**Express.js**

**What is Express.js**

* Express is a fast, assertive, essential and moderate web framework of Node.js.
* You can assume express as a layer built on the top of the Node.
* express js that helps manage a server and routes.
* It provides a robust set of features to develop web and mobile applications.

**Features Of Express Framework:**

* It can be used to design single-page, multi-page and hybrid web applications.
* It allows to setup middlewares to respond to HTTP Requests.
* It defines a routing table which is used to perform different actions based on HTTP method and URL.
* It allows to dynamically render HTML Pages based on passing arguments to templates.

**Why use Express**

* Ultra-fast I/O.
* Asynchronous and single threaded.
* MVC like structure.
* Robust API makes routing easy.

# Installing

**step 1 : open cmd and type given command**

**(ex) npm install express**

# Hello world example

**const** express = **require(**"express"**);**

**const** app = express();

app**.get(**"/",(request,response) => {

response**.end(**"<h1> Welcome to express js </h1>"**);**

});

//listen express server

app**.listen(**3000,() => {console.log("Server is running ....");}**);**

**Run : node <fileName>.js**

**OPEN WEB BROWSER 🡺 REQUEST THIS URL : localhost:3000/**

**output :** Welcome to express js

# Basic routing

Routing refers to determining how an application responds to a client request to a particular endpoint, which is a URI (or path) and a specific HTTP request method (GET, POST, and so on).

**Syntax : app.METHOD(PATH, HANDLER);**

**Example :**

//create simple server use express frame work

//import express module from node core module

**const** express = **require(**"express"**);**

**const** app = express();

app.**get(**"/home",(request,response) => {

response**.end(**"<h1> Home Page </h1>"**);**

}**);**

app**.get(**"/about",(request,response) => {

response**.end(**"<h1> About Page </h1>"**);**

}**);**

app**.get(**"/error",(request,response) => {

response**.end(**"<p> Page not Found : ERROR CODE 404 ...... </p>"**);**

}**);**

//listen express server

app**.listen(**3000,() => {console.log("Server is running ....");}**);**

**Run : node <filename>.js**

**OPEN WEB BROWSER 🡺 REQUEST THIS URL : localhost:3000/home**

**NEXT 🡺 localhost:3000/about output :** About Page

**NEXT 🡺 localhost:3000/contact output :** Contact Page

**NEXT 🡺 localhost:3000/randomText output :** Page not Fount : 404

# Serving static files in Express

To serve static files such as images, CSS files, and JavaScript files, use the **express.static built-in middleware function** in Express.

