📁Uploading Multiple Files with Metadata in React and Node.js (Multer)

Язык оригинала: en

# Оригинал

👋 Hey there, welcome!  
Today, we’re diving into a very practical and essential feature in modern web development:  
📂 Uploading multiple files along with metadata (text inputs).  
This is a common requirement across many applications — from job portals and admin dashboards to content platforms. In this blog, I’ll guide you step-by-step on how to implement this feature using:  
🔧 React (Frontend) — for collecting files and text inputs  
🚀 Node.js with Express + Multer (Backend) — to handle and store the uploaded data  
You'll see how to accept user inputs like "employment type" and "salary range" along with multiple file uploads, all in one request.  
🚀 What You’ll Build  
A form where the user can:  
Upload up to  
5  
.pdf  
,  
.doc  
, or  
.docx  
files  
Fill in metadata like  
employmentType  
,  
salaryRange  
, etc.  
Submit everything to a backend that handles both file storage and metadata parsing  
🧪 Frontend in React: Uploading Files + Metadata  
We’ll use React Hook Form and  
FormData  
to handle file + text data submission.  
📌 Form Inputs: Text + File Upload  
<  
div  
className  
=  
"  
col-span-1 space-y-2  
"  
>  
<  
Label  
htmlFor  
=  
"  
employmentType  
"  
>  
Employment  
Type  
(  
Optional  
)  
<  
/Label  
>  
<  
Input  
id  
=  
"  
employmentType  
"  
type  
=  
"  
text  
"  
placeholder  
=  
"  
Enter here..  
"  
{...  
register  
(  
"  
employmentType  
"  
)}  
/  
>  
<  
/div  
>  
<  
div  
className  
=  
"  
col-span-1 space-y-2  
"  
>  
<  
Label  
htmlFor  
=  
"  
salaryRange  
"  
>  
Salary  
Range  
(  
Optional  
)  
<  
/Label  
>  
<  
Input  
id  
=  
"  
salaryRange  
"  
type  
=  
"  
text  
"  
placeholder  
=  
"  
Enter here..  
"  
{...  
register  
(  
"  
salaryRange  
"  
)}  
/  
>  
<  
/div  
>  
<  
div  
className  
=  
"  
col-span-1 space-y-2  
"  
>  
<  
Label  
htmlFor  
=  
"  
sampleAds  
"  
>  
Upload  
Sample  
Ads  
<  
/Label  
>  
<  
Input  
id  
=  
"  
sampleAds  
"  
className  
=  
"  
cursor-pointer  
"  
placeholder  
=  
"  
select here..  
"  
type  
=  
"  
file  
"  
multiple  
onChange  
=  
{  
handleFileUpload  
}  
accept  
=  
{  
ACCEPTED\_FILES  
}  
// ".pdf, .doc, .docx"  
/  
>  
<  
/div  
>  
Enter fullscreen mode  
Exit fullscreen mode  
🧠 Preparing FormData with Text + Files  
const  
formData  
=  
new  
FormData  
();  
formData  
.  
append  
(  
"  
name  
"  
,  
name  
.  
trim  
());  
formData  
.  
append  
(  
"  
about  
"  
,  
about  
.  
trim  
());  
formData  
.  
append  
(  
"  
jobLocation  
"  
,  
jobLocation  
);  
formData  
.  
append  
(  
"  
employmentType  
"  
,  
employmentType  
);  
formData  
.  
append  
(  
"  
salaryRange  
"  
,  
salaryRange  
);  
formData  
.  
append  
(  
"  
admins  
"  
,  
JSON  
.  
stringify  
(  
adminsEmail  
));  
formData  
.  
append  
(  
"  
inviteAdmins  
"  
,  
JSON  
.  
stringify  
(  
emails  
));  
formData  
.  
append  
(  
"  
twilioNumbers  
"  
,  
JSON  
.  
stringify  
(  
twilioNumbers  
.  
map  
((  
num  
)  
=>  
num  
.  
value  
)));  
uploadedFiles  
?.  
forEach  
((  
file  
)  
=>  
formData  
.  
append  
(  
"  
files  
"  
,  
file  
));  
Enter fullscreen mode  
Exit fullscreen mode  
🔑  
Tip:  
Always  
JSON.stringify()  
any arrays or objects you append to  
FormData  
.  
