README.md

```
# PDFy 📄✨

**Tired of IDEs exporting your code as massive, unsearchable image-based PDFs? PDFy is here to help!**

PDFy is a client-side web application built with SvelteKit and ShadcnSvelte that allows you to select a local project
folder, choose specific code and text files, preview them with syntax highlighting, and then print them to a clean,
text-based, searchable PDF using your browser's built-in print functionality.

**Live Demo:** [https://pdfy.oseifert.ch](https://pdfy.oseifert.ch)

## ⚠️ Important Note & Current Limitations

Please keep in mind that this project was put together quickly (in about a day!). As such, users **shouldn't expect
complete stability or full feature compatibility** at this stage.

*   **Not all advertised features are fully implemented.**
*   **HTML, Markdown (.md), and XML rendering has not been thoroughly tested and may not work correctly.**
*   Currently, PDFy primarily focuses on rendering **raw text files with syntax highlighting.**

Development is ongoing, and contributions are welcome to improve functionality and address these limitations!

## The Problem PDFy Solves

Many developers, especially students preparing project documentation, face issues with IDEs or tools that generate PDFs
by rasterizing code into images or that can't even do it. This results in:

*   HUGE file sizes (e.g., 100MB instead of <5MB).
*   Non-searchable content.
*   Poor text quality and inability to select/copy code.

PDFy aims to provide a simple, local, and effective solution to generate high-quality, text-based PDFs of your codebase
and documentation.

## Key Features ✨

*   **Local Processing:** No server-side uploads! All file processing happens directly in your browser using the File
    System Access API. Your code stays on your machine.
*   **Folder Select:** Select an entire project folder from your local file system.
*   **File Tree Navigation:** View your project's folder structure and select specific files.
*   **Live Preview:** See an instant preview of your selected files with syntax highlighting.
*   **Syntax Highlighting:** Powered by `highlight.js` for a wide range of programming languages.
*   **Markdown Support:** Preview rendered Markdown files (experimental, see note above).
*   **Print to PDF:** Uses your browser's native "Print to PDF" functionality with optimized print CSS for clean output.
```

```
*   **Text-Based & Searchable PDFs:** Generates PDFs where text is text, not an image.
*   **Customizable:** Settings for themes, line numbers, etc. (partially implemented).
*   **GitIgnore Support:** Automatically ignores files and directories specified in `.gitignore` files within the selected
    project directory.

## Tech Stack 🛠️

*   **Framework:** [SvelteKit](https://kit.svelte.dev/)
*   **UI Components:** [ShadcnSvelte](https://www.shadcn-svelte.com/)
*   **Styling:** [Tailwind CSS](https://tailwindcss.com/)
*   **Package Manager & Runtime:** [Bun](https://bun.sh/)
*   **Syntax Highlighting:** [highlight.js](https://highlightjs.org/)

## Getting Started 🚀

To run PDFy locally, you'll need [Bun](https://bun.sh/) installed.

1.  **Clone the repository:**
    ```bash
    git clone https://github.com/ImGajeed76/pdfy.git
    cd pdfy
    ```

2. **Install dependencies:**
    ```bash
    bun install
    ```

3. **Run the development server:**
    ```bash
    bun run dev
    ```
    This will start the development server, typically on `http://localhost:5173`.

How It Works ⚙️

1. You select a local project directory using the browser's File System Access API.
2. PDFy reads the directory structure and displays it.
3. You select files you want to include in your PDF.
4. Selected files are previewed with appropriate syntax highlighting or rendering.
5. When you're ready, you click "Print to PDF".
6. PDFy uses CSS media queries (`@media print`) to style the content specifically for printing, hiding the application
    UI and formatting the code/text for a standard page.
7. Your browser's print dialog opens, allowing you to save the output as a PDF.

Supported File Types 📚

PDFy aims to support a wide variety of text-based files, including but not limited to:
```

```
92
93  *   **Programming Languages:** JavaScript, TypeScript, Python, Java, C, C++, C#, Rust, Go, PHP, Ruby, Swift,
94      Kotlin, Scala, and many more supported by `highlight.js`.
95  *   **Markup & Data:** HTML, CSS, JSON, XML, YAML, TOML, Markdown (.md - rendering is experimental, see note above).
