Failed to load PDF: ${error.message}

h-screen w-full bg-primary text-primary flex items-center justify-center ${className}

grow h-full w-full bg-
secondary text-primary overflow-hidden
flex flex-col ${className} ${isFullscreen ?
'fixed inset-0 z-50' : ''}

p-2 hover:bg-
secondary ${isMobile ? 'hidden' : ''}

scale(${scale})

${(currentPage / numPages) * 100}%

react-pdf

---

ts

```
const [pageDimensions, setPageDimensions] = useState<Record<number,
```

onPageRenderSuccess

tsx

```
const onPageRenderSuccess = useCallback((page: any) => {
  if (page.width && page.height && !pageDimensions[page.pageNumber])
    setPageDimensions(prev => ({
      ...prev,
      [page.pageNumber]: { width: page.width, height: page.height }
    }));
```

```
  }
}, [pageDimensions]);
```

`pageWidth` `pageHeight`

```ts
const dimensions = pageDimensions[currentPage] ?? { width: pageWidth
```

`<Page>`

```tsx
<div
  className="bg-primary shadow-lg rounded-lg overflow-hidden border b
  style={{
    width: dimensions.width * scale,
    height: dimensions.height * scale,
    transition: 'all 0.2s ease-out',
    overflow: 'hidden'
  }}
>
  <Page
    pageNumber={currentPage}
    renderAnnotationLayer={false}
    renderTextLayer={false}
    width={dimensions.width}
    onRenderSuccess={onPageRenderSuccess}
    className="max-w-none"
    loading={
      <div
        className="flex items-center justify-center"
        style={{
          width: dimensions.width,
          height: dimensions.height
        }}
      >
        <Loader2 className="h-6 w-6 animate-spin accent-primary" />
      </div>
    }
  />
</div>
```

`transitioningPage`

renderMode="svg"

react-pdf

renderMode="svg"

```tsx
<Page renderMode="svg" ... />
```

```ts
const pageChangeTimeout = useRef<NodeJS.Timeout | null>(null);

const delayedGoToPage = useCallback((pageNum: number) => {
  if (pageChangeTimeout.current) clearTimeout(pageChangeTimeout.curre
  pageChangeTimeout.current = setTimeout(() => {
    if (pageNum >= 1 && pageNum <= numPages) {
      setCurrentPage(pageNum);
      setPageInput(pageNum.toString());
    }
  }, 150);
}, [numPages]);
```

Strategy	Why It's Important
Store page dimensions	Prevents layout shifts
Set wrapper size ahead of time	Stops flickering
Use memoization	Avoids unnecessary re-renders
Optional: renderMode="svg"	Reduces flicker for high-res PDFs
Optional: delay page switch	Smoother UX on rapid nav

