TEXT CLASS REVIEW

TEMAS A TRATAR EN LA CUE

- Subida de archivos al servidor utilizando Express
- Uso del paquete express-fileupload o Instalación o Configuración
- Creando una aplicación para subir archivos al servidor
- Validando existencia del archivo
- Validando extensiones permitidas

USO DEL PAQUETE EXPRESS-FILEUPLOAD O INSTALACIÓN O CONFIGURACIÓN

Express-fileupload, es un middleware para la carga de archivos en el servidor con express y Node.js
Instalación:

- Con NPM

```
npm i express-fileupload
```

- Con Yarn

```
yarn add express-fileupload
```

CONFIGURACIÓN DE EXPRESS-FILEUPLOAD

Al usar express-fileupload tenemos una serie de opciones disponibles para personalizar nuestro uso de este paquete. Desde el uso de API como Busboy hasta el uso de un depurador, puedes encontrar una lista completa de configuraciones y opciones aquí.

CREANDO UNA APLICACIÓN PARA SUBIR ARCHIVOS AL SERVIDOR

Instalando las dependencias:

```
$ npm install express express-fileupload
```

Creamos el servidor Express, en el index.js el cual contiene lo siguiente:

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

// Middleware Simple Express para cargar archivos
```

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


const app = express()

// Usamos el Middleware con la opción que pueda crear el path
app.use(fileUpload({
    createParentPath: true
}));

// Creamos la nueva ruta POST /upload
app.post('/upload', async (req, res) => {


    // Utilice el nombre del campo de entrada (es decir, "fileName")
    // para recuperar el archivo cargado
    let fileRecived = req.files.fileName
    console.log(req.files);



    // Utilice el método mv() para colocar el archivo en el directorio
    // a cargar  (es decir, "files")
    fileRecived.mv('./files/${fileRecived.name}', err => {
        if (err) return res.status(500).send({
            message: err
        })

        return res.status(200).send({
            message: 'Archivo Subido al Servidor'
        })
    })
});


app.listen(3000, () => console.log('Corriendo'))
```

VALIDANDO EXISTENCIA DEL ARCHIVO

Para validar si el archivo es nulo debemos verificar el `req.files`, esto es:

```
if (!req.files) {
    res.send({
        status: false,
        message: 'No se cargo el archivo o es nulo'
    });
    else {
```

```
        …
        Se define el método de carga…
    }

    También se puede utilizar la siguiente validación:

        if (!req.files || Object.keys(req.files).length === 0) {
            res.send({
                status: false,
                message: 'Archivo no subido al servidor',
                error: "400"
            });
        });
else {
    …
    Se define el método de carga…
}
```

Es conveniente colocarlo dentro de un control de flujo y manejo de errores, es decir, la declaración **try...catch** señala un bloque de instrucciones a intentar (**try**), y especifica una respuesta si se produce una excepción (**catch**).

El código anterior para la carga de un archivo dentro de un **try...catch** quedaría de la siguiente manera:

```
app.post('/upload', async (req, res) => {
    // Manejo de un control de flujo y manejo de errores  try--ca
    try {
        //Verificación de la existencia del Archivo no nulo
        if (!req.files || Object.keys(req.files).length === 0) {
            res.send({
                status: false,
                message: 'Archivo no subido al servidor',
                error: "400"
            });
        } else {
            let fileRecived = req.files.fileName;
            fileRecived.mv('./files/${fileRecived.name}', err => {
                if (err) {
                    return res.status(500).send({
                        message: err
                    });
                }
                return res.status(200).send({
                    message: 'Archivo Subido al Servidor'
                });
```

```
            });
        }
    } catch (err) {
        res.status(500).send(err);
    }
});
```

VALIDANDO EXTENSIONES PERMITIDAS:

Para evaluar y restringir el tipo de archivo según su extensión o tipo de parámetro mimetype que está en la siguiente estructura, siendo el `req.files.foo` el objeto de tipo archivo en `express-fileupload` :

`req.files.foo.mimetype` : que nos devuelve el tipo del archivos

Por ejemplo, si solo deseamos subir y guardar en nuestro servidor archivos del tipo texto lo validamos de la siguiente manera:

```
…

let fileRecived = req.files.fileName;

if (fileRecived.mimetype === "text/plain") {
    …
    //Estructura del código de Guardado en el servidor


}
```

Algunos tipos de archivos que nos devuelve el parámetro `mimetype` :

Tipo de archivo	mimetype
.odt	“application/vnd.oasis.opendocument.text”
.docx	“application/vnd.openxmlformats-officedocument.wordprocessingml.document”
.png	“'image/png”
.jpeg	“image/jpeg”
.txt	“text/plain”
.zip	“application/zip”

La otra manera es validar según su extensión, por ejemplo .jpg, ,png o ,jpeg, por ejemplo. Para este caso colocamos en una variable los formatos permitidos y luego procedemos a validar por extensión, es:

```
…
if (!req.files || Object.keys(req.files).length === 0) {

    //Código de archivo vacío
} else {
    let fileRecived = req.files.fileName;
    let extName = fileRecived.name;
    let allowedExtension = ['.png', 'jpg', 'jpeg'];
    uploadPath = './files/' + extName;

    if (!allowedExtension.includes(extName)) {

        //Código para cargar el archivo en el servidor

    }
}
```