**Syntax :** **express.static(root, [options])**

**PROJECT STRUCTURE VIEW :**

**rootFolder**

**public**

**css**

style.css

**images**

img-1.jpg

img-2.jpg

img-3.jpg

**videos**

sampleVideo.mp4

**js**

index.js

**template**

index.html

app.js

**FOLDERS :** rootFolder,public,css,images,videos,js

**FILE :** index.html,style.css,index.js

**IMAGES & VIDEOS :** img-1.jpg,img-2.jpg,img-3.jpg,sampleVideo.mp4

**FILE : index.html**

**<html>**

**<head>**

**<title>** Document **</title>**

**<link rel = "**stylesheet**" href =** "../css/style.css"**>**

**</head>**

**<body>**

**<h1>** Welcome to express **</h1>**

**<img src =** "../images/img-1.jpg" **>**

**<video src =** "../videos/sampleVideo.mp4" **controls > </video>**

**<script src =** "../js/index.js" **> </script>**

**</body>**

**</html>**

**FILE : style.css**

**h1{**

**background-color: yellow;**

**}**

**img{**

**height: 500px;**

**width: 500px;**

**border : 2px solid brown;**

**}**

**video{**

**height: 500px;**

**width: 500px;**

**}**

**FILE : index.js**

**alert("** Message From Js . . . . **");**

**FILE : app.js**

**const** express **= require("**express**");**

**const** path **= require("**path**");**

**const** app **=** express();

app**.use(**express**.static(**path**.join(\_\_dirname,"**public**")));**

app**.get("**/**",(**request**,**response**)** => {

response**.status(**200**);**

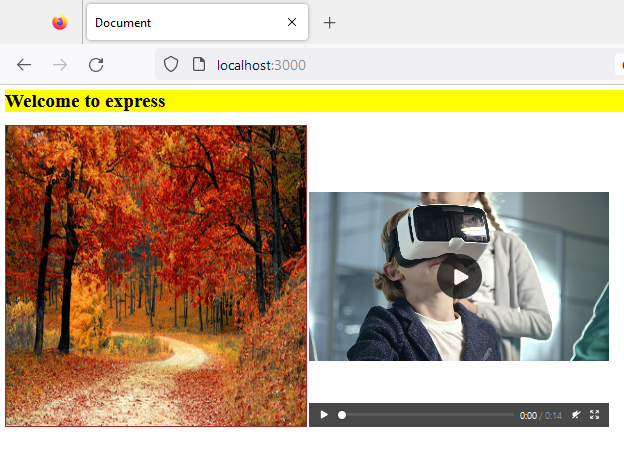
response**.sendFile(path.join(\_\_dirname,"**public/template/index.html**"));**

}**);**

app**.listen(3000,**() => {console.log("Server is Running ....");}**);**

**Run : node app.js**

**OPEN WEB BROWSER 🡺 REQUEST THIS URL : localhost:3000/**

****

# Express.js Request Object

* Express.js **Request and Response** objects are the parameters of the callback function which is used in Express applications.
* The express.js request object represents the HTTP request and has properties for the request **query string, parameters, body, HTTP headers, and so on.**

## Properties

|  |  |
| --- | --- |
| Properties | Description |
| hostname | It contains the hostname from the "host" http header. |
| ip | It specifies the remote IP address of the request. |
| params | An object containing properties mapped to the named route ?parameters?. For example, if you have the route /user/:name, then the "name" property is available as req.params.name. This object defaults to {}. |
| path | It contains the path part of the request URL. |
| protocol | The request protocol string, "http" or "https" when requested with TLS. |
| query | An object containing a property for each query string parameter in the route. |
| route | The currently-matched route, a string. |
| secure | A Boolean that is true if a TLS connection is established. |
| subdomains | It represents an array of subdomains in the domain name of the request. |

**Example :**

**const** express = **require(**"express"**);**

**const** app = express();

app**.get(**"/:id",(request,response) => {

response.end("<h1> Welcome to Express </h1>");

console.log(request**.hostname**);

console.log(request**.ip**);

console.log(request**.query**);

console.log(request**.params**);

console.log(request**.path**);

console.log(request.**protocol**);

console.log(request.**route**);

console.log(request**.secure**);

console.log(request**.subdomains**);

}**);**

//listen express server

app**.listen(**3000,() => {console.log("Server is running ....");}**);**

# Express.js Response Object

The Response object (res) specifies the HTTP response which is sent by an Express app when it gets an **HTTP request**.

|  |  |
| --- | --- |
| Methods | Description |
| json(<jsonObject) | Sends a JSON response. This method sends a response (with the correct content-type) that is the parameter converted to a JSON string using [JSON.stringify()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/stringify). |
| redirect(<url>) | Redirects to the URL derived from the specified path, with specified status, a positive integer that corresponds to an [HTTP status code](http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html) . If not specified, status defaults to “302 “Found”. |
| send(<body>) | Sends the HTTP response.The body parameter can be a Buffer object, a String, an object, Boolean, or an Array. |
| sendFile(<path>) | Transfers the file at the given path. path must be an absolute path to the file. |
| sendStatus(<code>) | Sets the response HTTP status code to statusCode. |
| set(<object>) | Sets the response’s HTTP header field to value. |
| end(<data>) | Ends the response process. This method actually comes from Node core, specifically the [response.end() method of http.ServerResponse](https://nodejs.org/api/http.html#http_response_end_data_encoding_callback). |
| download(<filePath>) | Transfers the file at path as an “attachment”. Typically, browsers will prompt the user for download. |

# HTTP Methods

The primary or most commonly-used HTTP methods are **POST**, **GET**, **PUT**, **PATCH, and DELETE**. These methods correspond to **create**, **read**, **update**, and **delete** **(CRUD)** operations, respectively.

|  |  |  |  |
| --- | --- | --- | --- |
| HTTP Method | CRUD operation | Entire Collection (e.g. /users) | Specific Item (e.g. /users/{id}) |
| GET | Read | 200 (OK), list of entities. Use pagination, sorting and filtering to navigate big lists. | 200 (OK), single entity.  404 (Not Found), if ID not found or invalid. |
| POST | Create | 201 (Created), Response contains response similar to GET /user/{id} containing new ID. | not applicable |
| PATCH | Update | [Batch API](https://doc.oroinc.com/api/batch-api/#web-services-api-batch-api) | 200 (OK) or 204 (No Content).  404 (Not Found), if ID not found or invalid. |
| DELETE | Delete | 204 (No Content).  400(Bad Request) if no filter is specified. | 204 (No Content).  404 (Not Found), if ID not found or invalid. |
| PUT | Create/Replace | not implemented | not implemented |

## GET :

The HTTP GET method is used to ***read* (or retrieve)** a representation of a resource. In case of success (or non-error), GET returns a representation in JSON and an HTTP response status code of **200** (OK). In an error case, it most often returns a **404** (NOT FOUND) or 400 (BAD REQUEST).

