Comprehensive Technical Specification and Strategic Design Framework for Operating System Simulation Portfolios

Executive Summary

The digital portfolio serves as the primary artifact of competence for creative and technical professionals. In an increasingly saturated market, the standard "gallery" portfolio format—characterized by static grids of thumbnails and biographical text—has diminished in impact. This report analyzes a divergent trend: the "Operating System (OS) Simulation" portfolio. Specifically, it deconstructs the architectural, design, and strategic requirements for building a high-fidelity web-based desktop environment, using the Windows XP aesthetic as a primary case study while abstracting principles applicable to any OS metaphor.

This document is designed for professional developers and designers seeking to execute this complex archetype. It moves beyond superficial mimicry to rigorously examine the software engineering challenges involved: implementing window managers, handling global state for z-index stacking, virtualizing file systems, and resolving the paradox of desktop metaphors on mobile devices. The analysis draws upon a review of existing implementations—including the notable work of Mitch Ivin—and industry-standard libraries to provide a definitive "build manual" for a portfolio that serves not just as a showcase of work, but as a proof of technical mastery in itself.

The central thesis of this report is that an OS portfolio must be treated as a **Single Page Application (SPA)** with the complexity of a system utility, not a brochure website. Success depends on the seamless integration of three pillars: **Nostalgic Fidelity** (Visual Design), **System Architecture** (Engineering), and **User Experience** (Performance and Responsiveness).

Part 1: Strategic Definition and User Persona Analysis

1.1 The Paradigm Shift in Portfolio Engineering

The transition from static web pages to dynamic web applications has raised the bar for developer portfolios. The "OS Portfolio" represents a convergence of these trends, functioning as a "meta-portfolio"—a container that houses other projects while being a project itself.

1.1.1 The "Why" Behind the OS Metaphor

The decision to emulate an operating system, particularly a retro one like Windows XP or 98, is rarely purely aesthetic. It serves as a sophisticated signaling mechanism to potential employers.

- **For Visual Designers:** It demonstrates an ability to execute a cohesive design system. Recreating the "Luna" theme of Windows XP requires an obsessive attention to detail—gradients, shadows, border radii, and typography must be pixel-perfect to trigger the "nostalgia" response. As noted in critiques of modern portfolios, generic designs fail to capture attention; a fully realized OS simulation acts as a "Purple Cow" in a field of monochrome resumes.[1]
- **For Front-End Engineers:** It serves as a live technical interview. Building a window manager requires solving complex problems in state management, event propagation, and DOM manipulation. It proves the candidate understands component composition and the browser rendering lifecycle.
- **For Backend Engineers:** The OS metaphor allows for the inclusion of a "Terminal" or "Console" application. This provides a native context to showcase mastery of CLIs, APIs, and server-side logic without forcing a recruiter to read dry text.

1.1.2 Audience-Centric Feature Selection

The critical error in many portfolios is a lack of focus on the target audience.[1] An OS portfolio must be curated.

- **The Recruiter Persona:** They have limited time. The "Boot Sequence" must be fast or skippable. Key information (Resume, Contact) must be accessible via obvious desktop icons, not buried in a simulated file tree.
- **The Engineering Manager Persona:** They will inspect the code. The implementation of the window manager (e.g., using react-rnd vs. raw event listeners) and the cleanliness of the state management (Redux/Zustand) will be scrutinized.
- **The Design Director Persona:** They will look for fidelity. Does the "Start" button have the correct hover state? Do windows cast the correct drop shadow?

1.2 Defining the Archetype: Choosing Your OS

The choice of OS dictates the technical stack and design constraints.

OS Archetype	**Target Persona**	**Key Technical Challenges**	**Aesthetic Goals**
Windows XP (Luna)	Visual Designer / Front-End	Complex CSS gradients, non-standard	Warmth, nostalgia, "Fisher-Price" aesthetic,

		window shapes (rounded top corners), heavily image-based assets.	approachability.
Windows 98/2000	Systems Engineer / Retro Dev	High contrast, bevels, pure CSS styling (no images needed), rigorous grid alignment.	Brutalist, functional, "hacker" credibility, lightweight performance.
macOS (Aqua/Modern)	Product Designer / UX Lead	Glassmorphism (backdrop-filter), complex shadows, smooth Framer Motion animations.	Premium feel, modern interaction patterns, high-fidelity blurring.
Unix/Linux Terminal	Backend / DevOps / Security	Text parsing, history management, keyboard event handling, WebSocket integration.	Efficiency, minimalism, technical depth, command-line expertise.

The "Windows XP" archetype, as exemplified by Mitch Ivin, occupies a unique sweet spot. It is modern enough to support rich media (images, video) but retro enough to evoke strong emotional connections for the generation currently in hiring power.[2]

Part 2: Design System and Visual Architecture

Designing a Windows XP simulation requires a forensic approach to UI reconstruction. We are not designing *for* the web; we are designing *for a desktop environment running inside a web browser*. This requires a strict adherence to the visual language of Microsoft's "Luna" theme.

2.1 The Visual Semiotics of Windows XP

Windows XP was defined by its departure from the grey bevels of Windows 95/98 into a colorful, bitmap-heavy interface.

