

Analysis of Operating System Concepts: OS-Style Design and Template Implementation Guide for Web Indexing

Jue Kong

ABSTRACT

The Web Indexing OS-Style Template redefines webpage indexing by treating the index page as an entry-point operating system rather than a content display system. It prioritizes intuitive navigation, repository-based storage, and seamless file access through browser-native tab management. By delegating process and window management to the browser, this template maintains a minimalist, high-availability structure that ensures logical consistency. With features like automatic update reports, card-based layouts, and multi-faceted file stacking, it offers an efficient, self-contained solution for organizing and accessing project-related resources.

Chapter 1: Concepts of Operating Systems and Interaction Design

Since its inception, the concept of the operating system has continuously expanded and evolved. From the early, simple resource scheduling tools to today's sophisticated human-computer interaction platforms, operating systems have always developed around the goal of managing system resources and user activities in an increasingly intuitive and efficient manner. In order to clarify the specific advanced interaction design addressed by this project, it is first necessary to define what an operating system is and to pinpoint its core functionalities.

Fundamental Concepts of Interaction Flow

Within the context of this project, my definition of an operating system centers on its role as a gateway for resource indexing and management, rather than solely a content display platform. Specifically, the core function of an operating system is to maintain the topological structure of files and manage the lifecycle of various windows and processes without directly engaging in content presentation. This design philosophy emphasizes a clear demarcation between the operating system, as an abstract management layer, and the tangible resources (such as files, devices, or applications). Such positioning not only aligns with the principles of modular design but also enhances the system's overall flexibility and scalability.

In a fully functional operating system, three main types of entry points must typically be considered: the gateway for system management software (for example, process management tools like the Command Prompt in Windows); the gateway for IO peripheral management software (which encompasses controlling keyboards, mice, cameras, screen brightness, volume, and network connections); and the file gateway, which not only includes the files themselves but also the applications designed for opening and editing those files. Although some of these applications might be partially integrated into the system, their functional roles are clearly distinguished from the other two categories, as they do not participate in the core operations of system control.

Separation of Interaction and Retrieval

However, when we attempt to incorporate the first two entry points—system management software and peripheral management software—into a file-centric indexing structure, inherent issues arise. The root of these issues lies in the fundamental difference between interactive operations and retrieval operations: retrieval tasks (such as opening a file) are closed and well-defined, functioning solely within the current interface without affecting unrelated resources or other interfaces; in contrast, interactive operations (for example, adjusting brightness, volume, or network status) have a global impact and generally cannot be confined to a single interface or context.

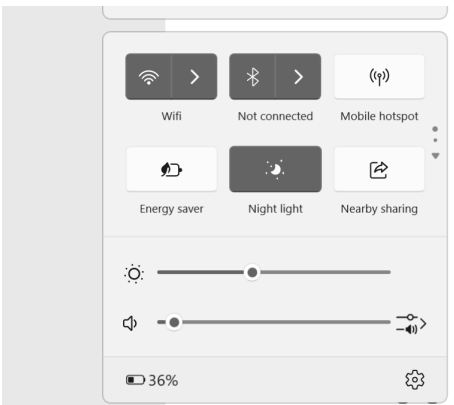
For instance, the control center, which is launched by swiping from the edge of a smartphone screen, exemplifies the design philosophy behind interactive entry points. This type of entry is intuitively independent of the file system, allowing users to initiate interactive operations at any time without conflating them with file retrieval processes. Conversely, if a function such as screen brightness adjustment were mistakenly integrated into the file indexing structure, a user might inadvertently trigger brightness control while searching for a file, resulting in disrupted interaction flow and a fragmented user experience.

Thus, the advanced interaction flow we propose is founded on a strict demarcation between “interaction” and “retrieval” operations. In this framework, “interaction” comprises all actions that have an immediate effect on the device or system state—such as taking a photo (direct hardware control), accessing network data via a browser (direct manipulation of network resources), or remotely controlling a drone (real-time operation of external devices). In contrast, “retrieval” is solely limited to the operations for opening files and their associated applications; these operations do not induce global state changes and have clearly defined operational boundaries.

Through this analysis, it is evident that in smart devices, only pure files and the applications designed to open them are truly suitable for file structure management, whereas interactive functions should be independently designed as special entry points to preserve the integrity and clarity of the file structure. This design principle not only ensures a streamlined internal system architecture and easier configuration but also provides users with a coherent and efficient interaction experience, significantly enhancing the operating system's usability and reliability in practical application scenarios.

Analysis of the App Mode Design Concept

The App Mode converts system management software and most IO peripheral management software into flexibly configurable, system-level components, thereby enabling unified access through a global control center. This strategy not only ensures an intuitive mode of interaction but also clearly delineates interactive operations from file retrieval tasks. Secondary or backup IO peripheral entry points, along with various applications responsible for opening files, are consolidated as the foundational layer above the desktop. Furthermore, the overall topological indexing structure of files is abstracted into a dedicated "File" app, which is treated as an independent application within the system. This design effectively differentiates, on an intuitive level, system control software from general file access applications—particularly by distinctly categorizing peripheral management software apart from file-opening programs—thereby further enhancing user interaction efficiency.