🚀 Submitting FormData via Axios (React Query)  
const  
useCreateCompany  
=  
useMutation  
({  
mutationFn  
:  
async  
(  
formData  
:  
FormData  
)  
=>  
{  
const  
response  
=  
await  
apiClient  
.  
post  
(  
`  
${  
API\_COMPANY\_URL  
}  
`  
,  
formData  
,  
{  
headers  
:  
{  
"  
Content-Type  
"  
:  
"  
multipart/form-data  
"  
,  
},  
});  
return  
response  
.  
data  
;  
},  
});  
Enter fullscreen mode  
Exit fullscreen mode  
📤 Backend in Node.js: Express + Multer  
To handle  
multipart/form-data  
, we’ll use the  
multer  
package.  
🧰 Setting Up Multer with Disk Storage  
import  
multer  
from  
"  
multer  
"  
;  
import  
\*  
as  
fs  
from  
"  
node:fs  
"  
;  
import  
\*  
as  
path  
from  
"  
node:path  
"  
;  
// Create an upload folder if it doesn't exist  
const  
uploadDir  
=  
path  
.  
join  
(  
process  
.  
cwd  
(),  
"  
upload  
"  
);  
if  
(  
!  
fs  
.  
existsSync  
(  
uploadDir  
))  
{  
fs  
.  
mkdirSync  
(  
uploadDir  
);  
}  
// Disk storage config  
const  
diskStorage  
=  
multer  
.  
diskStorage  
({  
destination  
:  
(  
req  
,  
file  
,  
cb  
)  
=>  
cb  
(  
null  
,  
uploadDir  
),  
filename  
:  
(  
req  
,  
file  
,  
cb  
)  
=>  
cb  
(  
null  
,  
file  
.  
originalname  
),  
});  
Enter fullscreen mode  
Exit fullscreen mode  
✅ File Filter (Allow Only .pdf, .doc, .docx)  
const  
fileFilter  
=  
(  
req  
,  
file  
,  
cb  
)  
=>  
{  
const  
allowedMimeTypes  
=  
[  
"  
application/msword  
"  
,  
"  
application/pdf  
"  
,  
"  
application/vnd.openxmlformats-officedocument.wordprocessingml.document  
"  
,  
];  
allowedMimeTypes  
.  
includes  
(  
file  
.  
mimetype  
)  
?  
cb  
(  
null  
,  
true  
)  
:  
cb  
(  
new  
Error  
(  
"  
Invalid file type. Only doc and PDFs are allowed.  
"  
));  
};  
Enter fullscreen mode  
Exit fullscreen mode  
🧩 Final Multer Config  
export  
const  
upload  
=  
multer  
({  
storage  
:  
diskStorage  
,  
limits  
:  
{  
fileSize  
:  
5  
\*  
1024  
\*  
1024  
},  
// Max 5MB  
fileFilter  
:  
fileFilter  
,  
}).  
array  
(  
"  
files  
"  
,  
5  
);  
// Max 5 files allowed  
Enter fullscreen mode  
Exit fullscreen mode  
📬 Route Setup with Middleware  
this  
.  
router  
.  
post  
(  
"  
/  
"  
,  
upload  
,  
CompanyRoutes  
.  
createCompany  
);  
Enter fullscreen mode  
Exit fullscreen mode  
📦 Reading Metadata + Files in Controller  
const  
{  
name  
,  
about  
,  
jobLocation  
,  
employmentType  
,  
salaryRange  
}  
=  
req  
.  
body  
;  
const  
admins  
=  
JSON  
.  
parse  
(  
req  
.  
body  
.  
admins  
);  
const  
inviteAdmins  
=  
JSON  
.  
parse  
(  
req  
.  
body  
.  
inviteAdmins  
);  
const  
twilioNumbers  
=  
JSON  
.  
parse  
(  
req  
.  
body  
.  
twilioNumbers  
);  
const  
files  
=  
req  
.  
files  
as  
Express  
.  