## POST :

The POST method is most often utilized to ***create* new resources**. In particular, it is used to create subordinate resources. That is subordinate to some other (e.g. parent) resource. In other words, when creating a new resource, POST to the parent and the service takes care of associating the new resource with the parent, assigning an **ID** (new resource URI), etc.

## PATCH :

## PATCH is used to modify resources. The PATCH request only needs to contain the changes to the resource, not the complete resource.

## DELETE :

DELETE is quite easy to understand. It is used to ***delete* a resource** identified by filters or **ID.**

# REST API Architecture

REST stands for **Representational State Transfer** and API stands for **Application Program Interface**. REST is a software architectural style that defines the set of rules to be used for creating web services. Web services which follow the REST architectural style are known as RESTful web services. It allows requesting systems to access and manipulate web resources by using a uniform and predefined set of rules. Interaction in REST based systems happen through Internet’s **Hypertext Transfer Protocol (HTTP).**

**Rules of REST API:**

There are certain rules which should be kept in mind while creating REST API endpoints.

* REST is based on the resource or noun instead of action or **verb** based. It means that a URI of a REST API should always end with a **noun**.

**Example:** **/api/users**

* HTTP verbs are used to identify the **action**. Some of the HTTP verbs are : **GET, PUT, POST, DELETE, GET, PATCH**.
* A web application should be organized into resources like users and then uses HTTP verbs like : **GET, PUT, POST, DELETE** to **modify those resources**. And as a developer it should be clear that what needs to be done just by looking at the endpoint and HTTP method used.

**Example :**

|  |  |  |
| --- | --- | --- |
| **URI** | **HTTP verb** | **Description** |
| api/users | GET | Get all users |
| api/users | POST | Add a user |
| api/users/1 | PUT | Update a user with id = 1 |
| api/users/1 | DELETE | Delete a user with id = 1 |

**CRUD OPERATION USING REST API**

Create, Read, Update, and Delete (CRUD) are the four basic functions that models should be able to do, at most.

#### CREATE :

To create resources in a REST environment, we most commonly use the HTTP **POST** method. POST creates a new resource of the specified resource type.

**Response: Status Code** **-** **201** (CREATED)

#### READ :

To read resources in a REST environment, we use the **GET** method. Reading a resource should never change any information - it should only retrieve it. If you call GET on the same information 10 times in a row, you should get the same response on the first call that you get on the last call.

**Response: Status Code - 200** (OK)

#### UPDATE :

**PUT** or **PATCH** is the HTTP method used for the CRUD operation, Update.

**Response: Status Code - 200** (OK)

#### DELETE :

The CRUD operation Delete corresponds to the HTTP method **DELETE**. It is used to remove a resource from the system.

**Response: Status Code - 204** (NO CONTENT)

**PROJECT STRUCTURE VIEW :**

**IMPORTENT : POST MAN APPLICATION NEED**

**rootFolder**

**data**

**userRecords.json**

[

{

“id” : 1,

“name” : “Person 1”,

“age” : 20,

“email” : “person1@gmail.com”

},

{

“id” : 2,

“name” : “Person 3”,

“age” : 60,

“email” : “person2@gmail.com”

}

]