PDFViewer

-
-
-

---

tsx

```tsx
"use client";
import { useCallback, useEffect, useRef, useState, useMemo } from "react"
import { Document, Page, pdfjs } from "react-pdf";
import {
  ZoomIn,
  ZoomOut,
  Download,
  ChevronLeft,
  ChevronRight,
  Maximize2,
  Minimize2,
  FileText,
  Loader2,
  RotateCcw
} from "lucide-react";
import "react-pdf/dist/Page/AnnotationLayer.css";
import "react-pdf/dist/Page/TextLayer.css";
import { Button } from "@/components/ui/button";
```

```tsx
import { Input } from "@/components/ui/input";
import { cn } from "../../libs/tailwindUtils";

pdfjs.GlobalWorkerOptions.workerSrc = new URL(
  "pdfjs-dist/build/pdf.worker.min.mjs",
  import.meta.url
).toString();

interface MobilePDFViewerProps {
  src: string;
  className?: string;
  showPDF?: boolean;
  altComponent?: React.ReactNode;
  onLoad?: () => void;
}

export const PDFViewer = ({
  src,
  className = "",
  showPDF = true,
  altComponent,
  onLoad
}: MobilePDFViewerProps) => {
  const containerRef = useRef<HTMLDivElement>(null);
  const [numPages, setNumPages] = useState<number>(0);
  const [currentPage, setCurrentPage] = useState<number>(1);
  const [scale, setScale] = useState(1);
  const [isLoading, setIsLoading] = useState(true);
  const [error, setError] = useState<string | null>(null);
  const [isFullscreen, setIsFullscreen] = useState(false);
  const [pageInput, setPageInput] = useState("1");
  const [isMobile, setIsMobile] = useState(false);
  const [isTablet, setIsTablet] = useState(false);
  const [containerDimensions, setContainerDimensions] = useState({ width

  const [pageDimensions, setPageDimensions] = useState<Record<numbe

  useEffect(() => {
    const handleResize = () => {
      const width = window.innerWidth;
      setIsMobile(width < 768);
      setIsTablet(width >= 768 && width < 1024);

      if (containerRef.current) {
        const rect = containerRef.current.getBoundingClientRect();
        setContainerDimensions({ width: rect.width, height: rect.height });
      }
    };

    handleResize();
    window.addEventListener('resize', handleResize);
    return () => window.removeEventListener('resize', handleResize);
  }, []);

  useEffect(() => {
    const handleFullscreenChange = () => {
      setIsFullscreen(!!document.fullscreenElement);
    };
    document.addEventListener('fullscreenchange', handleFullscreenChang
    return () => document.removeEventListener('fullscreenchange', handleF
  }, []);

  const goToPage = useCallback((pageNum: number) => {
```

```
  if (pageNum >= 1 && pageNum <= numPages) {
    setCurrentPage(pageNum);
    setPageInput(pageNum.toString());
  }
}, [numPages]);

const optimalPageWidth = useMemo(() => {
  const padding = isMobile ? 16 : 32;
  const availableWidth = containerDimensions.width - padding;
  if (isMobile) return Math.min(availableWidth, 400);
  if (isTablet) return Math.min(availableWidth, 600);
  return Math.min(availableWidth, 800);
}, [containerDimensions.width, isMobile, isTablet]);

const currentDimensions = pageDimensions[currentPage] ?? {
  width: optimalPageWidth || 400,
  height: (optimalPageWidth || 400) * 1.414
};

const onPageRenderSuccess = useCallback((page: any) => {
  if (page.width && page.height && !pageDimensions[page.pageNumber]
    setPageDimensions(prev => ({
      ...prev,
      [page.pageNumber]: { width: page.width, height: page.height }
    }));
  }
}, [pageDimensions]);

if (!showPDF) return altComponent || null;

const onDocumentLoad = ({ numPages }: { numPages: number }) => {
  setNumPages(numPages);
  setIsLoading(false);
  setError(null);
  onLoad?.();
};

const onLoadError = (error: Error) => {
  setError(`Failed to load PDF: ${error.message}`);
  setIsLoading(false);
};

const zoomIn = () => setScale(prev => Math.min(prev + 0.25, 3.0));
const zoomOut = () => setScale(prev => Math.max(prev - 0.25, 0.5));
const resetZoom = () => setScale(1);

const handlePageInputChange = (e: React.ChangeEvent<HTMLInputElem
  setPageInput(e.target.value);
};

const handlePageInputSubmit = (e: React.FormEvent) => {
  e.preventDefault();
  const pageNum = parseInt(pageInput);
  if (pageNum >= 1 && pageNum <= numPages) {
    goToPage(pageNum);
  } else {
    setPageInput(currentPage.toString());
  }
};

const toggleFullscreen = () => {
  if (!document.fullscreenElement) {
    containerRef.current?.requestFullscreen();
```

```
    } else {
      document.exitFullscreen();
    }
  };

  useEffect(() => {