96  *   **Configuration & Scripts:** `.sh`, `.bash`, `.ps1`, `Dockerfile`, `.env`, `.conf`, `.ini`, `.sql`.
97  *   **Plain Text:** `.txt`, `.log`, `.gitignore`, and other generic text files.
98
99  If a file is text-based, PDFy will do its best to display it with syntax highlighting. Unknown code file types will be treated as plain text.
100 For HTML, MD, and XML, rendering is currently experimental and may default to raw text.
101
102 ## Contributing 🤝
103
104 Contributions are welcome! If you have ideas for improvements, new features, or bug fixes, please feel free to:
105
106 1.  Fork the repository.
107 2.  Create a new branch (`git checkout -b feature/your-feature-name`).
108 3.  Make your changes.
109 4.  Commit your changes (`git commit -m 'Add some feature'`).
110 5.  Push to the branch (`git push origin feature/your-feature-name`).
111 6.  Open a Pull Request.
112
113 Please ensure your code follows the existing style and that any new dependencies are justified.
114
115 ## License 📜
116
117 This project is licensed under the **GNU General Public License v3.0**.
118 See the [LICENSE](LICENSE) file for more details.
119
120 ---
121
122 Made with ❤️ to solve a frustrating problem. Hope it helps you too!
123
```

src/routes/+page.svelte

```svelte
<script lang="ts">

    import { toast } from 'svelte-sonner';
    import { openDirectory } from '$lib/fileSystem';
    import * as Resizable from '$lib/components/ui/resizable/index.js';
    import { Button } from '$lib/components/ui/button';
    import { fileTree, isLoadingDirectory, rootDirectoryHandle, selectedFilesForPrint } from '$lib/stores';
    import FileExplorer from '$lib/components/ui/pdfy/FileExplorer.svelte';
    import FilePreview from '$lib/components/ui/pdfy/FilePreview.svelte';
    import { ScrollArea } from '$lib/components/ui/scroll-area';

    let fsApiSupported = $state(true); // Assuming the API is supported

    $effect(() => {
        if (!window.showDirectoryPicker) {
            fsApiSupported = false;
            toast.error(
                'Your browser does not support the File System Access API. Please use a compatible browser like Chrome or Edge.'
            );
        }
    });

    async function handleLoadFolder() {
        isLoadingDirectory.set(true);
        fileTree.set(null);

        const result = await openDirectory();
        if (result) {
            rootDirectoryHandle.set(result.handle);
            fileTree.set(result.tree);
            toast.success('Directory loaded successfully!');
        } else {
            toast.error(
                'Failed to load directory. Please ensure you have selected a valid folder.'