Control Center in Windows 11 (simulated plastic card material)

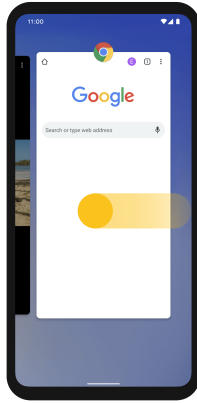
However, this model exhibits several obvious deficiencies. Once a user navigates into the file tree, they must sift through a complex and expansive path structure to locate various files, resulting in reduced operational efficiency and difficulty in promptly accessing the desired content. Moreover, the model performs suboptimally in professional scenarios where frequent software switching is required to handle the same file.

To address these issues, we propose an enhanced optimization: transforming the singular “File” app into multiple independent file apps, thereby decoupling them from the traditional file system’s unified root node. Each file or folder can be indexed via multiple distinct paths, which eliminates redundancy inherent in conventional single-path systems. For instance, a folder named “Guidelines” could be associated with several different projects concurrently without duplicative storage, enabling users to reach the same target through multiple pathways. This model is particularly suited for multi-scenario utilization of shared resources.

Additionally, it is important to note that, in this context, the file entry—which is an integral part of the system—must support a variety of opening methods and allow users to choose their preferred window mode (such as opening multiple instances or launching a new window) to ensure both interactive flexibility and operational efficiency.

Constructing a System Logic Aligned with Intuitive User Operation

Building on the aforementioned ideas, we have gradually outlined a comprehensive framework that accords with users’ intuitive operating habits. We have preserved the primary advantages of the App Mode by consolidating all interactive software—such as process management and IO control components—into the screen edges or global quick-access areas (e.g., via edge swipes or shortcut gestures). This strategy effectively eliminates the scattered and unnecessary entry points or tools typically found in the sidebars or footers of traditional system interfaces. Furthermore, we have eliminated the app desktop entry’s backup role for these interactive applications, thereby ensuring that the desktop focuses solely on hosting “retrieval” type files and applications that are closely bound to them (such as games or notepads). When a user opens a folder, they can conveniently choose the desired method: a light tap automatically invokes the default opening mode, while a long press quickly brings up alternative available options. In addition, file opening actions are, by default, executed in a new window or popup mode, rendering the operational logic more intuitive and further enhancing the overall smoothness and convenience of the user experience.



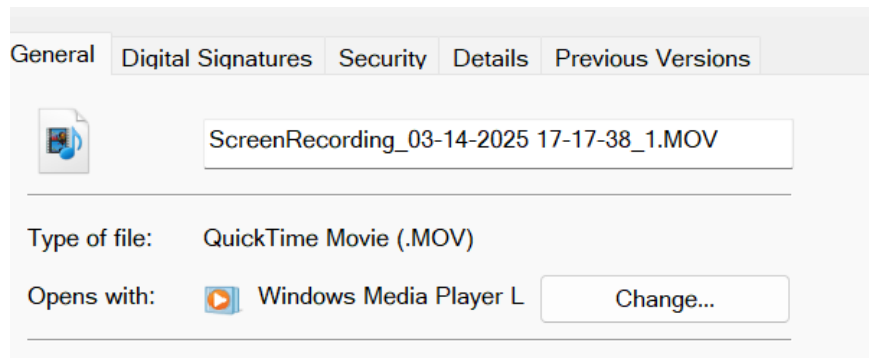
Edge touch controls in Android serve as a global quick-access entry point.

Special Software Worth Mentioning: Terminal

Within the aforementioned logical framework, the Terminal is a software tool that merits special attention. As previously noted, the Terminal essentially functions as a resource manager, executing code through embedded commands to provide an intuitive and user-friendly interface for software that otherwise lacks a graphical user interface. This category includes certain applications designed for file opening. In practical file access operations, we cannot directly select these file openers—due to their lack of an independent interface. In such cases, it becomes necessary to combine a specific file opener with the Terminal (cmd) to seamlessly access file contents and perform subsequent processing.

Another Special Tool: the Configurator

In addition, it is important to highlight another critical software tool: the Configurator. The Configurator encompasses two primary configuration tasks: dedicated configuration for specific files and general configuration for non-specific files. This configuration involves not only determining the default methods for opening files but also setting various specialized parameters. In traditional operating systems, the “configuration” function was commonly manifested through a file properties (Properties) tool. However, in our system design, we take it a step further by treating the Configurator as a specialized editor, thereby enabling it to be invoked and managed in a more intuitive and fluid manner within the file structure.



Properties interface in Windows

It is worth noting that operations such as copying and moving are not categorized as "editing" within our logical framework; rather, they are more akin to interactive operations. Such operations should be activated through a unified control center or quick-access mode that launches a specific editing interface.