Multer  
.  
File  
[];  
if  
(  
files  
&&  
files  
.  
length  
>  
5  
)  
{  
return  
ErrorResponse  
(  
res  
,  
422  
,  
{  
message  
:  
"  
Not more than 5 files are allowed  
"  
,  
});  
}  
Enter fullscreen mode  
Exit fullscreen mode  
🧽 Bonus: Delete Uploaded Files After Processing  
If you're using  
disk storage  
, files are saved on your server. After processing (like uploading to a cloud or OpenAI), you should delete them to avoid clutter.  
import  
\*  
as  
fs  
from  
"  
fs  
"  
;  
import  
\*  
as  
path  
from  
"  
path  
"  
;  
// Delete files after processing  
files  
.  
forEach  
((  
file  
)  
=>  
{  
const  
filePath  
=  
path  
.  
join  
(  
process  
.  
cwd  
(),  
"  
upload  
"  
,  
file  
.  
filename  
);  
fs  
.  
unlink  
(  
filePath  
,  
(  
err  
)  
=>  
{  
if  
(  
err  
)  
{  
console  
.  
error  
(  
`Failed to delete file  
${  
file  
.  
filename  
}  
:`  
,  
err  
);  
}  
else  
{  
console  
.  
log  
(  
`Deleted file:  
${  
file  
.  
filename  
}  
`  
);  
}  
});  
});  
Enter fullscreen mode  
Exit fullscreen mode  
🏁 Conclusion  
With this full-stack setup, you can reliably handle both  
multiple file uploads  
and  
structured metadata  
in your apps. Whether you're building a form for job ads, templates, or resumes — this pattern is scalable, secure, and easy to maintain.  
💡 Pro Tip: If you're planning to use cloud storage (e.g., AWS S3, GCS, or OpenAI), process the file in-memory or store temporarily, then delete the local copy to save space.

# Перевод на русский

👋 Hey there, welcome!  
Today, we’re diving into a very practical and essential feature in modern web development:  
📂 Uploading multiple files along with metadata (text inputs).  
This is a common requirement across many applications — from job portals and admin dashboards to content platforms. In this blog, I’ll guide you step-by-step on how to implement this feature using:  
🔧 React (Frontend) — for collecting files and text inputs  
🚀 Node.js with Express + Multer (Backend) — to handle and store the uploaded data  
You'll see how to accept user inputs like "employment type" and "salary range" along with multiple file uploads, all in one request.  
🚀 What You’ll Build  
A form where the user can:  
Upload up to  
5  
.pdf  
,  
.doc  
, or  
.docx  
files  
Fill in metadata like  
employmentType  
,  
salaryRange  
, etc.  
Submit everything to a backend that handles both file storage and metadata parsing  
🧪 Frontend in React: Uploading Files + Metadata  
We’ll use React Hook Form and  
FormData  
to handle file + text data submission.  
📌 Form Inputs: Text + File Upload  
<  
div  
className  
=  
"  
col-span-1 space-y-2  
"  
>  
<  
Label  
htmlFor  
=  
"  
employmentType  
"  
>  
Employment  
Type  
(  
Optional  
)  
<  
/Label  
>  
<  
Input  
id  
=  
"  
employmentType  
"  
type  
=  
"  
text  
"  
placeholder  
=  
"  
Enter here..  
"  
{...  
register  
(  
"  
employmentType  
"  
)}  
/  
>  
<  
/div  
>  
<  
div  
className  
=  
"  
col-span-1 space-y-2  
"  
>  
<  
Label  
htmlFor  
=  
"  
salaryRange  
"  
>  
Salary  
Range  
(  
Optional  
)  
<  
/Label  
>  
<  
Input  
id  
=  
"  
salaryRange  
"  
type  
=  
"  
text  
"  
placeholder  
=  
"  
Enter here..  