            );
        }

        isLoadingDirectory.set(false);
    }

    function handlePrint() {
        if ($selectedFilesForPrint.length === 0) {
            toast.info('Please select some files to print.');
            return;
        }
        window.print();
    }
```

```
</script>

<svelte:head>
    <title>PDFy - Code to PDF</title>
    <style global>
      @media print {
          @page {
              size: A4; /* Or 'letter' etc. */

              @top-center {
                  content: ''; /* Cleared, as file path is in FilePreview */
                  font-size: 9pt;
                  font-family: Arial, sans-serif;
                  color: #555555;
              }

              @bottom-right {
                  content: counter(page) ' / ' counter(pages); /* counter(pages) is often unsupported */
                  font-size: 9pt;
                  font-family: Arial, sans-serif;
                  color: #555555;
              }
              @bottom-left {
                  content: ''; /* Can add date or other info here if needed */
              }
          }

          body,
          html {
              background-color: white !important;
              color: black !important;
              font-family: 'Times New Roman', Times, serif; /* Common print serif font */
              height: auto !important; /* Allow content to flow naturally */
              width: auto !important;
              overflow: visible !important; /* Ensure all content is available for print */
          }

          /* Hide elements specifically marked as print:hidden */
          .print\:hidden {
              display: none !important;
          }
          /* Ensure elements marked print:block are indeed block */
          .print\:block {
              display: block !important;
          }

          /* Global reset for links in print if desired */
          a {
```

```
            text-decoration: none !important; /* Avoid underlines unless specifically styled */
            color: inherit !important; /* Links should not stand out unless intended */
        }
        /* Optional: Show hrefs for external links
                    a[href^="http"]::after, a[href^="https"]::after {
                            content: " (" attr(href) ")";
                            font-size: 0.8em;
                            color: #337ab7;
                    }
                */

        /* Ensure Resizable Panes don't interfere with print layout */
        [data-resizable-pane-group], [data-resizable-pane] {
            display: block !important; /* Override flex/grid from resizable */
            width: auto !important;
            height: auto !important;
            overflow: visible !important;
        }
        [data-resizable-handle] {
            display: none !important; /* Hide resize handles */
        }
        .border-2, .border, .border-r-transparent { /* Reset borders that might be on panes */
            border: none !important;
        }
        .rounded-tl-xl, .rounded-bl-xl {
            border-radius: 0 !important;
        }
        .bg-muted\/20 { /* Reset backgrounds */
            background-color: transparent !important;
        }
        .py-5, .p-2, .p-3 { /* Reset paddings if they conflict with page margins */
            padding: 0 !important;
        }
        .h-full {
            height: auto !important;
        }
        .w-full {
            width: auto !important; /* Let print media decide width */
        }

        /* Ensure ScrollArea content is fully visible */
        [data-radix-scroll-area-viewport] {
            height: auto !important;
            overflow: visible !important;
        }
        [data-radix-scroll-area-scrollbar] {
            display: none !important;
        }
```

```
          }
      }
    </style>
</svelte:head>

<div class="flex h-screen flex-col print:hidden">
    <header
        class="flex h-14 items-center gap-4 border-b bg-muted/40 px-4 lg:h-[60px] lg:px-6 print:hidden"
    >
        <a href="/" class="flex items-center gap-2 font-semibold">
            <!-- SVG Icon -->
            <svg xmlns="http://www.w3.org/2000/svg" width="24" height="24" viewBox="0 0 24 24" fill="none"
                stroke="currentColor" stroke-width="2" stroke-linecap="round" stroke-linejoin="round" class="h-6 w-6">
                <path d="M14.5 2H6a2 2 0 0 0-2 2v16a2 2 0 0 0 2 2h12a2 2 0 0 0 2-2V7.5L14.5 2z"></path>
                <polyline points="14 2 14 8 20 8"></polyline>
                <line x1="16" y1="13" x2="8" y2="13"></line>
                <line x1="16" y1="17" x2="8" y2="17"></line>
                <line x1="10" y1="9" x2="8" y2="9"></line>
            </svg>
            <span>PDFy</span>
        </a>
        <Button onclick={handleLoadFolder} disabled={!fsApiSupported || $isLoadingDirectory} class="ml-auto">
            {#if $isLoadingDirectory}
                Loading...
            {:else}
                Load Project Folder
            {/if}
        </Button>
        <Button variant="outline" onclick={handlePrint}>Print to PDF</Button>
    </header>

    {#if !fsApiSupported}
        <div class="flex flex-1 items-center justify-center p-4 print:hidden">
            <p class="text-destructive text-center">
                PDFy requires the File System Access API, which is not supported by your
                browser. <br /> Please try a modern browser like Chrome, Edge, or Opera.
            </p>
        </div>
    {:else}
        <Resizable.PaneGroup direction="horizontal" class="flex-1 print:hidden">
            <Resizable.Pane defaultSize={25} minSize={15}>
                <div class="flex h-full items-start justify-center p-1 print:hidden">
                    {#if $isLoadingDirectory}
                        <p class="p-4 text-muted-foreground">Loading directory structure...</p>
                    {:else if $fileTree}
                        <FileExplorer tree={$fileTree} />
                    {:else}
```

```
                        <p class="p-4 text-center text-muted-foreground">
                            Click "Load Project Folder" to begin.