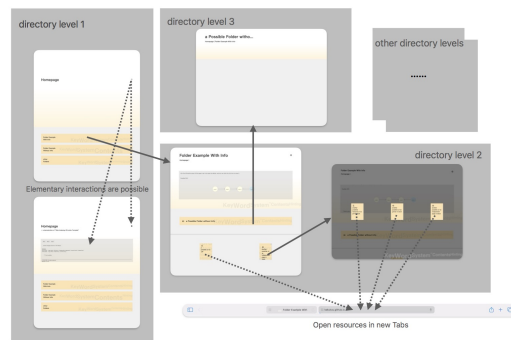
Batch Processing

Finally, to facilitate batch operations on files, we have designed a more convenient handling method. If the viewer itself supports batch file opening (for example, via the Configurator), users can enter a multi-select mode for files by simply long-pressing the background area; long-pressing a folder will automatically select all files within that folder. It is important to mention that this long-press gesture also falls within the realm of process management for quick interactions, with the aim of further enhancing operational efficiency and the intuitive responsiveness of the system.

Chapter 2: A Preview of Simplified Interaction Design Implemented via the Browser

Building on the previously established system design, we can further isolate scenarios dedicated solely to viewing and previewing. Within the framework of an existing operating system, leveraging the basic functionalities of a browser allows the creation of a more lightweight and easily implemented simplified version for user experience. In the context of web implementation, we only need to consider one type of file—namely, file links in URL form. This design inherently eliminates the complexities associated with bulk file opening, and reduces what would normally require system-level process and file window management to simply the browser's "open in new tab" operation.

In effect, the indexing webpage adopts an operating system perspective that is dependent on the browser, serving purely as an index portal for files and resources. It utilizes the browser along with repository platforms (such as GitHub) to handle data storage and content display redirection, thereby achieving a highly streamlined management model. Thanks to this design, we are able to construct a webpage template that is simple to configure, highly available, and endowed with a self-consistent navigation flow.



Example structure of the webpage <https://kakukuu.github.io/Web-Indexing-OS-style-Template/index.html>

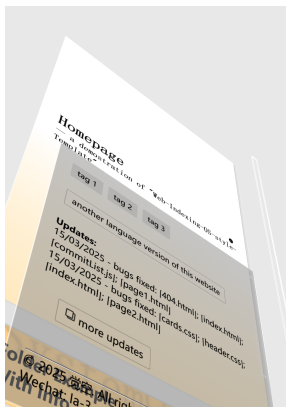
Chapter 3: Building an Intuitive Web Template with Two Core Layers

This project not only provides a simplified implementation of conceptual operating system design but also serves directly as a practical template for personal homepages or project resource distribution platforms. We deliberately abandon the traditional approach of directly displaying concrete content, instead focusing entirely on the two core functions of “indexing” and “overview.” Through an intuitive, card-based layout, efficient and concise navigation flow, and contextual cues throughout the page, this template creates a unified and harmonious interactive experience. Furthermore, to enhance the practical value of the template, an automatic update report feature based on GitHub Pages is integrated, enabling dynamic information synchronization at minimal maintenance cost.

The following sections will detail the template’s design concepts, architectural implementation, and practical application methods with reference to the actual code.

Integrating Flat Interaction Intuition with Unstructured Visual Elements

Since the screen serves as the medium for information display and interaction, it naturally accords with intuitive operations based on a two-dimensional space. This intuition drives us to prioritize visual elements with clear, flat attributes so that users can perceive and interact with them directly and accurately. Moreover, the chosen visual elements should not only have distinct flat characteristics but also avoid complex internal topologies that might confuse interactions with the file system. Therefore, our template favors naturally unstructured phenomena—such as a cloudless sky or a pervasive light mist—that inherently lack well-defined internal hierarchies. These elements can be used as flat backgrounds or supportive visual layers, thereby establishing a sense of calm and order without interfering with the intuitive interaction of files and processes. This visual tranquility also helps focus users' attention more on content interaction itself.



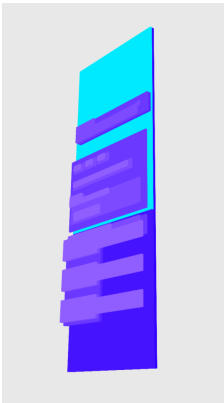
Various layers of flat design

Selecting the Right Structure for Flat Interaction and Visual Elements

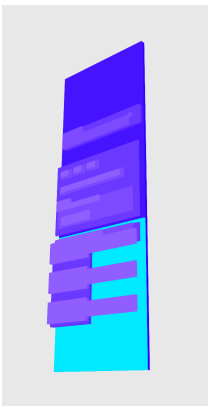
Given that the screen, as an interactive medium, fundamentally aligns with a two-dimensional spatial understanding, our initial design prioritizes visual elements with explicit flat properties to ensure users can intuitively and accurately comprehend the interface layout. To further reinforce interaction intuitiveness, we deliberately avoid visual elements with complex internal topological structures that might cause confusion with file system interactions. Based on this principle, we favor materials that convey a strong sense of flatness—such as cards, plastic sheets, or frosted glass—combined with natural, structureless elements like a cloudless sky or light mist. This strategy achieves visual calm and order while effectively guiding users to focus on the interaction itself rather than on decorative elements.