    const handleKeyDown = (event: KeyboardEvent) => {
      if (event.key === 'Escape' && isFullscreen) {
        document.exitFullscreen();
      }
    };
    document.addEventListener('keydown', handleKeyDown);
    return () => document.removeEventListener('keydown', handleKeyDown)
  }, [isFullscreen]);

  if (error) {
    return (
      <div className={`h-screen w-full bg-primary text-primary flex items-ce
        <div className="text-center p-8">
          <FileText className="mx-auto h-16 w-16 text-secondary-red mb-4" />
          <h3 className="text-lg font-medium mb-2">Failed to load PDF</h3>
          <p className="text-tertiary">{error}</p>
        </div>
      </div>
    );
  }

  return (
    <div
      ref={containerRef}
      className={`grow h-full w-full bg-secondary text-primary overflow-hid
    >
      {/* Toolbar */}
      <div className={cn("w-full bg-primary shadow-sm border-b border-pr
        <div className="px-3 py-2 flex items-center justify-between w-full">
          <div className="flex items-center space-x-2">
            <Button onClick={() => goToPage(currentPage - 1)} disabled={curren
              <ChevronLeft className="h-4 w-4" />
            </Button>

            <form onSubmit={handlePageInputSubmit} className="flex items-
              <Input
                type="number"
                min="1"
                max={numPages}
                value={pageInput}
                onChange={handlePageInputChange}
                className="w-12 text-center text-sm bg-primary"
              />
              <span className="text-sm text-tertiary">/{numPages}</span>
            </form>

            <Button onClick={() => goToPage(currentPage + 1)} disabled={curren
              <ChevronRight className="h-4 w-4" />
            </Button>
          </div>

          <div className="flex items-center space-x-1">
            {!isMobile && <span className="text-xs text-tertiary mr-2">{Math.
            <Button onClick={zoomOut} variant="ghost" size="sm" className=
            <Button onClick={resetZoom} variant="ghost" size="sm" classNam
            <Button onClick={zoomIn} variant="ghost" size="sm" className="p
            <div className="h-4 w-px bg-primary" />
```

```
            <Button onClick={toggleFullscreen} variant="ghost" size="sm" class
              {isFullscreen ? <Minimize2 className="h-4 w-4" /> : <Maximize2 c
            </Button>
            <a href={src} download className="inline-flex items-center justify
              <Download className="h-4 w-4" />
            </a>
          </div>
        </div>
      </div>

      {/* Viewer */}
      <div className="flex-1 w-full overflow-hidden relative">
        <Document
          file={src}
          onLoadSuccess={onDocumentLoad}
          onLoadError={onLoadError}
          loading={
            <div className="flex items-center justify-center h-full bg-secondar
              <Loader2 className="h-8 w-8 animate-spin mx-auto mb-4 accent
              <p className="text-base text-tertiary">Loading PDF...</p>
            </div>
          }
        >
          <div ref="h-full w-full overflow-auto bg-secondary minimal-s
            <div ref="p-2 md:p-4 flex justify-center" style={{ minWidth:
              <div
                ref="bg-primary shadow-lg rounded-lg overflow-hidden
                style={{
                  width: currentDimensions.width * scale,
                  height: currentDimensions.height * scale,
                  transition: 'all 0.2s ease-out',
                  overflow: 'hidden'
                }}
              >
                <Page
                  pageNumber={currentPage}
                  width={currentDimensions.width}
                  renderAnnotationLayer={false}
                  renderTextLayer={false}
                  onRenderSuccess={onPageRenderSuccess}
                  ref="max-w-none"
                  loading={
                    <div
                      ref="flex items-center justify-center"
                      style={{
                        width: currentDimensions.width,
                        height: currentDimensions.height
                      }}
                    >
                      <Loader2 ref="h-6 w-6 animate-spin accent-primary"
                    </div>
                  }
                />
              </div>
            </div>
          </div>
        </Document>
      </div>
    </div>
  );
};
```