"  
{...  
register  
(  
"  
salaryRange  
"  
)}  
/  
>  
<  
/div  
>  
<  
div  
className  
=  
"  
col-span-1 space-y-2  
"  
>  
<  
Label  
htmlFor  
=  
"  
sampleAds  
"  
>  
Upload  
Sample  
Ads  
<  
/Label  
>  
<  
Input  
id  
=  
"  
sampleAds  
"  
className  
=  
"  
cursor-pointer  
"  
placeholder  
=  
"  
select here..  
"  
type  
=  
"  
file  
"  
multiple  
onChange  
=  
{  
handleFileUpload  
}  
accept  
=  
{  
ACCEPTED\_FILES  
}  
// ".pdf, .doc, .docx"  
/  
>  
<  
/div  
>  
Enter fullscreen mode  
Exit fullscreen mode  
🧠 Preparing FormData with Text + Files  
const  
formData  
=  
new  
FormData  
();  
formData  
.  
append  
(  
"  
name  
"  
,  
name  
.  
trim  
());  
formData  
.  
append  
(  
"  
about  
"  
,  
about  
.  
trim  
());  
formData  
.  
append  
(  
"  
jobLocation  
"  
,  
jobLocation  
);  
formData  
.  
append  
(  
"  
employmentType  
"  
,  
employmentType  
);  
formData  
.  
append  
(  
"  
salaryRange  
"  
,  
salaryRange  
);  
formData  
.  
append  
(  
"  
admins  
"  
,  
JSON  
.  
stringify  
(  
adminsEmail  
));  
formData  
.  
append  
(  
"  
inviteAdmins  
"  
,  
JSON  
.  
stringify  
(  
emails  
));  
formData  
.  
append  
(  
"  
twilioNumbers  
"  
,  
JSON  
.  
stringify  
(  
twilioNumbers  
.  
map  
((  
num  
)  
=>  
num  
.  
value  
)));  
uploadedFiles  
?.  
forEach  
((  
file  
)  
=>  
formData  
.  
append  
(  
"  
files  
"  
,  
file  
));  
Enter fullscreen mode  
Exit fullscreen mode  
🔑  
Tip:  
Always  
JSON.stringify()  
any arrays or objects you append to  
FormData  
.  
🚀 Submitting FormData via Axios (React Query)  
const  
useCreateCompany  
=  
useMutation  
({  
mutationFn  
:  
async  
(  
formData  
:  
FormData  
)  
=>  
{  
const  
response  
=  
await  
apiClient  
.  
post  
(  
`  
${  
API\_COMPANY\_URL  
}  
`  
,  
formData  
,  
{  
headers  
:  
{  
"  
Content-Type  
"  
:  
"  
multipart/form-data  
"  
,  
},  
});  
return  
response  
.  
data  
;  
},  
});  
Enter fullscreen mode  
Exit fullscreen mode  
📤 Backend in Node.js: Express + Multer  
To handle  
multipart/form-data  
, we’ll use the  
multer  
package.  
🧰 Setting Up Multer with Disk Storage  
import  
multer  
from  
"  
multer  
"  
;  
import  
\*  
as  
fs  
from  
"  
node:fs  
"  
;  
import  
\*  
as  
path  
from  
"  
node:path  
"  
;  
// Create an upload folder if it doesn't exist  
const  
uploadDir  
=  
path  
.  
join  
(  
process  
.  
cwd  
(),  
"  
upload  
"  
);  
if  
(  
!  
fs  
.  
existsSync  
(  
uploadDir  
))  
{  
fs  
.  
mkdirSync  
(  
uploadDir  
);  
}  
// Disk storage config  
const  
diskStorage  
=  
multer  
.  
diskStorage  
({  
destination  
:  
(  
req  
,  
file  
,  
cb  
)  
=>  
cb  
(  
null  
,  
uploadDir  
),  
filename  
:  
(  
req  
,  
file  
,  
cb  
)  
=>  
cb  
(  
null  
,  
file  
.  