                        </p>
                    {/if}
                </div>
            </Resizable.Pane>
            <div class="h-full flex items-center print:hidden">
                <Resizable.Handle withHandle class="h-[80%] bg-transparent" />
            </div>
            <Resizable.Pane defaultSize={75} class="py-5">
                <div
                    class="w-full h-full border-2 border-r-transparent p-2 rounded-tl-xl rounded-bl-xl bg-muted/20 print:hidden">
                    <ScrollArea class="h-full p-3">
                        {#if $selectedFilesForPrint.length === 0}
                            <p class="p-6 text-center text-muted-foreground">
                                Select files from the explorer to preview them here.
                            </p>
                        {:else}
                            {#each $selectedFilesForPrint as selectedFile, _index (selectedFile.id)}
                                {#if selectedFile.kind === "file"}
                                    <FilePreview pdfyFile={selectedFile} />
                                {:else}
                                    <p class="p-4 text-muted-foreground">
                                        {selectedFile.name} is not a file or is not selected.
                                    </p>
                                {/if}
                            {/each}
                        {/if}
                    </ScrollArea>
                </div>
            </Resizable.Pane>
        </Resizable.PaneGroup>
    {/if}
</div>

<div class="hidden print:block" aria-hidden="true">
    {#if $selectedFilesForPrint.length === 0}
        <p class="p-6 text-center text-muted-foreground">
            Select files from the explorer to preview them here.
        </p>
    {:else}
        {#each $selectedFilesForPrint as selectedFile, _index (selectedFile.id)}
            {#if selectedFile.kind === "file"}
                <FilePreview pdfyFile={selectedFile} />
            {:else}
                <p class="p-4 text-muted-foreground">
                    {selectedFile.name} is not a file or is not selected.
```

```
                </p>
            {/if}
        {/each}
    {/if}
</div>
```

src/lib/types.ts

```ts
export type PDFYFileSystemEntry = {
    id: string,
    name: string,
    kind: 'directory',
    path: string, // Relative path from the root directory
    handle: FileSystemDirectoryHandle,
    children: PDFYFileSystemEntry[], // For directories
} | {
    id: string,
    name: string,
    kind: 'file',
    path: string, // Relative path from the root directory
    handle: FileSystemFileHandle,
    selected: boolean,
    content: string | null,
    language: string | null, // Detected language for syntax highlighting
    fileType: PDFYFileType | null; // To help with rendering
}

export type PDFYFileType = 'code' | 'rendered' | 'text' | 'graphic' | 'binary'

export enum RenderMode {
    Rendered = 'rendered',
    Raw = 'raw',
}

export interface IgnoreRuleSet {
    rules: string[]; // Raw rule strings from one .gitignore file
    basePath: string; // The path of the directory containing this .gitignore, relative to scan root
}

export const CODE_EXTENSIONS: ReadonlySet<string> = new Set([
    "js",
    "ts",
    "jsx",
    "tsx",
    "json",
    "html",
    "htm",
    "css",
    "scss",
    "less",
    "xml",
    "yaml",
    "yml",
    "toml",
    "py",
```

```
    "rb",
    "php",
    "java",
    "c",
    "cpp",
    "h",
    "hpp",
    "cs",
    "rs",
    "go",
    "swift",
    "kt",
    "sh",
    "bash",
    "ps1",
