**SENDING A FILE AS RESPONSE TO THE SERVER  
 🔹 res.sendFile()**

* A new kind of method for sending responses!
* Used when you want to send an actual **file** (like HTML, PDF, or image) to the client.
* Requires an **absolute path** to the file.(we can use path module for defining absoulute path)

**Example:**

const path = require("path");

app.get("/", (req, res) => {

res.sendFile(path.join(\_\_dirname, "index.html"));

});

👉 Sends the index.html file to the browser.

**⚡ Key Difference:**

* res.send() → send **content/data** (string, object, JSON, etc.)
* res.sendFile() → send an **actual file** from the server’s filesystem

**Question:**  
is it called server side rendering no right as we are sending a file directly to the client to render it on local client

Exactly, **res.sendFile() is *not* server-side rendering (SSR).**

**🔹 Why?**

* With res.sendFile(), Express is just **sending a static file** (like HTML, CSS, JS, images) from the server to the client’s browser.
* The browser itself is responsible for rendering it.
* The server is **not generating or injecting content dynamically** before sending.

This is closer to **static file serving**, not SSR. ✅

**🔹 What is Server-Side Rendering (SSR)?**

* SSR means the server **builds the final HTML (with dynamic data)** and sends it to the client.
* The client just receives a “ready-to-render” HTML page.
* Example: using a templating engine like **EJS, Pug, Handlebars**, or frameworks like **Next.js**.

app.get("/", (req, res) => {

res.render("index", { username: "John" });

});

Here:

* index.ejs is processed on the server.
* username is injected.
* Final HTML is sent to the client → **SSR**.

**🔹 Quick Contrast:**

| **Method** | **What Happens** |
| --- | --- |
| **res.sendFile()** | Sends raw file → Browser renders it locally |
| **SSR (res.render, Next.js, etc.)** | Server builds dynamic HTML → sends complete page |
|  |  |

PDF MERGER – WebApp

**MULTER NPM PACKAGE/MODULE:**Multer is a node.js middleware for handling multipart/form-data, which is primarily used for uploading files.

Installation : npm install mutler

**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.

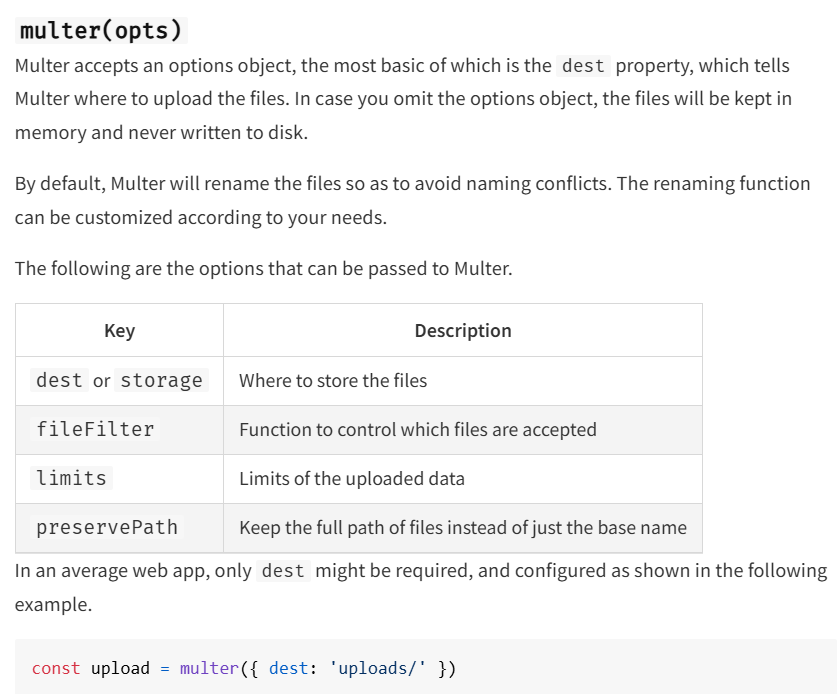
Basic usage example:



**REQ.FILE OBJECT**



**MULTER OPTIONS:**



**DIFFERENT METHODS OF GETTING FILES**

**.single(fieldname)**

Accept a single file with the name fieldname. The single file will be stored in req.file.

**.array(fieldname[, maxCount])**

Accept an array of files, all with the name fieldname. Optionally error out if more than maxCount files are uploaded. The array of files will be stored in req.files.

**.fields(fields)**

Accept a mix of files, specified by fields. An object with arrays of files will be stored in req.files.

fields should be an array of objects with name and optionally a maxCount. Example:

[

{ name: 'avatar', maxCount: 1 },

{ name: 'gallery', maxCount: 8 }

]

**.none()**

Accept only text fields. If any file upload is made, error with code "LIMIT\_UNEXPECTED\_FILE" will be issued.

**.any()**

Accepts all files that comes over the wire. An array of files will be stored in req.files.

**WARNING:** Make sure that you always handle the files that a user uploads. Never add multer as a global middleware since a malicious user could upload files to a route that you didn't anticipate. Only use this function on routes where you are handling the uploaded files.

**IN THIS PROJECT WHILE USING THE LIBRARY(PDF-MERGER-JS) WE USED :  
const PDFmerger = require(“pdf-merger-js”).default;**

* pdf-merger-js is written in **ESM** and uses export default.(PDF Merger is a custom made library not a regular module)
* When you use require() (CommonJS), Node wraps the module, so you get an **object** like:
* { default: [class PDFMerger] }
* That’s why you must use .default to access the class:
* const PDFMerger = require("pdf-merger-js").default;
* If you switch to **ESM** (import/export), you can drop .default.

👉 In short: **.  
default is needed because you’re mixing CommonJS (require) with an ESM default export.  
Regular Node modules (CommonJS)** usually do:

* module.exports = MyClass;

→ so require("module") gives you MyClass directly.

* **ESM modules** often do:
* export default MyClass;

→ when you use require(), Node wraps it into { default: MyClass }.

pdf-merger-js is written in **ESM with export default**, so when you load it with CommonJS (require), you need .default to unwrap the actual class.