**node\_modules**

**express js and other modules**

**app.js**

**FOLDERS :** rootFolder,data,node\_modules.

**FILE :** userRecords.json,app.js.

**FILE : userRecords.json**

**[**

**{**

**"id":1,**

**"name":"Person1",**

**"age":10,**

**"email":"person1@gmail.com",**

**"job":"web dev",**

**"salary":1000**

**},**

**{**

**"id":2,**

**"name":"Person2",**

**"age":20,**

**"email":"person2@gmail.com",**

**"job”: hacker",**

**"salary":2000**

**},**

**{**

**"id":3,**

**"name":"Person3",**

**"age":30,**

**"email":"person3@gmail.com",**

**"job":"hacker",**

**"salary":3000**

**},**

**{**

**"id":4,**

**"name":"Person4",**

**"age":40,**

**"email":"person4@gmail.com",**

**"job":"hacker"**

**,"salary":4000**

**}**

**]**

**FILE : app.js**

**const** express **= require("**express**");**

**const** fs **= require("**fs**");**

**const** app **=** express();

**let** users **= JSON.parse(**fs**.readFileSync(\_\_dirname +** "/data/userRecords.json"**));**

app**.use(**express**.json());**

app**.get("**/api/v1/users**",**(request,response) => {

response**.status(200).json(**{

**"**status**" : "**success**",**

**"**count**" :** users.length**,**

**"**data**" :** {

"users" : **users**

}

}**);**

}**);**

app**.post("**/api/v1/users**",**(request,response) => {

**let** newID **=** users[users**.length** -1]**.id** + 1;

**let** newUser **= Object.assign({**"id" **:** newID**},**request**.body);**

users**.push(**newUser**);**

fs**.writeFile("**./data/userRecords.json**",JSON.stringify(**users**),(**error**) => {**

**if(**error**) throw** error;

response**.status(201).json(**{

"status" : "success",

"data" : {

newUser : **newUser**

}

});

}**);**

}**);**

app**.get("**/api/v1/users/:id **",**(request,response) => {

**const** id **=** request**.params.id;**

**let** user **=** users**.find(**user **=>** user**.id** == **id);**

**if(**user != **null){**

**return** response**.status(200).json(**{

"status" : "success",

"data" : {

"user" : user

}

}**);**

**}**

**else** response**.status(404).json(**{

"status" : "fail",

"message" : `User with ID ${id} is not Found`

}**);**

**});**

app**.patch("**/api/v1/users/:id**",**(request,response) => {

**const** id **=** request**.params.id;**

**let** userToUpdate=users**.find(**user=> user.**id** == **id);**

**if(**userToUpdate== **null){**

**return** response**.status(404).json(**{

"status" : "fail",

"message" : "No User Record with ID " + id + " is not Found"

}**);**

**}**

**let** index **=** users**.indexOf(**userToUpdate**);**

**let** updatedUserRecord **= Object.assign(**userToUpdate**,**request**.body);**

users**[**index**] =** updatedUserRecord;

**fs.writeFile("**./data/userRecords.json**",JSON.stringify(**users**),(**error**) => {**

**if(**error**) throw** error**;**

response**.status(200).json(**{

"status" : "success",

"data" : {

"users" : updatedUserRecord

}

}**);**

}**);**

}**);**

app**.delete("**/api/v1/users/:id**",**(request,response) => {

**let** id **=** request**.params.id;**

**let** deleteUserRecord **=** users**.find(**user **=>** user**.id == id);**

**if(**deleteUserRecord **== null){**

**return** response**.status(404).json({**

"status" : "fail",

"message" : "No User Record with ID " + id + " is not Found"