Distinct Layering of the Interaction Section (Extended) and the Content Section (Pure File Display)

In Chapter 1, we explored the separation between file access and interaction design. However, during the file indexing process, certain scenarios inherently require interactive operations (e.g., bulk file opening). This indicates that, while minimizing interactive elements, it remains necessary to allocate sufficient space for essential interactive processes. Accordingly, this template further defines its functional layering by dividing the screen into two primary regions: the lower “Content Section,” dedicated to file display and indexing to ensure pure information presentation, and the upper “Interaction Section,” which extends to the screen edges to support path navigation, file descriptions, and various auxiliary prompts. This consolidation of interactive components prevents redundant dispersal across the interface, thereby optimizing user experience and ensuring intuitive operation.



Interaction Section

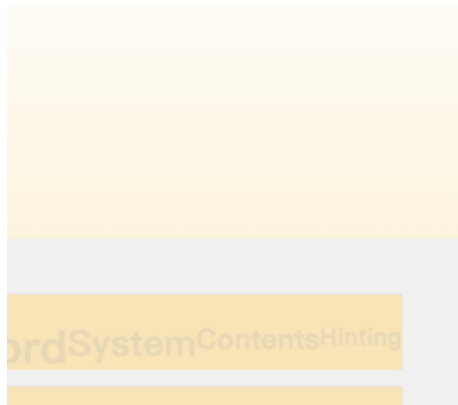


Content Section

To clearly differentiate these areas, we use a full-span layout that extends the Interaction Section horizontally to the screen’s edges, creating a visual sense of boundlessness and extension. This layout naturally accentuates the differences between the upper and lower areas and effectively directs the user’s focus toward the top of the screen to rapidly access essential navigation information and file summaries. Additionally, the Interaction Section serves as the final cue for the content area; an “Info Block” placed under a folder’s title, whether presented as text

or a schematic diagram, enables users to swiftly grasp the folder's contents and purpose without needing to delve deeper.

Regarding color selection, to ensure visual coherence and clear layering between these two sections, we emphasize color continuity so that the Interaction and Content Sections naturally blend yet remain functionally distinct. In line with this principle, we selected imagery of a pale, humid dusk sky and overlaid it with card elements. This soft, unified color scheme not only imparts a sense of balance to the interface but also reinforces the association between file elements and interactive components, ensuring they coherently coexist in the same visual space while maintaining their distinct functionalities. Leveraging this design philosophy, the arrangement of files and folders appears as scattered fragments across the interface, with the Interaction Section acting as an overlaying informational guide that provides clear operational cues to users.



Material close-up

Furthermore, this approach allows further enhancement of card content flexibility to adapt to diverse scenarios. For example, adding a cover image to a folder title can provide more intuitive visual identification, or in specific cases, omitting part of the “info block” can reduce information redundancy, thereby rendering the interface more concise and clear.

As a highly flexible card display style, this approach is particularly well-suited for personal research homepages or resource distribution websites. In such scenarios, rather than showcasing all files directly on the homepage, the focus is placed on differentiating the featured content or navigation information on the first screen. In these cases, the interactive section can be expanded to cover the entire screen, effectively serving as the project cover to enhance the page's hierarchy and guidance. This design not only optimizes the indexing structure for greater clarity, but also renders the user interaction process more intuitive and seamless. In doing so, the interactive area becomes more than just an auxiliary information layer—it acts as a key guiding element within the page's information architecture, directing

users to quickly comprehend the page structure and providing clear navigational cues when needed.

As a further extension of the aforementioned interactive design, this layout mode, when appropriately expanded, still exhibits strong compatibility. For example, we can integrate multiple highly coupled interactive elements into the interaction section, forming a feature-rich control area that blurs the traditional boundaries between content display and interactive control. This design enables users to perform a variety of operations within a single interface, thereby increasing overall efficiency.








In fact, many modern applications already employ similar interaction methods. A notable example is the interface layout of NetEase Cloud Music on the iPad, which exemplifies a highly integrated multi-interaction approach. Its design blends content display with interactive operations, allowing users to control playback, browse playlists, and perform other functions on the same screen without the need for frequent page switching.

However, despite the significant advantages of such a design in certain scenarios, its applicability must be carefully considered. In file management systems that prioritize simplicity and intuitive operation, an excessively complex interactive area can compromise a clear hierarchical structure, making it difficult for users to quickly locate and access desired functions. Therefore, in our design philosophy, the use of multi-level interactive elements should be exercised with restraint to ensure that the interface remains both sufficiently flexible and cleanly intuitive in its operational logic.

Chapter 4: Further Implementation Details

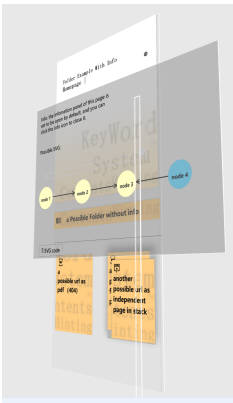
File Stacking Mode and Dynamic Background Changes

To handle files with identical names yet differing in functionality or content more intuitively and efficiently, we introduce the concept of stacking in the interface. This stacking mode aggregates multiple closely related files or functions into a visually unified and conveniently operable card stack. In its initial state, the user sees only the top surface of a single card—thereby preventing interface clutter (see Figure 1) while effectively conveying the logical and functional relationships among these files. By clicking or expanding the stack, users can easily access each file or function and switch between them as needed.