originalname  
),  
});  
Enter fullscreen mode  
Exit fullscreen mode  
✅ File Filter (Allow Only .pdf, .doc, .docx)  
const  
fileFilter  
=  
(  
req  
,  
file  
,  
cb  
)  
=>  
{  
const  
allowedMimeTypes  
=  
[  
"  
application/msword  
"  
,  
"  
application/pdf  
"  
,  
"  
application/vnd.openxmlformats-officedocument.wordprocessingml.document  
"  
,  
];  
allowedMimeTypes  
.  
includes  
(  
file  
.  
mimetype  
)  
?  
cb  
(  
null  
,  
true  
)  
:  
cb  
(  
new  
Error  
(  
"  
Invalid file type. Only doc and PDFs are allowed.  
"  
));  
};  
Enter fullscreen mode  
Exit fullscreen mode  
🧩 Final Multer Config  
export  
const  
upload  
=  
multer  
({  
storage  
:  
diskStorage  
,  
limits  
:  
{  
fileSize  
:  
5  
\*  
1024  
\*  
1024  
},  
// Max 5MB  
fileFilter  
:  
fileFilter  
,  
}).  
array  
(  
"  
files  
"  
,  
5  
);  
// Max 5 files allowed  
Enter fullscreen mode  
Exit fullscreen mode  
📬 Route Setup with Middleware  
this  
.  
router  
.  
post  
(  
"  
/  
"  
,  
upload  
,  
CompanyRoutes  
.  
createCompany  
);  
Enter fullscreen mode  
Exit fullscreen mode  
📦 Reading Metadata + Files in Controller  
const  
{  
name  
,  
about  
,  
jobLocation  
,  
employmentType  
,  
salaryRange  
}  
=  
req  
.  
body  
;  
const  
admins  
=  
JSON  
.  
parse  
(  
req  
.  
body  
.  
admins  
);  
const  
inviteAdmins  
=  
JSON  
.  
parse  
(  
req  
.  
body  
.  
inviteAdmins  
);  
const  
twilioNumbers  
=  
JSON  
.  
parse  
(  
req  
.  
body  
.  
twilioNumbers  
);  
const  
files  
=  
req  
.  
files  
as  
Express  
.  
Multer  
.  
File  
[];  
if  
(  
files  
&&  
files  
.  
length  
>  
5  
)  
{  
return  
ErrorResponse  
(  
res  
,  
422  
,  
{  
message  
:  
"  
Not more than 5 files are allowed  
"  
,  
});  
}  
Enter fullscreen mode  
Exit fullscreen mode  
🧽 Bonus: Delete Uploaded Files After Processing  
If you're using  
disk storage  
, files are saved on your server. After processing (like uploading to a cloud or OpenAI), you should delete them to avoid clutter.  
import  
\*  
as  
fs  
from  
"  
fs  
"  
;  
import  
\*  
as  
path  
from  
"  
path  
"  
;  
// Delete files after processing  
files  
.  
forEach  
((  
file  
)  
=>  
{  
const  
filePath  
=  
path  
.  
join  
(  
process  
.  
cwd  
(),  
"  
upload  
"  
,  
file  
.  
filename  
);  
fs  
.  
unlink  
(  
filePath  
,  
(  
err  
)  
=>  
{  
if  
(  
err  
)  
{  
console  
.  
error  
(  
`Failed to delete file  
${  
file  
.  
filename  
}  
:`  
,  
err  
);  
}  
else  
{  
console  
.  
log  
(  
`Deleted file:  
${  
file  
.  
filename  
}  
`  
);  
}  
});  
});  
Enter fullscreen mode  
Exit fullscreen mode  
🏁 Conclusion  
With this full-stack setup, you can reliably handle both  
multiple file uploads  
and  
structured metadata  
in your apps. Whether you're building a form for job ads, templates, or resumes — this pattern is scalable, secure, and easy to maintain.  
💡 Pro Tip: If you're planning to use cloud storage (e.g., AWS S3, GCS, or OpenAI), process the file in-memory or store temporarily, then delete the local copy to save space.