React PDF Implementation Guide - The Simplest Way

Overview

This guide shows how to implement `react-pdf` in the **simplest way possible** in a Next.js application. No complex setup, no unnecessary configurations - just a working PDF viewer with minimal code.

Why This Approach?

After trying multiple complex implementations with API routes, context providers, and worker configurations that caused endless errors, this simple approach **actually works** without version conflicts or setup headaches.

Prerequisites

- Next.js application (tested with Next.js 15.4.1)
- React 19+
- Basic understanding of React hooks

Step-by-Step Implementation

Step 1: Install Dependencies

```
npm install react-pdf
```

Important: Only install `react-pdf`. Do NOT install `pdfjs-dist` separately as it will cause version conflicts.

Step 2: Basic Worker Configuration

In your component file, add this simple worker setup:

```
import { Document, Page, pdfjs } from 'react-pdf';
import 'react-pdf/dist/Page/AnnotationLayer.css';
import 'react-pdf/dist/Page/TextLayer.css';

// Simple worker configuration that actually works
pdfjs.GlobalWorkerOptions.workerSrc = `//unpkg.com/pdfjs-
dist@${pdfjs.version}/build/pdf.worker.min.mjs`;
```

Step 3: PDF Options (Optional but Recommended)

```
const pdfOptions = {
  cMapUrl: '/cmaps/',
  cMapPacked: true,
  standardFontDataUrl: '/standard_fonts/',
};
```

Step 4: Simple PDF Viewer Component

```
"use client";
import { useState } from "react";
import { Document, Page, pdfjs } from 'react-pdf';
import 'react-pdf/dist/Page/AnnotationLayer.css';
import 'react-pdf/dist/Page/TextLayer.css';

// Worker setup
pdfjs.GlobalWorkerOptions.workerSrc = `//unpkg.com/pdfjs-
dist@${pdfjs.version}/build/pdf.worker.min.mjs`;

export const SimplePDFViewer = ({ pdfUrl, title }) => {
  const [showPdf, setShowPdf] = useState(false);
  const [numPages, setNumPages] = useState(null);
  const [pageNumber, setPageNumber] = useState(1);
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState(null);

  function onDocumentLoadSuccess({ numPages }) {
    setNumPages(numPages);
    setLoading(false);
    setError(null);
  }

  function onDocumentLoadError(error) {
    setError(`Failed to load PDF: ${error.message}`);
    setLoading(false);
  }

  const handleViewPdf = () => {
    setShowPdf(true);
    setLoading(true);
    setError(null);
    setPageNumber(1);
  };

  return (
    <>
      {/* Simple trigger button */}
      <button onClick={handleViewPdf} className="bg-blue-600 text-white px-4 py-2
rounded">
        View PDF
      </button>

      {/* Simple modal */}
      {showPdf && (
        <div className="fixed inset-0 bg-black/50 flex items-center justify-center
```

```
z-50 p-4">
          <div className="bg-white rounded-lg w-full max-w-4xl h-full max-h-[90vh]
flex flex-col">

            {/* Header */}
            <div className="flex justify-between items-center p-4 border-b">
              <h2 className="text-xl font-semibold">{title}</h2>
              <button onClick={() => setShowPdf(false)} className="text-
2xl">×</button>
            </div>

            {/* PDF Content */}
            <div className="flex-1 overflow-auto p-4">
              {loading && <div className="text-center">Loading PDF...</div>}
              {error && <div className="text-center text-red-600">{error}</div>}

              {!error && (
                <div className="flex justify-center">
                  <Document
                    file={pdfUrl}
                    onLoadSuccess={onDocumentLoadSuccess}
                    onLoadError={onDocumentLoadError}
                  >
                    {numPages && (
                      <Page
                        pageNumber={pageNumber}
                        width={Math.min(800, window.innerWidth - 100)}
                      />
                    )}
                  </Document>
                </div>
              )}
            </div>

            {/* Navigation */}
            {numPages && (
              <div className="flex justify-between items-center p-4 border-t">
                <button
                  onClick={() => setPageNumber(p => Math.max(p - 1, 1))}
                  disabled={pageNumber <= 1}
                  className="px-4 py-2 bg-blue-600 text-white rounded disabled:bg-
gray-400"
                >
                  Previous
                </button>
                <span>Page {pageNumber} of {numPages}</span>
                <button
                  onClick={() => setPageNumber(p => Math.min(p + 1, numPages))}
                  disabled={pageNumber >= numPages}
                  className="px-4 py-2 bg-blue-600 text-white rounded disabled:bg-
gray-400"
                >
                  Next
                </button>
              </div>
            )}
          </div>
        </div>
      )}
    </>
  );
};
```