    "bat",
    "cmd",
    "sql",
    "graphql",
    "gql",
    "svelte",
    "vue",
]);
export const MARKDOWN_EXTENSIONS: ReadonlySet<string> = new Set(["md", "markdown"]);
export const TEXT_EXTENSIONS: ReadonlySet<string> = new Set(["txt", "log", "csv", "tsv"]);
export const IMAGE_EXTENSIONS: ReadonlySet<string> = new Set([
    "png",
    "jpg",
    "jpeg",
    "gif",
    "bmp",
    "webp",
    "svg",
    "ico",
]);
export const PDF_EXTENSIONS: ReadonlySet<string> = new Set(["pdf"]);
export const GENERIC_BINARY_EXTENSIONS: ReadonlySet<string> = new Set([
    "zip",
    "tar",
    "gz",
    "rar",
    "7z",
    "exe",
    "dll",
    "bin",
    "o",
    "a",
    "so",
```

```
    "dylib",
    "app",
    "msi",
    "doc",
    "docx",
    "xls",
    "xlsx",
    "ppt",
    "pptx",
    "odt",
    "ods",
    "odp",
    "mp3",
    "wav",
    "ogg",
    "aac",
    "flac",
    "mp4",
    "avi",
    "mov",
    "wmv",
    "mkv",
    "webm",
    "iso",
    "img",
    "dmg",
    "wasm",
    "eot",
    "ttf",
    "woff",
    "woff2",
    "class",
    "jar",
]);

export type IntrinsicFileType =
    | "markdown"
    | "image"
    | "pdf"
    | "code"
    | "text"
    | "generic_binary";
```

src/lib/fileSystem.ts

```ts
import {
    CODE_EXTENSIONS, GENERIC_BINARY_EXTENSIONS,
    type IgnoreRuleSet,
    IMAGE_EXTENSIONS, type IntrinsicFileType,
    MARKDOWN_EXTENSIONS,
    PDF_EXTENSIONS,
    type PDFYFileSystemEntry, type PDFYFileType, TEXT_EXTENSIONS
} from '$lib/types';
import NodeIgnore from 'ignore';

async function processDirectory(
    directoryHandle: FileSystemDirectoryHandle,
    currentPath: string = '', // Path of this directoryHandle, relative to the initial scan root
    // Chain of rule sets from .gitignore files in ancestor directories
    ancestorRuleSets: IgnoreRuleSet[] = [],
    // Names of ignore files to look for (e.g., ['.gitignore', '.myignore'])
    ignoreFileNames: string[] = ['.gitignore'],
    maxDepth: number = 5, // Maximum recursion depth
    // Hardcoded names of directories/files to always ignore by their direct name
    alwaysIgnoreNames: string[] = ['.git', 'node_modules', '.vscode', '.idea']
): Promise<PDFYFileSystemEntry[]> {
    if (maxDepth <= 0) {
        console.warn(`Max depth reached for directory: ${currentPath}`);
        return [];
    }

    // 1. Read rules from local ignore files in the current directory
    const localRulesRaw: string[] = [];
    for (const ignoreFileName of ignoreFileNames) {
        try {
            const ignoreFileHandle = await directoryHandle.getFileHandle(ignoreFileName);
            const file = await ignoreFileHandle.getFile();
            const content = await file.text();
            content.split('\n').forEach((line) => {
                const trimmedLine = line.trim();
                // Add if not empty and not a comment
                if (trimmedLine && !trimmedLine.startsWith('#')) {
                    localRulesRaw.push(trimmedLine);
                }
            });
        } catch (_error) {
            // Ignore file not found, which is normal and expected
        }
    }

    // 2. Prepare the full list of rule sets for children directories
    // This includes rules from all ancestors plus the rules from the current directory
```

```
    const ruleSetsForChildren: IgnoreRuleSet[] = [...ancestorRuleSets];
    if (localRulesRaw.length > 0) {
        ruleSetsForChildren.push({
            rules: localRulesRaw,
            basePath: currentPath
        });
    }

    // 3. Create an `ignore` instance for checking entries at *this* current level.