- renderMode="svg" <Page>
-

- opacity: 0
-

tsx

```tsx
"use client";
import { useCallback, useEffect, useMemo, useRef, useState } from "react"
import { Document, Page, pdfjs } from "react-pdf";
import {
  ChevronLeft, ChevronRight, ZoomIn, ZoomOut,
  RotateCcw, Maximize2, Minimize2, Download,
  FileText, Loader2
} from "lucide-react";
import "react-pdf/dist/Page/AnnotationLayer.css";
import "react-pdf/dist/Page/TextLayer.css";
import { Button } from "@/components/ui/button";
import { Input } from "@/components/ui/input";
import { cn } from "../../libs/tailwindUtils";

pdfjs.GlobalWorkerOptions.workerSrc = new URL(
  "pdfjs-dist/build/pdf.worker.min.mjs",
  import.meta.url
).toString();

interface PDFViewerProps {
  src: string;
  className?: string;
  showPDF?: boolean;
  altComponent?: React.ReactNode;
  onLoad?: () => void;
}

export const PDFViewer = ({
  src,
  className = "",
  showPDF = true,
  altComponent,
  onLoad
}: PDFViewerProps) => {
  const containerRef = useRef<HTMLDivElement>(null);
  const [numPages, setNumPages] = useState(0);
  const [currentPage, setCurrentPage] = useState(1);
  const [scale, setScale] = useState(1);
  const [isLoading, setIsLoading] = useState(true);
  const [error, setError] = useState<string | null>(null);
  const [isFullscreen, setIsFullscreen] = useState(false);
  const [pageInput, setPageInput] = useState("1");
  const [isMobile, setIsMobile] = useState(false);
  const [isTablet, setIsTablet] = useState(false);
  const [containerDimensions, setContainerDimensions] = useState({ width

  const [pageDimensions, setPageDimensions] = useState<Record<numbe

  // Device detection and container size
  useEffect(() => {
    const handleResize = () => {
      const width = window.innerWidth;
      setIsMobile(width < 768);
      setIsTablet(width >= 768 && width < 1024);
      if (containerRef.current) {
        const rect = containerRef.current.getBoundingClientRect();
```

```
      setContainerDimensions({ width: rect.width, height: rect.height });
    }
  };
  handleResize();
  window.addEventListener('resize', handleResize);
  return () => window.removeEventListener('resize', handleResize);
}, []);

// Fullscreen listener
useEffect(() => {
  const handleFullscreenChange = () => setIsFullscreen(!!document.fullscr
  document.addEventListener('fullscreenchange', handleFullscreenChang
  return () => document.removeEventListener('fullscreenchange', handleF
}, []);

const optimalPageWidth = useMemo(() => {
  const padding = isMobile ? 16 : 32;
  const availableWidth = containerDimensions.width - padding;
  if (isMobile) return Math.min(availableWidth, 400);
  if (isTablet) return Math.min(availableWidth, 600);
  return Math.min(availableWidth, 800);
}, [containerDimensions.width, isMobile, isTablet]);

const currentDimensions = useMemo(() => {
  return pageDimensions[currentPage] ?? {
    width: optimalPageWidth || 400,
    height: (optimalPageWidth || 400) * 1.414
  };
}, [pageDimensions, currentPage, optimalPageWidth]);

const onPageRenderSuccess = useCallback((page: any) => {
  const { pageNumber, width, height } = page;
  if (width && height && !pageDimensions[pageNumber]) {
    setPageDimensions(prev => ({
      ...prev,
      [pageNumber]: { width, height }
    }));
  }
}, [pageDimensions]);

const goToPage = useCallback((pageNum: number) => {
  if (pageNum >= 1 && pageNum <= numPages) {
    setCurrentPage(pageNum);
    setPageInput(pageNum.toString());
  }
}, [numPages]);

const handlePageInputChange = (e: React.ChangeEvent<HTMLInputElem
  setPageInput(e.target.value);
};

const handlePageInputSubmit = (e: React.FormEvent) => {
  e.preventDefault();
  const pageNum = parseInt(pageInput);
  if (pageNum >= 1 && pageNum <= numPages) {
    goToPage(pageNum);
  } else {
    setPageInput(currentPage.toString());
  }
};

const toggleFullscreen = () => {
  if (!document.fullscreenElement) {
```

```
      containerRef.current?.requestFullscreen();
    } else {
      document.exitFullscreen();
    }
  };

  useEffect(() => {
    const handleKeyDown = (e: KeyboardEvent) => {
      if (e.key === "Escape" && isFullscreen) {