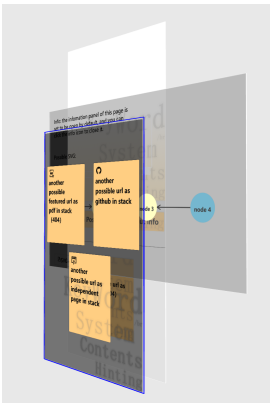
 Shapley.aux	24/01/2025 19:54
 Shapley.fdb_latexmk	24/01/2025 19:54
 Shapley.fls	24/01/2025 19:54
 Shapley.log	24/01/2025 19:54
 Shapley.pdf	24/01/2025 19:54
 Shapley.synctex.gz	24/01/2025 19:54
 Shapley.tex	24/01/2025 19:54

During a single compilation, LaTeX generates a multitude of files with different extensions

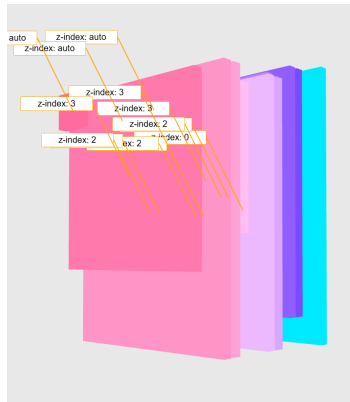
In this stacking interaction process, we introduce dynamic changes in display layering and color to enhance the interface's intuitiveness. For example, when a user interacts with a stack of files, the stack’s z-index shifts from 0 (the background level) to 6 (above all other elements), thereby emphasizing the focus of the currently active stack. Simultaneously, it gradually darkens to avoid visual interference and further optimize the user interaction experience.



Collapsed Stack



Expanded Stack



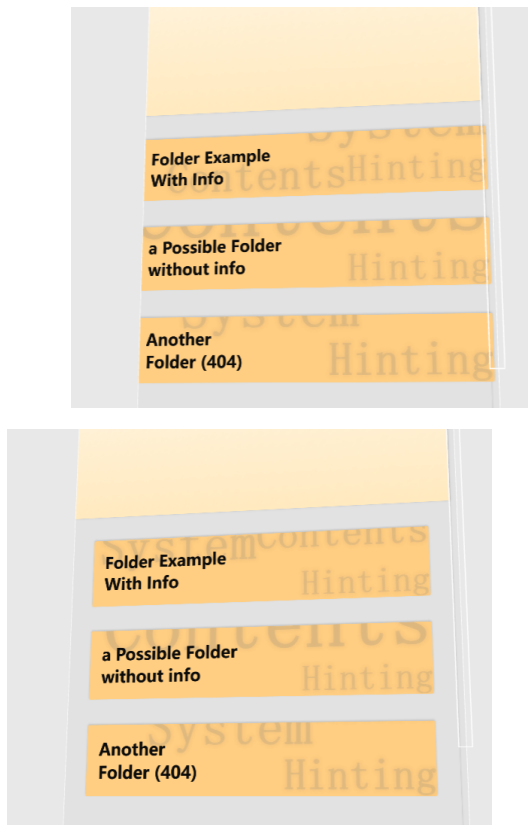
Multiple cards in a collapsed stack are staggered

Template Element Simplification and Intuitive Optimization Strategies

Adaptive Card Width Layout

In the design of this template's interface, we abandon fixed width settings and instead assign each card element a relative width based on the inherent flat intuition of the screen. Specifically, for three different screen size categories, the width of each card is set as a percentage of the screen width. This design approach ensures that, regardless of the display size, the layout remains consistently stable. Moreover, because the overall width of the webpage always aligns with the screen width, horizontal scrolling is eliminated—thereby reducing operational interference.

Especially considering compatibility with external operating system gestures and navigation methods: if left-right swipes are designated for forward/back navigation and edge swipes (or vertical gestures on the side) are allocated for multi-task management, these gestures will not conflict with interactions on interface elements, ensuring smooth operation and a seamless user experience.



When the screen width is less than 480px, the folder card nearly fills the screen

Card Keyword Background and Content Cue Design

To visually enhance the information cues on file cards, this template further introduces a card background keyword system. This system allows keywords or label content to be displayed in a semi-transparent or light-colored manner on the card's background layer, enabling users to intuitively grasp the general attributes or content type of a file without needing to open the card explicitly. This design notably reduces cognitive load, effectively improves file retrieval efficiency, and enriches the interface with layered visual cues.