    // This instance will be configured with rules from all relevant .gitignore files
    // (ancestors + current). All patterns are prefixed to be relative to the scan root
    // to work correctly with the `ignore` library's path matching.
    const igChecker = NodeIgnore();

    // Add rules from all ancestor .gitignore files, correctly prefixed
    for (const ruleSet of ancestorRuleSets) {
        const prefixedRules = ruleSet.rules.map((rule) => {
            let pattern = rule;
            let negate = '';
            if (pattern.startsWith('!')) {
                negate = '!';
                pattern = pattern.substring(1);
            }

            // If pattern starts with '/', it's anchored to its .gitignore's directory (basePath)
            if (pattern.startsWith('/')) {
                return negate + ruleSet.basePath + pattern; // e.g., basePath "src", pattern "/foo" -> "src/foo"
            } else {
                // Pattern is relative to its .gitignore's directory (basePath)
                // e.g., basePath "src", pattern "foo" -> "src/foo"
                // e.g., basePath "", pattern "foo" -> "foo"
                return negate + (ruleSet.basePath ? ruleSet.basePath + '/' : '') + pattern;
            }
        });
        igChecker.add(prefixedRules);
    }

    // Add rules from the current directory's .gitignore files, correctly prefixed
    const prefixedLocalRules = localRulesRaw.map((rule) => {
        let pattern = rule;
        let negate = '';
        if (pattern.startsWith('!')) {
            negate = '!';
            pattern = pattern.substring(1);
        }
        if (pattern.startsWith('/')) {
            return negate + currentPath + pattern;
```

```
        } else {
            return negate + (currentPath ? currentPath + '/' : '') + pattern;
        }
    });
    igChecker.add(prefixedLocalRules);

    const entries: PDFYFileSystemEntry[] = [];
    for await (const handle of directoryHandle.values()) {
        // First, check against the list of names to always ignore
        if (alwaysIgnoreNames.includes(handle.name)) {
            continue;
        }

        // Construct the full path of the entry relative to the scan root
        const entryPath = `${currentPath}${currentPath ? '/' : ''}${handle.name}`;
        const isDirectory = handle.kind === 'directory';

        // Prepare the path for checking with the `ignore` instance.
        // Directories should end with a slash for correct matching.
        let pathForChecking = entryPath;
        if (isDirectory) {
            // Ensure entryPath is not empty (root "" has no children named "")
            // and append slash if it's a directory and doesn't have one.
            if (pathForChecking && !pathForChecking.endsWith('/')) {
                pathForChecking += '/';
            } else if (!pathForChecking && handle.name) {
                // Case where currentPath is "" (root), entryPath is just handle.name
                pathForChecking = handle.name + '/';
            }
        }

        // Check if the entry is ignored by the aggregated rules
        if (igChecker.ignores(pathForChecking)) {
            continue; // Skip this ignored entry
        }

        // Entry is not ignored, process it
        if (handle.kind === 'file') {
            entries.push({
                id: entryPath,
                name: handle.name,
                kind: 'file',
                path: entryPath,
                handle,
                selected: false, // Assuming 'selected' is for UI state
                content: null, // Content will be loaded on demand
                language: null, // Language will be determined later
                fileType: null // File type will be determined later
            });
```

```
        } else if (handle.kind === 'directory') {
            entries.push({
                id: entryPath,
                name: handle.name,
                kind: 'directory',
                path: entryPath,
                handle,
                // Recursively process the subdirectory, passing the updated rule sets
                children: await processDirectory(
                    handle,
                    entryPath,
                    ruleSetsForChildren, // These include current dir's rules for its children
                    ignoreFileNames,
                    maxDepth - 1,
                    alwaysIgnoreNames
                )
            });
        }
    }

    // Sort entries: directories first, then files, all alphabetically
    return entries.sort((a, b) => {
        if (a.kind === 'directory' && b.kind === 'file') return -1;
        if (a.kind === 'file' && b.kind === 'directory') return 1;
        return a.name.localeCompare(b.name);
    });
}

export async function openDirectory(): Promise<{
    handle: FileSystemDirectoryHandle;
    tree: PDFYFileSystemEntry[];
} | null> {
    try {
        const handle = await window.showDirectoryPicker();
        if (handle) {
            const tree = await processDirectory(handle);
            return { handle, tree };
        }
    } catch (error) {
        if ((error as DOMException).name === 'AbortError') {
            console.log('User aborted directory selection.');
        } else {
            console.error('Error opening directory:', error);
        }
    }
    return null;
}

```

```
function getFileExtension(fileName: string): string {
    const lastDotIndex = fileName.lastIndexOf(".");
    // Ensure dot is not the first character (e.g., .gitignore)
    // and there is something after the dot.
