To store PDF files in MongoDB in a **MERN stack** application (MongoDB, Express.js, React, Node.js), the recommended way is using **GridFS** via Node.js. Below is a **complete setup guide** for handling PDF uploads and storing them in MongoDB.

□ Overview

- **Frontend (React)**: Upload a PDF file via a form
- **Backend (Node.js + Express)**: Use `multer` and `gridfs-stream` to upload the file to MongoDB (GridFS)
- **MongoDB**: Stores the PDF in `fs.files` and `fs.chunks` collections

□ Backend Setup

1. Install Required Packages

```
npm install express mongoose multer multer-gridfs-storage gridfs-stream
```

---

2. Express Server with GridFS

```
// server.js
const express = require('express');
const mongoose = require('mongoose');
const multer = require('multer');
const { GridFsStorage } = require('multer-gridfs-storage');
const Grid = require('gridfs-stream');
const crypto = require('crypto');
const path = require('path');
const cors = require('cors');

const app = express();
app.use(cors());

// Connect to MongoDB
const mongoURI = 'mongodb://localhost:27017/yourDB';
const conn = mongoose.createConnection(mongoURI, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});

let gfs;
conn.once('open', () => {
  gfs = Grid(conn.db, mongoose.mongo);
  gfs.collection('uploads'); // collection name
});

// Create storage engine
```

```
const storage = new GridFsStorage({
  url: mongoURI,
  file: (req, file) => {
    return new Promise((resolve, reject) => {
      crypto.randomBytes(16, (err, buf) => {
        if (err) return reject(err);
        const filename = buf.toString('hex') + path.extname(file.originalname);
        const fileInfo = {
          filename: filename,
          bucketName: 'uploads', // should match collection name above
        };
        resolve(fileInfo);
      });
    });
  },
});

const upload = multer({ storage });

// @route POST /upload
app.post('/upload', upload.single('file'), (req, res) => {
  res.json({ file: req.file });
});

// @route GET /file/:filename
app.get('/file/:filename', (req, res) => {
  gfs.files.findOne({ filename: req.params.filename }, (err, file) => {
    if (!file || file.length === 0) return res.status(404).json({ err: 'No file
exists' });

    // Check if it is a PDF
    if (file.contentType === 'application/pdf') {
      const readstream = gfs.createReadStream(file.filename);
      readstream.pipe(res);
    } else {
      res.status(400).json({ err: 'Not a PDF file' });
    }
  });
});

const PORT = 5000;
app.listen(PORT, () => console.log(`Server started on port ${PORT}`));
```

□ Frontend (React) – Upload PDF

```
// UploadComponent.js
import React, { useState } from 'react';
import axios from 'axios';

const UploadComponent = () => {
  const [file, setFile] = useState(null);

  const onFileChange = e => setFile(e.target.files[0]);

  const onSubmit = async e => {
    e.preventDefault();
```

```
    const formData = new FormData();
    formData.append('file', file);
    try {
      const res = await axios.post('http://localhost:5000/upload', formData, {
        headers: { 'Content-Type': 'multipart/form-data' },
      });
      console.log('File uploaded:', res.data);
    } catch (err) {
      console.error('Upload failed:', err);
    }
  };

  return (
    <form onSubmit={onSubmit}>
      <input type="file" accept="application/pdf" onChange={onFileChange} />
      <button type="submit">Upload PDF</button>
    </form>
  );
};

export default UploadComponent;
```

□ Result in MongoDB

The uploaded PDF is split into chunks and stored in:

- `uploads.files`
- `uploads.chunks`

You can preview and download it using `/file/:filename`.

✅Tips

- For production, secure uploads (file size limit, type validation).
- To stream PDFs inline in the browser, use `<iframe src="/file/:filename">`.
- You can add metadata to files using GridFS's options.

Great! Here's how you can extend your **MERN PDF upload system** to support:

1. ✅**Listing uploaded PDFs**
2. □ **Deleting a specific file**

□ BACKEND UPDATES (Node.js / Express)

Assuming you're using the same Express server as above:

✅1. List All Uploaded PDFs

```
// @route GET /files
app.get('/files', (req, res) => {
  gfs.files.find().toArray((err, files) => {
    if (!files || files.length === 0) {
      return res.status(404).json({ message: 'No files found' });
    }

    // Filter for PDFs only
    const pdfFiles = files.filter(file => file.contentType === 'application/pdf');
    return res.json(pdfFiles);
  });
});
```

□ 2. Delete a File by ID

```
// @route DELETE /file/:id
const { ObjectId } = mongoose.Types;

app.delete('/file/:id', (req, res) => {
  gfs.remove({ _id: ObjectId(req.params.id), root: 'uploads' }, (err) => {
    if (err) return res.status(500).json({ err });
    res.json({ message: 'File deleted successfully' });
  });
});
```

□ FRONTEND (React)

✅Display PDF List with Delete Button

```
// FileListComponent.js
import React, { useEffect, useState } from 'react';
import axios from 'axios';

const FileListComponent = () => {
  const [files, setFiles] = useState([]);

  const fetchFiles = async () => {
    const res = await axios.get('http://localhost:5000/files');
    setFiles(res.data);
  };

  const deleteFile = async (id) => {
    if (window.confirm('Are you sure you want to delete this file?')) {
      await axios.delete(`http://localhost:5000/file/${id}`);
```

```
      fetchFiles(); // Refresh list
    }
  };

  useEffect(() => {
    fetchFiles();
  }, []);

  return (
    <div>
      <h2>Uploaded PDFs</h2>
      <ul>
        {files.map((file) => (
          <li key={file._id}>
            <a href={`http://localhost:5000/file/${file.filename}`} target="_blank"
rel="noreferrer">
              {file.filename}
            </a>
             
            <button onClick={() => deleteFile(file._id)}>Delete</button>
          </li>
        ))}
      </ul>
    </div>
  );
};

export default FileListComponent;
```

□ Final Integration in React App

```
// App.js
import React from 'react';
import UploadComponent from './UploadComponent';
import FileListComponent from './FileListComponent';

function App() {
  return (
    <div>
      <h1>PDF Upload & Management</h1>
      <UploadComponent />
      <FileListComponent />
    </div>
  );
}

export default App;
```

□ Summary

Feature	Route/Component
Upload PDF	`POST /upload` + `UploadComponent.js`

Feature	Route/Component
List PDFs	`GET /files` + `FileListComponent.js`
View PDF	`GET /file/:filename`
Delete PDF	`DELETE /file/:id`

Let me know if you want:

- □ File size validation
- □ Authentication for uploads/deletion
- □ Upload folders or categories