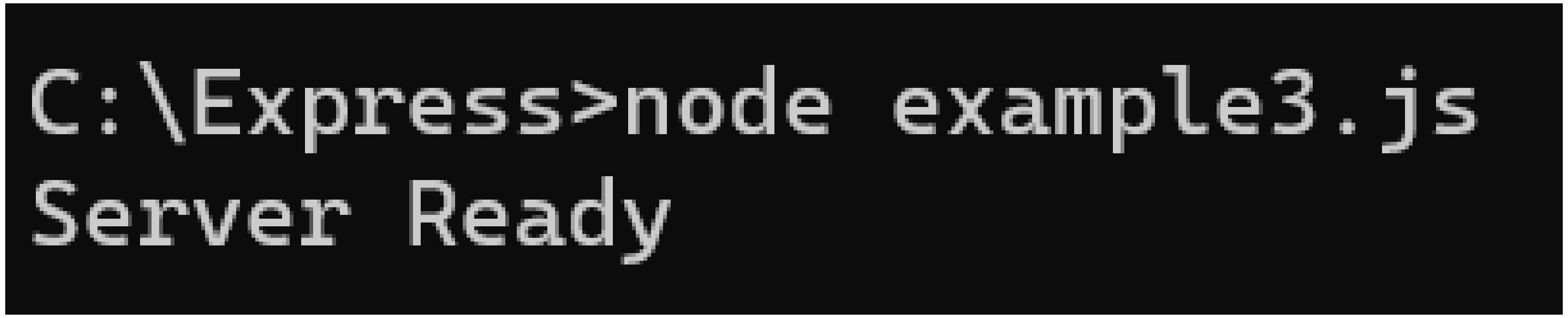
  string, parameters, body, HTTP headers, and so on.
- ***Response Object*** – The response object represents the HTTP response that an Express app

  sends when it gets an HTTP request.

Code Snippet

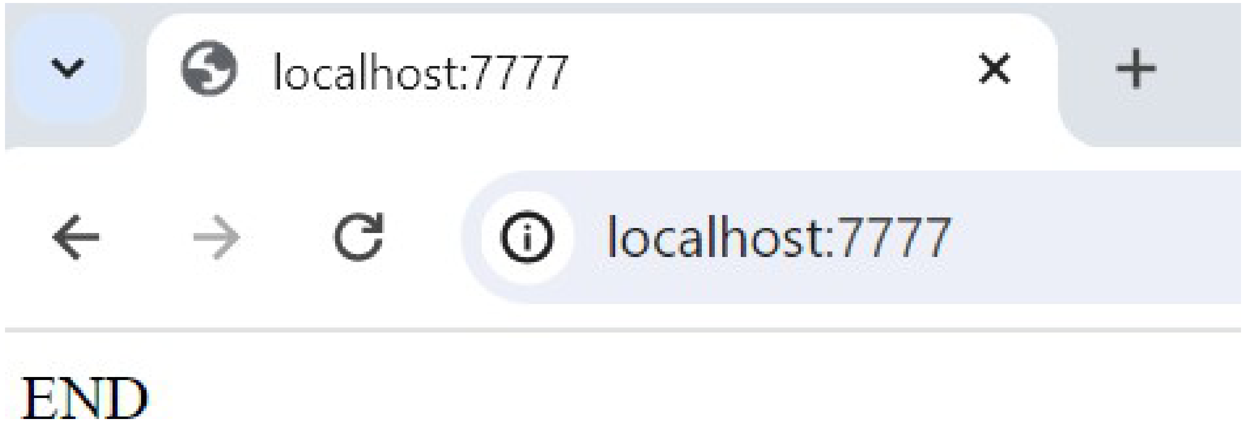
```
const exp=require("express"); const app=exp();

app.get("/",function(req,res){res.send("END");});
app.get("/so",function(req,res){res.send("OF")}); app.

get("/finally",function(req,res){res.

send("SYLLABUS")});

app.get('*',(req,res)=>{res.send(“Good Luck for

Exams !!");}); app.listen(7777, () => console.

log(`Server Ready`));
```

Output – Command Prompt



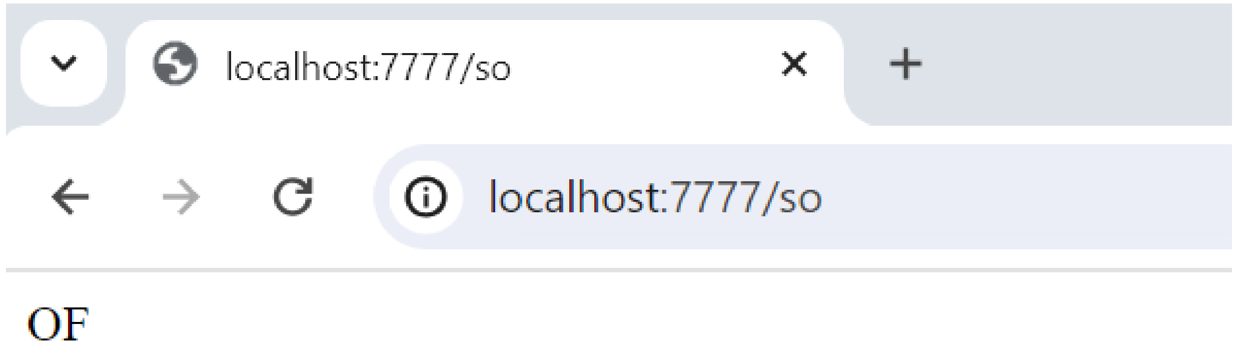
```
C:\Express>node example3.js
Server Ready
```