    if (
        lastDotIndex === -1 ||
        lastDotIndex === 0 ||
        lastDotIndex === fileName.length - 1
    ) {
        return "";
    }
    return fileName.substring(lastDotIndex + 1).toLowerCase();
}

function classifyFileType(
    fileName: string,
    mimeType?: string,
): IntrinsicFileType {
    const extension = getFileExtension(fileName);

    if (MARKDOWN_EXTENSIONS.has(extension)) return "markdown";
    if (IMAGE_EXTENSIONS.has(extension)) return "image";
    if (PDF_EXTENSIONS.has(extension)) return "pdf";
    if (CODE_EXTENSIONS.has(extension)) return "code";
    if (TEXT_EXTENSIONS.has(extension)) return "text";

    if (mimeType) {
        if (mimeType.startsWith("image/")) return "image";
        if (mimeType === "application/pdf") return "pdf";
        if (mimeType === "text/markdown") return "markdown";
        if (mimeType === "text/plain") return "text";
        if (
            mimeType.startsWith("text/") || // Catches text/html, text/css, text/javascript
            mimeType.includes("javascript") ||
            mimeType.includes("json") ||
            mimeType.includes("xml")
        ) {
            return "code";
        }
        if (
            mimeType.startsWith("audio/") ||
            mimeType.startsWith("video/") ||
            mimeType === "application/octet-stream"
        ) {
            return "generic_binary";
        }
    }

    if (GENERIC_BINARY_EXTENSIONS.has(extension)) return "generic_binary";
```

```
241
242     // Default for unknown extensions not explicitly binary:
243     // Consider it text, as it might be a less common code or data format.
244     return "text";
245 }
246
247 export function determineFileDisplayProperties(
248     fileName: string,
249     fileMimeType?: string,
250 ): { language: string; fileType: PDFYFileType } {
251     const extension = getFileExtension(fileName);
252     const intrinsicType = classifyFileType(fileName, fileMimeType);
253
254     let language = extension || "unknown"; // Default language to extension
255
256     switch (intrinsicType) {
257         case "markdown":
258             return { language: "markdown", fileType: "rendered" };
259         case "image":
260             return { language: extension, fileType: "graphic" };
261         case "pdf":
262             return { language: "pdf", fileType: "binary" };
263         case "code":
264             if (extension === "html" || extension === "htm") language = "html";
265             else if (extension === "js" || extension === "jsx") language = "javascript";
266             else if (extension === "ts" || extension === "tsx") language = "typescript";
267             else if (extension === "css") language = "css";
268             else if (extension === "json") language = "json";
269             return { language, fileType: "code" };
270         case "text":
271             return { language: "plaintext", fileType: "text" };
272         case "generic_binary":
273             return { language: extension, fileType: "binary" };
274         default:
275             // Should be unreachable if classifyFileType is exhaustive
276             return { language: "unknown", fileType: "binary" };
277     }
278 }
279
280 export async function readFileContent(
281     fileHandle: FileSystemFileHandle,
282 ): Promise<string> {
283     try {
284         const file = await fileHandle.getFile();
285         const intrinsicType = classifyFileType(file.name, file.type);
286
287         if (
288             intrinsicType === "code" ||
```

```
            intrinsicType === "text" ||
            intrinsicType === "markdown"
        ) {
            return await file.text();
        } else {
            return `(Preview not available for ${intrinsicType} file: ${file.name})`;
        }
    } catch (error) {
        console.error(`Error reading file ${fileHandle.name}:`, error);
        if (error instanceof DOMException) {
            return `(Error reading file: ${fileHandle.name} - ${error.message})`;
        }
        return `(Error reading file: ${fileHandle.name})`;
    }
}
```