}**);**

**}**

**let** index **=** users**.indexOf(**deleteUserRecord**);**

users**.splice(**index,1**);**

**fs.writeFile("**./data/userRecords.json**",JSON.stringify(**users**),**(error) => {

**if(**error**) throw** error**;**

response**.status(204).json(**{

"status" : "success",

"data" : {

"users" : null

}

}**);**

}**);**

}**);**

**app.listen(3000,() => {console.log("Server is Running .......")});**

**Run : node app.js**

**OPEN POST MAN APPLICATION SEND REQUEST BELOW URLS WITH BODY**

**READ ALL USER RECORD**

**METHOD : GET**

**REQUEST URL: 127.0.0.1:3000/api/v1/users**

**BODY : NONE**

**RESPONSE : 200 (OK) with (ALL USER RECORDS)**

**GET USER RECORD BY ID**

**METHOD : GET**

**REQUEST URL: 127.0.0.1:3000/api/v1/users/2**

**BODY : NONE**

**RESPONSE : 200 (OK)**

**{**

**status : success,**

**data : {**

**user : {**

**"id":2,**

**"name":"Person2",**

**"age":20,**

**"email":"person2@gmail.com",**

**"job”: hacker",**

**"salary":2000**

**}**

**}**

**}**

**CREATE NEW USER RECORD**

**METHOD : POST**

**REQUEST URL: 127.0.0.1:3000/api/v1/users**

**BODY : {**

**"name":"newPerson",**

**"age":40,**

**"email":"person4@gmail.com",**

**"job":"hacker",**

**"salary":4000**

**}**

**RESPONSE : 200 (OK)**

**{**

**status : success,**

**data : {**

**newUser : {**

**"id":"5",**

**"name":"newPerson",**

**"age":40,**

**"email":"person4@gmail.com",**

**"job":"hacker",**

**"salary":4000**

**}**

**}**

**}**

**UPDATE USER RECORD**

**METHOD : PUT OR PATCH**

**REQUEST URL: 127.0.0.1:3000/api/v1/users/2**

**BODY : { salary : 20000 }**

**RESPONSE : 200 (OK) with (UPDATE USER RECORD)**

**DELETE USER RECORD**

**METHOD : POST**

**REQUEST URL: 127.0.0.1:3000/api/v1/users/2**

**BODY : NONE**

**RESPONSE : 204 (NO CONTENT)**

## Method : app.route(<url>).methods()...

## You can create chainable route handlers for a route path by using app.route(). Because the path is specified at a single location, creating modular routes is helpful, as is reducing redundancy and typos

**(ex)**

**const** express **= require("**express**");**

**const** app **=** express();

**const** getMethod **= function (**request,response**){**

response**.end("**REQUEST METHOD : **"** + request**.method);**

**}**

**const** postMethod **= function (**request,response**){**

response**.end("**REQUEST METHOD: **" +** request**.method)**

**}**

**const** patchMethod **= function (**request,response**){**

response**.end("**REQUEST METHOD : **"** +request.**method)**

**}**

**const** deleteMethod **= function(**request,response**){**

response**.end("**REQUEST METHOD : " +request**.method)**

**}**

**app.route("/")**

**.get(**getMethod**)**

**.post(**postMethod**)**

**.patch(**patchMethod**)**

**.delete(**deleteMethod**);**

app.**listen(3000,**() => {console.log("Server is Running ....")}**);**

**Run : node app.js**

# MIDDLEWARE

Express is a routing and middleware web framework that has minimal functionality of its own: An Express application is essentially a series of **middleware function calls**.

***Middleware*** functions are functions that have access to the [**request object**](https://expressjs.com/en/4x/api.html#req)(request), the [**response object**](https://expressjs.com/en/4x/api.html#res) (response), and the **next middleware** function in the application’s **request-response cycle**. The next middleware function is commonly denoted by a **variable** named next.

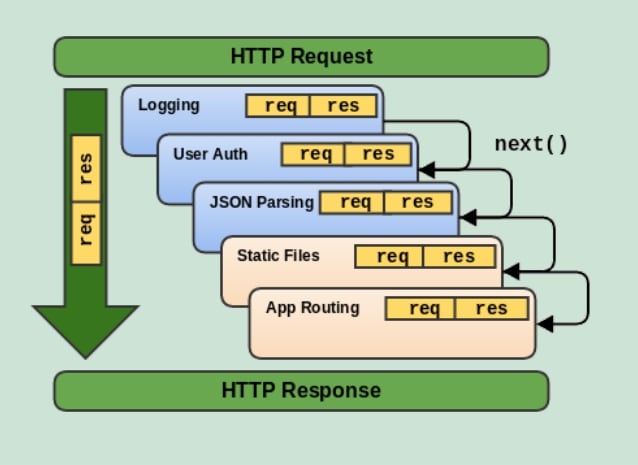
Middleware functions can perform the following tasks:

* 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 **hanging**.

An Express application can use the following types of middleware:

* [Application-level middleware](https://expressjs.com/en/guide/using-middleware.html#middleware.application)
* [Router-level middleware](https://expressjs.com/en/guide/using-middleware.html#middleware.router)
* [Error-handling middleware](https://expressjs.com/en/guide/using-middleware.html#middleware.error-handling)
* [Built-in middleware](https://expressjs.com/en/guide/using-middleware.html#middleware.built-in)
* [Third-party middleware](https://expressjs.com/en/guide/using-middleware.html#middleware.third-party)



**CREATE CUSTOM MIDDLEWARE**

**syntax : function(<request>,<response>,<next>){**

**//some code**

**next();**

**}**

**Example :**

**const express = require("express");**

**const app = express();**

//CREATE MIDDLEWARE FUNCTIONS

**function** myMiddleware**(**request,response,next**){**

console.log("MyMiddleware is called ....");

next();