The keyword system can be applied both to individual cards and to the header's interaction section, thereby achieving a visually coherent experience when managing folders by matching the two. (As shown)



Application Effect of the Keyword System in the Interaction Section

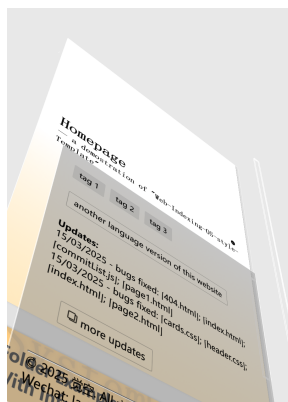
Simplification of Traditional Web Elements and Optimization of Interaction Entry Points

In traditional web design, footers and sidebars are typically cluttered with numerous scattered tools and entry points, which not only increase interface complexity but also run counter to our emphasis on streamlined interaction design. Consequently, this template completely eliminates the dispersed tools and entry points from conventional footers and sidebars, consolidating all critical links and tool access points into a unified information area at the top of the page. This arrangement creates a more centralized, concise, and immediately clear navigation structure. The information area is presented as simple text links, ensuring a consistent intuitive interface.

Version Selection

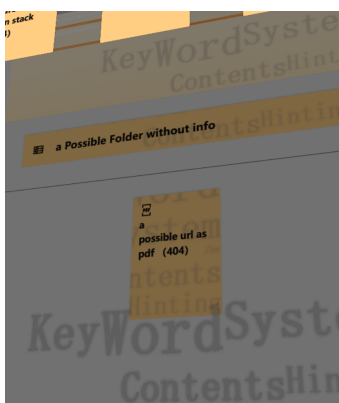
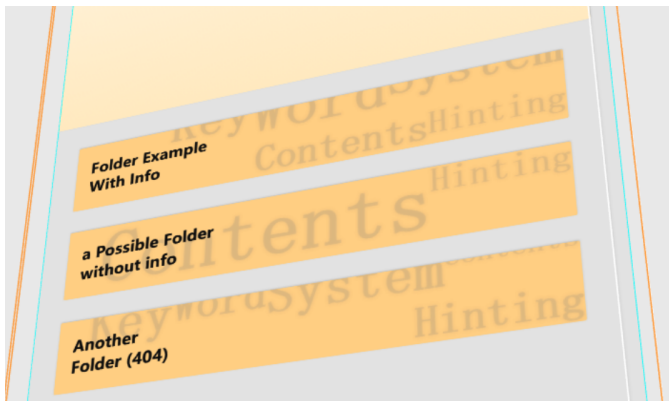
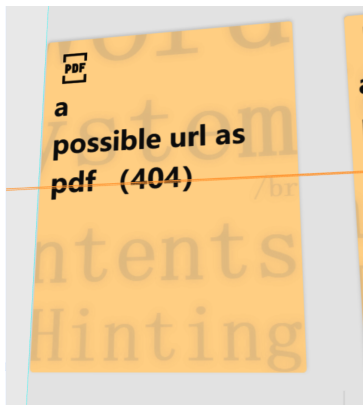
In previous approaches, website version selection (e.g., language or regional settings) was relegated to other areas of the page—often in the footer or sidebar—and typically presented as a collection of scattered links or dropdown menus. While this design technically enables switching, it suffers from fragmentation and complexity. In particular, footers or sidebars can become cluttered with unrelated tools and links, thereby increasing the overall interface complexity. This forces users to dedicate extra cognitive effort to understand the purpose and placement of these options, especially when the page design lacks consistency.

In contrast, the current approach centralizes website version selection in the info area at the top of the page, presenting it as linked navigation tags within a single tab. A map may even be introduced as an interactive index for added clarity. This method avoids the redundant design of footers or sidebars, resulting in a cleaner and more efficient page layout.



Webpage Redirection

In traditional web design, redirection links are often classified as miscellaneous elements primarily due to their lack of standardized design principles and clear visual presentation. They are typically distributed in a scattered manner across various areas of the page—such as sidebars, footers, or the main content—making them neither intuitive nor easily identifiable. This design forces users to expend additional cognitive effort when searching for redirection links, and in some cases, they may even be overlooked. Moreover, the styles and behaviors of these redirection links frequently differ from those of other page elements, which can disrupt the overall visual and interaction flow, further fragmenting the user experience.



Application effects of the keyword system across different cards

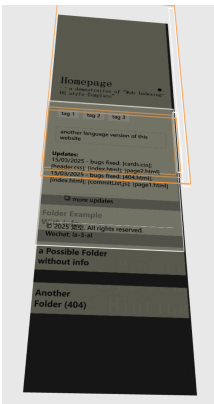
Navigation

Traditional sidebar navigation has long held a prominent position in web design; however, as user needs evolve, its drawbacks have become increasingly apparent. Sidebars typically accommodate a multitude of navigation links and tool access points. Although such designs are functionally comprehensive, they often lead to a complex interface hierarchy.

To resolve these issues, we propose the design concept of "eliminating sidebar navigation" by replacing it with fully structured path displays. All navigation levels are clearly presented within the interaction section at the top of the page, enabling users to immediately identify both their current location and other indexing paths. This design not only simplifies the overall layout but also significantly enhances navigation compatibility with diverse file structures.

Settings

It is especially noteworthy that this template has completely eliminated the traditional “Settings” option, as most clearly demonstrated by the adaptive display mode feature. By detecting whether the user's device or system is operating in day or night mode, the template automatically adjusts the webpage’s color theme, thereby removing the need for manual switching of display modes.