        document.exitFullscreen();
      }
    };
    document.addEventListener("keydown", handleKeyDown);
    return () => document.removeEventListener("keydown", handleKeyDown
  }, [isFullscreen]);

  const zoomIn = () => setScale(s => Math.min(s + 0.25, 3));
  const zoomOut = () => setScale(s => Math.max(s - 0.25, 0.5));
  const resetZoom = () => setScale(1);

  const onDocumentLoad = ({ numPages }: { numPages: number }) => {
    setNumPages(numPages);
    setIsLoading(false);
    setError(null);
    onLoad?.();
  };

  const onLoadError = (error: Error) => {
    setError(`Failed to load PDF: ${error.message}`);
    setIsLoading(false);
  };

  if (!showPDF) return altComponent || null;

  if (error) {
    return (
      <div className="h-screen w-full flex items-center justify-center bg-pri
        <div className="text-center">
          <FileText className="h-16 w-16 text-secondary-red mb-4 mx-auto" /
          <h3 className="text-lg font-medium">Failed to load PDF</h3>
          <p className="text-tertiary">{error}</p>
        </div>
      </div>
    );
  }

  return (
    <div
      ref={containerRef}
      className={cn("grow w-full h-full bg-secondary text-primary flex flex-c
    >
      {/* Toolbar */}
      <div className={cn("w-full bg-primary border-b shadow-sm z-10", isF
        <div className="flex justify-between px-4 py-2 items-center">
          <div className="flex items-center gap-2">
            <Button onClick={() => goToPage(currentPage - 1)} disabled={curren
            <form onSubmit={handlePageInputSubmit} className="flex items-
              <Input
                type="number"
                min="1"
                max={numPages}
                value={pageInput}
                onChange={handlePageInputChange}
```

```
            className="w-14 text-center text-sm"
          />
          <span className="ml-2 text-sm text-tertiary">/ {numPages}</spa
        </form>
        <Button onClick={() => goToPage(currentPage + 1)} disabled={currer
      </div>
      <div className="flex items-center gap-2">
        <span className="text-xs text-tertiary">{Math.round(scale * 100)}%
        <Button onClick={zoomOut} variant="ghost" size="sm"><ZoomOut />
        <Button onClick={resetZoom} variant="ghost" size="sm"><RotateCc
        <Button onClick={zoomIn} variant="ghost" size="sm"><ZoomIn /></
        <Button onClick={toggleFullscreen} variant="ghost" size="sm">{isFu
        <a href={src} download><Button variant="ghost" size="sm"><Down
      </div>
    </div>
  </div>

  {/* Viewer */}
  <div className="flex-1 overflow-auto relative bg-secondary">
    <Document
      file={src}
      onLoadSuccess={onDocumentLoad}
      onLoadError={onLoadError}
      loading={
        <div className="flex items-center justify-center h-full">
          <Loader2 className="h-8 w-8 animate-spin" />
          <p className="text-tertiary ml-2">Loading PDF...</p>
        </div>
      }
    >
      <div className="flex justify-center p-4">
        {/* Actual Page */}
        <div
          style={{
            width: currentDimensions.width * scale,
            height: currentDimensions.height * scale,
            transition: 'all 0.2s ease-out'
          }}
          className="relative bg-primary border rounded-lg shadow overf
        >
          <Page
            pageNumber={currentPage}
            width={currentDimensions.width}
            renderAnnotationLayer={false}
            renderTextLayer={false}
            onRenderSuccess={onPageRenderSuccess}
          />
          {/* Pre-render Next Page (invisible) */}
          {currentPage < numPages && (
            <div style={{ opacity: 0, position: 'absolute', pointerEvents: 'none'
              <Page
                pageNumber={currentPage + 1}
                width={pageDimensions[currentPage + 1]?.width || currentDi
                renderAnnotationLayer={false}
                renderTextLayer={false}
                onRenderSuccess={onPageRenderSuccess}
              />
            </div>
          )}
        </div>
      </div>
    </Document>
  </div>
```

```
    </div>
  );
};
```