**}**

**function** sendResponse(request,response){

response**.end(“** work ... **”);**

}

//USE MIDDLEWARE

app**.use(**myMiddleware**);**

app.**route("/").get(**sendResponse**);**

app.**listen(**3000**,() => {**console.log**("**Server is Running ....**")})**

**Run : node app.js**

**USE** [**Third-party middleware**](https://expressjs.com/en/guide/using-middleware.html#middleware.third-party)

# MORGAN : HTTP request logger middleware for node.js

# Installing :

**step 1 : open cmd and type given command**

**(ex) npm install morgan**

**Example :**

**const** express **= require("**express**");**

**const** morgan = **require("**morgan**");**

**const** app **= express();**

//CREATE MIDDLEWARE FUNCTIONS

**function** myMiddleware**(**request,response,next**){**

console.log("MyMiddleware is called ....");

next();

**}**

**function** sendResponse(request,response){

response**.end(“** work ... **”);**

}

//USE MIDDLEWARE

app**.use(**myMiddleware**);**

app**.use(**morgan(“dev”)**);**

app.**route("/").get(**sendResponse**);**

app.**listen(**3000**,() => {**console.log**("**Server is Running ....**")})**

**Run : node app.js**

## express.Router()

Use the express.Router class to create **modular**, mountable route handlers. A Router instance is a complete middleware and routing system; for this reason, it is often referred to as a **“mini-app”.**

**Example :**

**FILE : router.js**

**const** express **= require("**express**");**

**const** app **=** express();

**const** router **=** express.**Router();**

**const** sendHomePage **= function(**request,response**){**

response**.end("**Home Page**");**

**}**

**const** sendAboutPag**e = function(**request,response**){**

response**.end("**About Page**");**

**}**

**const** sendContactPage **= function(**request,response**){**

response**.end("**Contact Page**");**

**}**

**const** sendErrorPage **= function(**request,response**){**

response**.end("**Error Page**");**

**}**

router**.route("**/**").get(**sendHomePage**);**

router**.route("**/about**").get(**sendAboutPage**);**

router**.route("**/contact**").get(**sendContactPage**);**

router**.route("**/error**").get(**sendErrorPage**);**

**module.exports =** router**;**

**FILE : app.js**

**const** express **= require("**express**");**

**const** app **=** express();

**const** router **= require("**./router**");**

app**.use("**/**",**router**);**

app**.use("**/about**",**router**);**

app**.use("**/contact**",**router**);**

app**.use("**/error**",**router**);**

app**.listen("**3000**",**() => { console.log**("**Server is Running ...**"**); **});**

**Run : node app.js**

**OPEN WEB BROWSER 🡺 REQUEST THIS URL : localhost:3000/**

**NEXT 🡺 localhost:3000/about output :** About Page

**NEXT 🡺 localhost:3000/contact output :** Contact Page

**NEXT 🡺 localhost:3000/randomText output :** Error Page

# Upload Files

Multer is a node.js middleware for handling multipart/form-data, which is primarily used for uploading files. It is written on top of [busboy](https://github.com/mscdex/busboy) for maximum efficiency.

**NOTE**: Multer will not process any form which is not multipart (multipart/form-data)

## Installation

**>>** **npm install multer express**

## Usage

Multer adds a body object and a file or files object to the request object. The body object contains the values of the text fields of the form, the file or files object contains the files uploaded via the form.

**PROJECT STRUCTURE VIEW :**

**rootFolder**

**node\_modules**

index.html

app.js

**uploads**

**FILE : index.html**

<**html**>

<**head**> <title>Document </**title**>

</**head**>

<**body**>

<**form** **action**="/upload" **enctype**="multipart/form-data" **method**="POST">

<**h1**> Upload Your File : </**h1**>

<**input** **type**="file" **name** = "myFile"> <**br**><**br**>

<**input** **type**="submit" **value** = "Submit Your File">

</**form**>

</**body**>

</**html**>

**FILE : app.js**

**const** express = **require**("express");

**const** multer = **require**("multer");

**const** app = express();

**let** storage = multer.**diskStorage**({

destination : **function**(request,file,callBack) {

callBack(**null**,"./uploads");

},

filename : **function**(request,file,callBack) {

callBack(**null**,file.originalname);

}

});

**let** upload = **multer**({storage : storage}).**single**("myFile");

app.**get**("/",(request,response) => {

response.sendFile(<your index.html file path>);

});

app.**post**("/upload",(request,response) => {

upload(request,response,(error) => {

if(error) response.send("ERROR ...");

else response.send("File Upload ...");

});

});

app.**listen**("3000",() => {console.log("Server is Running ....");})