Night mode display effect

Others

Furthermore, if external browsers and operating systems can support the following feature—namely, that when a file is opened, it is initially previewed in a small window mode, and only when the user explicitly requires further access is a new browser tab opened to display it in full—this would further enhance the user experience. In this way, users can flexibly switch between operational modes in different scenarios, meeting the need for quick previews while also providing the convenience of comprehensive access when necessary.



Small window opening design as implemented in Flyme-OS

Font Selection and Visual Experience Optimization

Typography, once considered an elegant art, has gradually been neglected in the information age. However, this template explicitly differentiates between serif and sans-serif fonts for various content areas and enhances visual quality through adjustments in font weight. Additionally, the template utilizes the open-source Source Han Serif and Source Han Sans fonts, which are loaded via CDN to ensure both aesthetic appeal and consistency.

Chapter 5: Template Usage Instructions

The following instructions explain the template's directory structure, the roles of key files, and how to customize and extend the template to meet your requirements.

Project Files

Within the overall project structure, all files and resources must maintain the same hierarchy and paths as shown in the example to ensure that all links and script references work correctly. The HTML files (such as `index.html`, `page1.html`, `page2.html`, etc.) constitute the main structure and content layout of the pages; the `assets/images` folder contains various images and icon resources. To replace any material within this folder, simply add the new files into this directory and update the reference paths in your HTML or CSS accordingly. The `css` folder includes `reset.css`, `styles.css`, and `responsive.css`, which are used to clear default styles, define the basic layout and visual style, and handle adaptations for mobile or large-screen environments. For various interactive logic and dynamic page effects, the

`javascripts` folder contains `infoToggle.js`, `commitList.js`, `copyCommitList.js`, `scrollHandler.js`, `throttle.js`, and `main.js`, which respectively control the expansion of the information panel, update list reading and copying, scroll optimization, and initialization of core interactive functions. If you do not need specific functionalities, you can selectively remove the corresponding scripts and delete the relevant HTML references.

Interaction Section

In the design of the homepage (e.g., `index.html`), the layout of the interaction area is defined by the class names `index-header` and `index-header-content`, allowing direct addition of titles and signatures/paths. The information panel (`infoPanel`) in this area is collapsed by default; clicking the icon on the right expands it to reveal additional updates or redirection links. In some cases, you may include practical clickable tags—such as “Multilingual Versions”—within the information panel, or use JavaScript scripts (e.g., `commitList.js`) to dynamically retrieve and display project update information; convenient one-click copy functionality is also available through scripts like `copyCommitList.js`.

On standard pages (e.g., `page1.html`, `page2.html`, etc.), a similar layout is used; however, the interaction area does not use the same class names as on the homepage but instead is differentiated by names such as `page-header` or `page-header-content`, enabling users to intuitively perceive that these pages are “sub-pages” or “supplementary pages” of the project.

If an entire folder is associated with certain keywords, the keyword background (defined by the `keywords-background` and `keywords` containers) can also be applied at the top of the page. This not only provides visual continuity but also hints at the page theme. The background comprises multiple `` elements representing keywords, with varying font sizes to enhance the weight of the cues.

It is important to note that the expansion and collapse of the information panel are achieved by adding or removing the `open` class, which is triggered and controlled by the `infoToggle.js` script. For more advanced multi-layer interactions, you may introduce additional scripts at the top level of the page or within the information panel, or even integrate third-party services such as Google Analytics or Matomo for analysis and other extended functionalities.

Content Section

Beneath the interaction section, the content section of the page displays various folders or link cards in a compact card format, thereby concentrating visual focus on navigation and core resources.

The fundamental structure of each folder card is composed of an `<a>` tag linking to its respective folder page. Optionally, a folder card may display a folder icon through `` combined with its adaptive rectangular shape, enabling users to intuitively understand its function.

Similarly, file cards—presented in a rectangular format—are used to showcase specific documents or external resources indexed via URL, such as PDFs, GitHub pages, or standalone online pages. The automatic arrangement of these file cards is governed by a `<div class="card-container lecture">`. In addition, each `<a>` tag links to a specific resource, with its link attribute (`target="_blank"`) dictating that the resource opens in a new tab.

In certain scenarios, multiple files may share the same name yet differ in content—for instance, different versions of a document or resources in varied formats. To address this issue, the template provides a file stacking feature. Within a `<div class="card lecture stack">` container, multiple related files are grouped together and differentiated by icons and suffixes, allowing users to swiftly toggle between different versions. While each file retains its own clickable entry, all files are presented on the page as a single card group. A typical application of this design can be seen in the integration of GitHub code repositories and online project documentation; for example, one stacked card might link to a GitHub page (``), while another links to an individual project introduction page (``). Controlled by `cardStack.js`, users can intuitively expand or collapse the stacked content to enhance the organization and readability of the page.

```
<div class="card-container lecture">
```