- <Page /> currentPage - 1
-

tsx

```tsx
<div className={cn(
  "w-full bg-primary border-b border-primary shadow-sm z-10",
  isFullscreen && "hidden"
)}>
  <div className="flex flex-wrap justify-between px-2 sm:px-4 py-2 items-c
    {/* Left section */}
    <div className="flex flex-wrap items-center gap-2">
      <Button
        onClick={() => goToPage(currentPage - 1)}
        disabled={currentPage <= 1}
        variant="ghost"
        size="sm"
      >
        <ChevronLeft className="w-4 h-4" />
      </Button>
```

```
      <form onSubmit={handlePageInputSubmit} className="flex items-cen
        <Input
          type="number"
          min="1"
          max={numPages}
          value={pageInput}
          onChange={handlePageInputChange}
          className="w-14 text-center text-sm"
        />
        <span className="ml-2 text-sm text-tertiary whitespace-nowrap">/ {
      </form>

      <Button
        onClick={() => goToPage(currentPage + 1)}
        disabled={currentPage >= numPages}
        variant="ghost"
        size="sm"
      >
        <ChevronRight className="w-4 h-4" />
      </Button>
    </div>

    {/* Right section */}
    <div className="flex flex-wrap items-center gap-2 justify-end">
      <span className="text-xs text-tertiary">{Math.round(scale * 100)}%</
      <Button onClick={zoomOut} variant="ghost" size="sm">
        <ZoomOut className="w-4 h-4" />
      </Button>
      <Button onClick={resetZoom} variant="ghost" size="sm">
        <RotateCcw className="w-4 h-4" />
      </Button>
      <Button onClick={zoomIn} variant="ghost" size="sm">
        <ZoomIn className="w-4 h-4" />
      </Button>
      <Button onClick={toggleFullscreen} variant="ghost" size="sm">
        {isFullscreen ? <Minimize2 className="w-4 h-4" /> : <Maximize2 clas
      </Button>
      <a href={src} download>
        <Button variant="ghost" size="sm">
          <Download className="w-4 h-4" />
        </Button>
      </a>
    </div>
  </div>
</div>
```

