File Handling

We will see how the file upload works.

We will create a separate GET route.

```
app.get('/uploadDemo', (req, res) => {

})
```

We will would create uploadDemo.hbs in views\layouts folder. In uploadDemo.hbs we would have the form template inside it. the name attribute of the input tag is the property through which we will be accessing it.

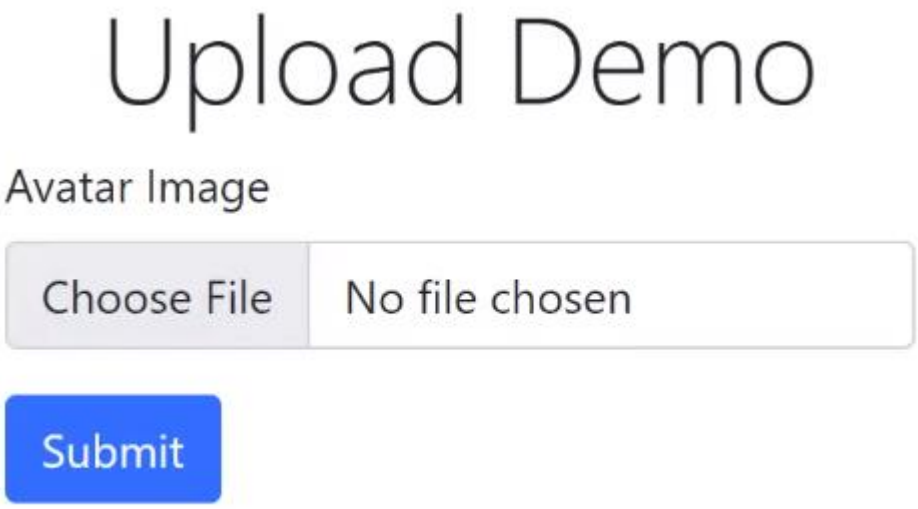
```
<div>
  <h1 class="text-center display-5">Upload Demo</h1>
  <div class="d-flex justify-content-center align-items-center">
    <form>
      <div class="mb-3">
        <label for="avatar" class="form-label">Avatar Image</label>
        <input type="file" name="avatar" class="form-
control" id="avatar">
      </div>
      <button type="submit" class="btn btn-primary">Submit</button>
    </form>
  </div>
</div>
```

So, we will write

```
app.get('/uploadDemo', (req, res) => { // path name
    res.render('uploadDemo') // view name
})
```

Localhost:3000/uploadDemo

Output:



The changes we have made are, changing the attribute type of input tag to **file**. We are having a file type of input. We will change the name attribute of input tag to "testFile" and give id= "demoFile". We have just created a form that goes through the handlebars. Now, we haven't yet defined a functionality. We select any particular file from the local folder.

This styling comes default by bootstrap itself. If we click her, on the choose file button, then we can see a window from which we can select a file to upload. Now when we click on submit, as a default behavior of form, it will create query string and call the GET route. We cannot handle it, so we need to put some extra function. The basic feature is ready. Now we will go through server and have that upload functionality.

We will have to create a specific route to handle all the file upload. All we have is GET route for rendering that part. So, we have to create a path that will handle data when we hit submit button.

Now, in **uploadDemo.hbs** in the form tag, we will add **action="/handleupload" method="POST"**

```
app.post('/handleUpload', (req, res) => {
    res.send({uploaded: true})
})
```

Once we upload the photo, we can

```
{
      uploaded: true
}
```

In the browser. This means post route is working fine and the file name is coming in the terminal. We need to have some additional functionality to handle file uploads. Let's see if we can see what is inside the body of the request. All we have to do is make sure that the middleware, **express.urlencoded** is setup properly.

Behind the scene there is a package called query string which can be either true or false. There are two packages internally that express uses, one is qs and the other is query string. These are third party packages. If we say true it will use qs package and if we say false, it will use query string package.

There is a way to actually do the entire thing. to accept the file as binary, encode, create directory and everything properly, but we need to write a lot of code in NodeJS itself. TO keep simple, we will use third-party package that will ease the development process.

Packages used mostly are,

- Multer
- Filemode
- Expree-fileupload

These are helpful in handling the file upload easily. How to find such packages?

We will search for file handling in node js or express js. The problem is we have to handle file upload. We have install it and inside the post route, that file should be available in the req.file property. It will have some basic functionality, some property, some file and then it goes on to customizing certain parts.

We can put file size limit. We have to install it first.

Inside the index.js, the main server running script, so to import it, we will say,

```
const fileUpload  = require("express-fileupload")

app.use()
```

As it is a middleware, we will use app.use("provide the middleware here"). the file upload package will create the function. When we call this as a function, this fileUpload() returns a function which should be passed into the app.use(). So, to save space,

```
//express to use express-fileupload to handle file upload
app.use(fileUpload())
```

Now, we have to see, how to accept the file. So,

```
const fileUpload  = require("express-fileupload")

//express to use express-fileupload to handle file upload
app.use(fileUpload())

app.post('/handleUpload', (req, res) => {

    console.log(req.body)
    console.log(req.files)
    res.send({uploaded: true})
})
```

When we click on submit, we do get the response, but in the terminal, we see undefined for req.files. the express.urlencoded is a default encoding format of all the form data. Everything gets encoded in a URL specific way. When we use file upload, the urlencoded will not work. It is not an encoding algorithm which can handle file type data.

So, we will have to go back to the form in uploadDemo.hbs, in the form tag we have to add,

```
<form action="/handleUpload" method="POST" enctype="application/x-www-form-urlencoded"
```

By default, the value is enctype="application/x-www-form-urlencoded". This is the algorithm, which browser uses by default to encode the data and then send it to the server. We had added the middleware which goes through the encoding and convert it into a javascript object and attaches to req.body.

It is not mandatory that file will only have textual form. it can be of any time. So, we have to change enctype to **multipart/form-data**. This encoding is a special format for the files which allows a huge data. It can be split into multiple part and is sent in binary format.

```
<form action="/handleUpload" method="POST" enctype="multipart/form-data"
```

When we go to file input, we have to use different file input. We should not get undefined. But the problem persists. The issue is we are using fileUpload after urlencoded. Everything will be coming in URL encoded format. Even for file input, it was using the urlencoded. So, we have to write fileUpload first and then put urlencoded. This fileUpload middleware specifies to handle multiple platform data.

It will not touch the urlencoded. For urlencoded we will use express.urlencoded.

When we click on submit, it shows an empty object logged in the terminal because req.body is empty. The multipart/form-data well be available through req.files. In the terminal we have also logged in testFile as a property which was given in the input tag which in itself is another object. We can see the list of properties. The size is converted to buffer and the total sum of the buffer number would give us the total file size. we have binary data which can be converted into any format. Before it was giving me undefined, now that we have enabled the third-party package of express fileupload to handle it for me, all the file related tags i.e., tags that have type="file" will come under req.files.

Inside the server when we click on submit button, I get the response back in such a way that, textual input came in as req.body, but the files information came inside another object for req.files now we can handle it accordingly.

Based on the two middlewares, fileUpload is taking the file part and putting inside the req.files and the textual information is being handled by the URL encoded.

We need to save the details in one folder instead of RAM to read it again.

1:17:29