Output1 – Browser – localhost/7777



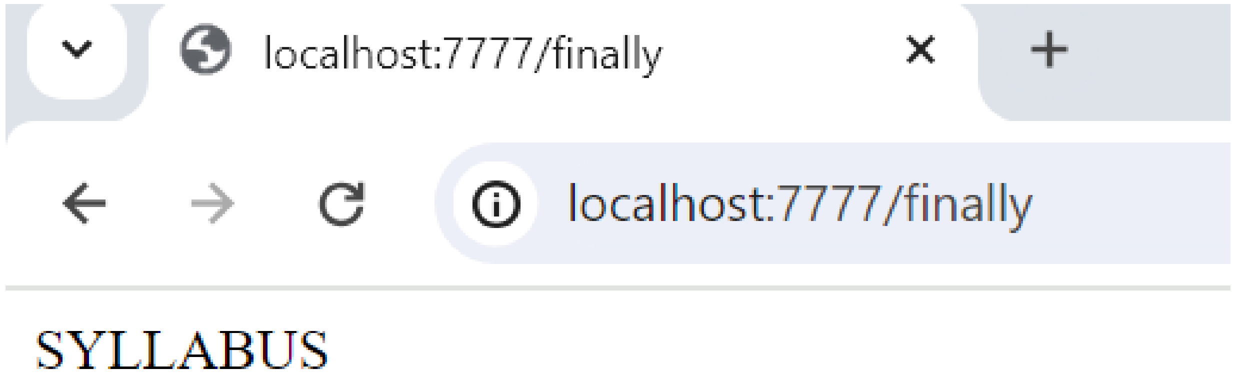
```
END
```

Output2 – Browser – localhost/so



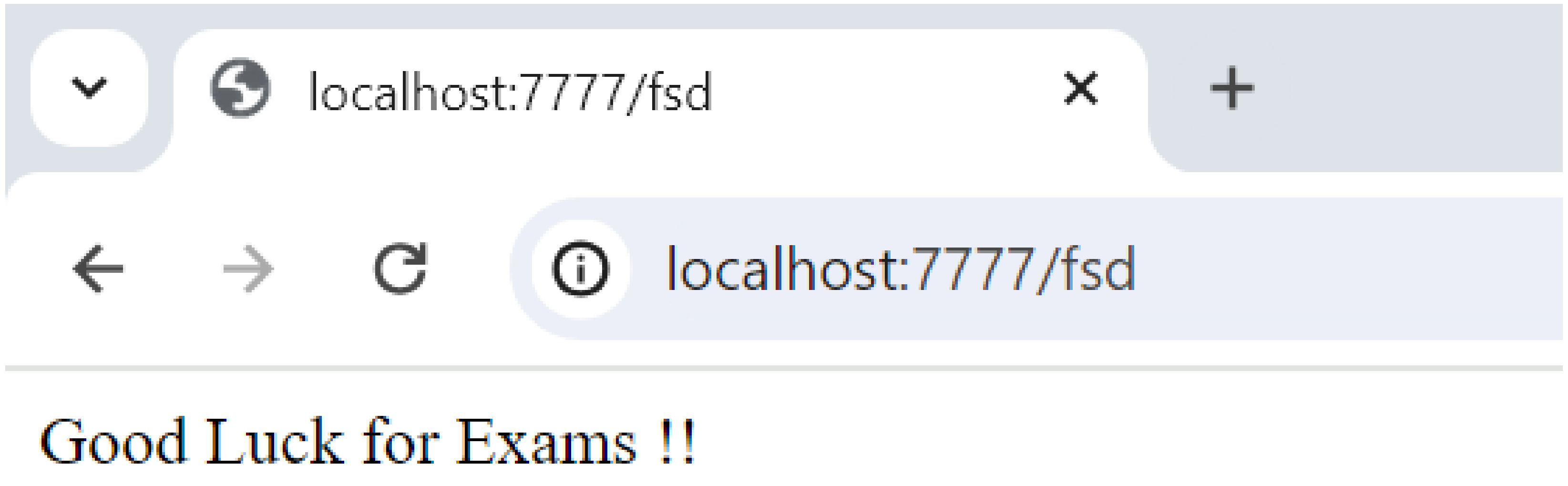
```
OF
```

Output3 – Browser – localhost/finally



```
SYLLABUS
```

Output4 – Browser – localhost/fsd



```
Good Luck for Exams !!
```

END OF MODULE