- `flex-wrap`
- `gap-y-2`
- `sm:px-4` `px-2`
- `whitespace-nowrap`
- `w-4 h-4`

---

sm+ DropdownMenu

```tsx
import {
  DropdownMenu,
  DropdownMenuTrigger,
  DropdownMenuContent,
  DropdownMenuItem,
  DropdownMenuSeparator,
  DropdownMenuLabel,
} from "@/components/ui/dropdown-menu";
import { MoreVertical } from "lucide-react";
```

```tsx
<div className={cn(
  "w-full bg-primary border-b border-primary shadow-sm z-10",
  isFullscreen && "hidden"
)}>
  <div className="flex flex-wrap justify-between px-2 sm:px-4 py-2 items-c
    {/* Left section */}
    <div className="flex flex-wrap items-center gap-2">
      <Button
        onClick={() => goToPage(currentPage - 1)}
        disabled={currentPage <= 1}
        variant="ghost"
        size="sm"
      >
        <ChevronLeft className="w-4 h-4" />
      </Button>

      <form onSubmit={handlePageInputSubmit} className="flex items-cen
        <Input
          type="number"
          min="1"
          max={numPages}
          value={pageInput}
          onChange={handlePageInputChange}
          className="w-14 text-center text-sm"
        />
```

```
      <span className="ml-2 text-sm text-tertiary whitespace-nowrap">/ {
    </form>

    <Button
      onClick={() => goToPage(currentPage + 1)}
      disabled={currentPage >= numPages}
      variant="ghost"
      size="sm"
    >
      <ChevronRight className="w-4 h-4" />
    </Button>
  </div>

  {/* Right section */}
  <div className="flex items-center gap-2">
    <span className="text-xs text-tertiary">{Math.round(scale * 100)}%</s
    <Button onClick={zoomOut} variant="ghost" size="sm">
      <ZoomOut className="w-4 h-4" />
    </Button>
    <Button onClick={resetZoom} variant="ghost" size="sm">
      <RotateCcw className="w-4 h-4" />
    </Button>
    <Button onClick={zoomIn} variant="ghost" size="sm">
      <ZoomIn className="w-4 h-4" />
    </Button>

    {/* Fullscreen + Download (visible on sm and up) */}
    <Button
      onClick={toggleFullscreen}
      variant="ghost"
      size="sm"
      className="hidden sm:inline-flex"
      aria-label={isFullscreen ? "Exit fullscreen" : "Enter fullscreen"}
    >
      {isFullscreen ? <Minimize2 className="w-4 h-4" /> : <Maximize2 clas
    </Button>
    <a href={src} download className="hidden sm:inline-flex">
      <Button variant="ghost" size="sm" aria-label="Download">
        <Download className="w-4 h-4" />
      </Button>
    </a>

    {/* Mobile dropdown (shown on < sm) */}
    <DropdownMenu>
      <DropdownMenuTrigger asChild>
        <Button
          variant="ghost"
          size="sm"
          className="inline-flex sm:hidden"
          aria-label="More actions"
        >
          <MoreVertical className="w-4 h-4" />
        </Button>
      </DropdownMenuTrigger>
      <DropdownMenuContent align="end" sideOffset={8}>
        <DropdownMenuLabel>Actions</DropdownMenuLabel>
        <DropdownMenuItem
          onSelect={(e) => {
            e.preventDefault(); // prevents menu from closing before action in
            toggleFullscreen();
          }}
        >
          {isFullscreen ? (
```

```
            <Minimize2 className="w-4 h-4 mr-2" />
           ) : (
            <Maximize2 className="w-4 h-4 mr-2" />
           )}
           {isFullscreen ? "Exit Fullscreen" : "Enter Fullscreen"}
          </DropdownMenuItem>
          <DropdownMenuSeparator />
          <DropdownMenuItem asChild>
           <a href={src} download className="flex items-center">
            <Download className="w-4 h-4 mr-2" />
            Download
           </a>
          </DropdownMenuItem>
         </DropdownMenuContent>
        </DropdownMenu>
       </div>
      </div>
     </div>
```

- `< sm` `More` `MoreVertical`
- `sm+`
-

Key	Action
`ArrowRight` / `PageDown`	Next page
`ArrowLeft` / `PageUp`	Previous page
`+`, `-`	Zoom in/out
`0`	Reset zoom

Key	Action
`f`	Toggle fullscreen

```tsx
useEffect(() => {
  function handleKeydown(e: KeyboardEvent) {
    if (e.target instanceof HTMLInputElement || e.target instanceof HTMLTe

    switch (e.key) {
      case "ArrowRight":
      case "PageDown":
        goToPage(currentPage + 1);
        e.preventDefault();
        break;
      case "ArrowLeft":
      case "PageUp":
        goToPage(currentPage - 1);
        e.preventDefault();
        break;
      case "+":
      case "=": // some keyboards use this for zoom in
        zoomIn();
        e.preventDefault();
        break;
      case "-":
        zoomOut();
        e.preventDefault();
        break;
      case "0":
        resetZoom();
        e.preventDefault();
        break;
      case "f":
        toggleFullscreen();
        e.preventDefault();
        break;
    }
  }

  window.addEventListener("keydown", handleKeydown);
  return () => window.removeEventListener("keydown", handleKeydown);
}, [currentPage, zoomIn, zoomOut, resetZoom, toggleFullscreen]);
```

- `<input>` `<textarea>`

- `preventDefault()`
  `PageDown`
- `goToPage` `zoomIn`

- `Home`
- `End`
- `Ctrl+F`

`enableKeyboardShortcuts`