Step 5: Usage

```
// In your page or component
import { SimplePDFViewer } from './SimplePDFViewer';

export default function MyPage() {
  return (
    <div>
      <SimplePDFViewer
        pdfUrl="https://example.com/document.pdf"
        title="My Document"
      />
    </div>
  );
}
```

Common Issues & Solutions

❌ Version Mismatch Error

Error: "The API version X does not match the Worker version Y"

Solution:

```
# Remove any separate pdfjs-dist installation
npm uninstall pdfjs-dist

# Only keep react-pdf
npm install react-pdf
```

❌ Worker Loading Issues

Problem: PDF not loading, worker errors

Solution: Use the CDN worker approach:

```
pdfjs.GlobalWorkerOptions.workerSrc = `//unpkg.com/pdfjs-
dist@${pdfjs.version}/build/pdf.worker.min.mjs`;
```

❌ Module Resolution Errors

Problem: Can't resolve worker files

Solution: Don't try to copy worker files manually. Use the CDN approach above.

Asset Management (Advanced)

If you need local assets for offline use, create a simple copy script:

copy-pdf-worker.js

```
import path from 'node:path';
import fs from 'node:fs';
import { fileURLToPath } from 'node:url';
import { createRequire } from 'node:module';

const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);
const require = createRequire(import.meta.url);

// Get pdfjs-dist from react-pdf dependencies
const reactPdfPath = path.dirname(require.resolve('react-pdf/package.json'));
const pdfjsPath = path.join(reactPdfPath, 'node_modules', 'pdfjs-dist');

// Copy assets to public directory
const assetsToCheck = ['cmaps', 'standard_fonts'];

assetsToCheck.forEach(assetDir => {
  const sourcePath = path.join(pdfjsPath, assetDir);
  const destPath = path.join(__dirname, 'public', assetDir);

  if (fs.existsSync(sourcePath)) {
    fs.cpSync(sourcePath, destPath, { recursive: true, force: true });
    console.log(`✅ ${assetDir} copied successfully`);
  }
});
```

package.json

```
{
  "scripts": {
    "copy-pdf-assets": "node copy-pdf-worker.js",
    "postinstall": "npm run copy-pdf-assets"
  }
}
```

Project Structure

```
your-project/
├── components/
│   └── SimplePDFViewer.jsx    # Your PDF viewer component
├── public/
│   ├── cmaps/                 # PDF font maps (auto-copied)
│   └── standard_fonts/        # PDF fonts (auto-copied)
├── copy-pdf-worker.js         # Asset copy script
└── package.json
```

Key Principles That Make This Work

1. **Single Package:** Only install `react-pdf`, never `pdfjs-dist` separately
2. **CDN Worker:** Use unpkg CDN for worker, avoids local file issues
3. **Version Consistency:** Let react-pdf manage its own pdfjs-dist version
4. **Simple State:** Basic useState hooks, no complex state management
5. **Error Handling:** Simple try-catch with user-friendly messages

What This Implementation Includes

✅ **PDF viewing in modal**
✅ **Page navigation**
✅ **Loading states**
✅ **Error handling**
✅ **Responsive design**
✅ **No version conflicts**
✅ **Works with Next.js 15+**
✅ **Zero configuration hassles**

Real Implementation Example

This guide is based on a working implementation in a Next.js mental health app where:

- PDFs are loaded from external URLs
- Multiple workbooks can be viewed
- Filtering and card-based UI
- Modal-based PDF viewer
- **Zero configuration issues**

The key lesson: **Simple approaches work better than complex ones.** After hours of trying complex setups, this simple approach worked immediately and has been stable ever since.