We achieve the spacing between cards and the screen edge at different screen widths by controlling the padding of the card container; therefore, cards must be placed within the card container element.

```
<a href="#404.html" target="_blank" class="card lecture">
  <h2 class="card-title lecture">
    <br>a<br>possible url as pdf (404)
    <!-- 外部引用: 重复使用的 PDF 图标 -->
  </h2>
  <div class="keywords-background">
    <div class="keywords">
      <span style="font-size: 68px;">Key</span>
      <span style="font-size: 80px;">Word</span>
      <span style="font-size: 64px;">System</span><br>
      <span style="font-size: 50px;">Contents</span>
      <span style="font-size: 42px;">Hinting</span>
    </div>
  </div>
</a>
```

A standard card placed directly within the card-container lecture

```
<div class="card lecture stack" id="cardStack">
  <a href="#404.html" target="_blank" class="card lecture subcard">
    <h2 class="card-title lecture"><br>another<br>possible url as pdf in stack (404)/h2>
    <div class="keywords-background">
      <div class="keywords">
        <span style="font-size: 68px;">Key</span>
        <span style="font-size: 80px;">Word</span>
        <span style="font-size: 64px;">System</span><br>
        <span style="font-size: 50px;">Contents</span>
        <span style="font-size: 42px;">Hinting</span>
      </div>
    </div>
  </a>
  <a href="https://github.com/kakou/Net-Indexing-OS-style-Template" target="_blank" class="card lecture subcard">
    <h2 class="card-title lecture"><br>another<br>possible url as github in stack/h2>
    <div class="keywords-background">
      <div class="keywords">
        <span style="font-size: 68px;">Key</span>
        <span style="font-size: 80px;">Word</span>
        <span style="font-size: 64px;">System</span><br>
        <span style="font-size: 50px;">Contents</span>
        <span style="font-size: 42px;">Hinting</span>
      </div>
    </div>
  </a>
  <a href="https://kakou.github.io/Net-Indexing-OS-style-Template/Project-Introduction/" target="_blank" class="card lecture subcard">
    <h2 class="card-title lecture"><br>another<br>possible url as independent page in stack/h2>
    <div class="keywords-background">
      <div class="keywords">
        <span style="font-size: 68px;">Key</span>
        <span style="font-size: 80px;">Word</span>
        <span style="font-size: 64px;">System</span><br>
        <span style="font-size: 50px;">Contents</span>
        <span style="font-size: 42px;">Hinting</span>
      </div>
    </div>
  </a>
</div>
```

The stack's layer is positioned between the "card-container lecture" and the card. When stacking cards, the repeated keyword background can also be moved to the stack layer to avoid code duplication.

Additionally, all cards support the same keyword background system (`<div class="keywords-background">`) as the interaction section, enabling users to quickly preview content and understand its hierarchical structure.

Others

Regarding customization, whether you want to add more folders and cards or change the visual theme and background, simply modify the corresponding HTML, CSS, or script files. For multilingual support, you can easily employ AI assistance to duplicate and adjust the HTML code on different pages, and switch the homepage via the hyperlinks included in the homepage info section.

When using this template, you typically only need to place the files on any static web server (such as GitHub Pages, Nginx, or Apache) and then preview the corresponding HTML page in your browser. If there are errors with file paths or script logic, icons, scripts, or external links on the page might not load correctly, so you should promptly verify that the references match the actual file locations. Once this is resolved, you can further personalize the template to suit your project needs—

such as adding new pages, including additional cards, expanding JS functionalities, or even removing modules that are not required.

Performance Comparison Test

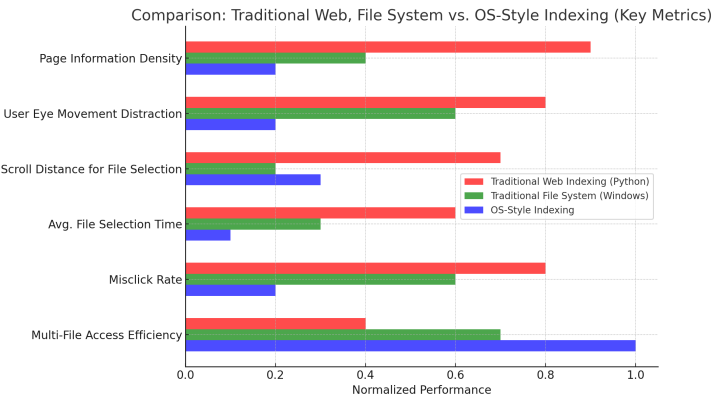
To validate the advantages of OS-Style Indexing, a simple comparison test can be conducted. The test involves performing the following tasks across different indexing methods:

File Selection Time Test: Open five randomly assigned files from a project using traditional indexing (e.g., README navigation, file explorer) and OS-Style Indexing. Record the total time taken.

Misclick Rate Measurement: Count the number of accidental misclicks when selecting files in each indexing system.

Multi-File Access Efficiency: Open and switch between five different files, measuring how smoothly navigation occurs without disrupting workflow.

Information Density Perception: Observe and note down how much additional text, menus, or irrelevant elements appear before finding the required files.



By comparing these results, we can quantitatively assess how OS-Style Indexing improves efficiency and reduces distractions compared to traditional file access methods.