2.1.1 The Luna Color Palette

The authenticity of the simulation rests entirely on the color palette. The "Default Blue" scheme is the most recognizable. The design system must define these as CSS variables to allow for theme switching (e.g., to "Homestead" Olive Green or "Metallic" Silver).[4]

Table 1: The Definitive Windows XP (Luna Blue) Color Palette Specification

UI Element	CSS Variable Name	Hex Code	Usage Notes
Taskbar Gradient Start	--xp-taskbar-start	#245EDC	The top edge of the taskbar background.
Taskbar Gradient End	--xp-taskbar-end	#3E86F6	The bottom edge; creates the glossy plastic look.
Start Button Green	--xp-start-bg	#3C8F32	The iconic green button.
Start Button Hover	--xp-start-hover	#47C036	Lighter green for interaction state.
Window Title Active	--xp-title-active	#0058EE	Deep blue gradient for focused windows.
Window Title Inactive	--xp-title-inactive	#7890B6	Desaturated blue-grey for background windows.

Window Background	--xp-window-bg	#ECE9D8	The warm beige/grey used for dialog interiors.
Selection Blue	--xp-selection	#316AC5	Used for highlighting selected icons or text.
Border Color	--xp-border	#00138C	Deep blue border used on active windows.

Contextual Insight: Unlike modern "Flat" design, XP relies heavily on borders and gradients to create depth. A simple background-color is rarely sufficient. Buttons, taskbars, and window headers utilize 3-stop vertical gradients to simulate a curved, plastic surface.

2.1.2 Typography and Font Stacks

Typography in Windows XP was transitional. It introduced ClearType, but the UI was heavily optimized for non-antialiased rendering.

- **Primary System Font:** Tahoma. This is the non-negotiable default for UI elements (menus, title bars, dialogs). It was designed by Matthew Carter specifically for on-screen legibility.[6]
- **Secondary Fonts:** Franklin Gothic Medium (often used in the "Welcome" screen) and Trebuchet MS (window title bars in some themes).
- **Fallback Strategy:** Modern Macs and Linux machines do not have Tahoma. The CSS stack must degrade gracefully without breaking the layout metrics.
  - font-family: "Tahoma", "Segoe UI", "Geneva", sans-serif;
  - **Note:** Segoe UI is the Vista/7/8/10/11 font.[8] It is wider than Tahoma. Layout containers must be flexible enough to handle this width difference, or a web-font version of Tahoma must be served (licensing permitting).

2.2 Iconography: The Asset Strategy

The icons of Windows XP (created by Iconfactory) are pixel-art masterpieces with alpha transparency. Using low-quality JPEGs or incorrect PNGs destroys the illusion.

2.2.1 Sourcing vs. Recreation

- **Authentic Bitmap Assets:** The most accurate look comes from extracting original .ico resources. However, these raster images (typically 32x32 or 48x48) will look blurry on

modern 4K/Retina displays.
- **Vector Recreation (Recommended):** The "Pro" approach involves using SVG recreations of classic icons. Libraries like Tabler Icons or specific XP asset packs on Figma/GitHub provide scalable versions that look crisp at any zoom level while retaining the aesthetic.[9]
- **Pixel Art Upscaling:** If using original bitmaps, apply image-rendering: pixelated in CSS. This preserves the sharp, blocky edges of the pixel art rather than applying a bilinear blur filter, maintaining the retro charm.[11]

2.3 Audio Design (Soundscapes)

Sound is a powerful but dangerous tool in web design. Windows XP is famous for its startup chime and click sounds.

- **Implementation:** Use the HTML5 <audio> API or a React wrapper like use-sound.
- **The Autoplay Barrier:** Modern browsers block audio autoplay. You *must* implement a "Login Screen" or "Boot Menu" that requires a user click. This user interaction grants the browser permission to play audio, allowing the startup chime to trigger immediately after login.[12]
- **Sound Palette:**
  - startup.wav: The "Windows XP Startup" sound.
  - click.wav: Subtle click for navigation.
  - chord.wav: For error modals.
  - *Constraint:* Keep file sizes tiny (MP3/OGG) to avoid blocking the main thread.

Part 3: Frontend Engineering: The Window Manager Architecture

This section details the core engineering challenge: building a functional Window Manager in React. This separates a "website that looks like XP" from a "web application that behaves like XP."

3.1 The Component Hierarchy

The application structure should mirror the OS architecture.

src/

├── components/

│ ├── os/

```
| | ├── Desktop.tsx // The main container, handles background & icons
| | ├── Taskbar.tsx // Fixed bottom bar, holds Start Button & Window List
| | ├── StartMenu.tsx // The recursive menu system
| | ├── Window.tsx // The HOC wrapper for draggable/resizable behavior
| | └── WindowManager.tsx // The orchestrator of z-index and state
| ├── apps/
| | ├── Notepad.tsx
| | ├── Browser.tsx
| | ├── Winamp.tsx
| | └── Portfolio.tsx // The core content app
| └── ui/
| ├── Button.tsx // Styled XP buttons
| └── Scrollbar.tsx // Custom scrollbar implementation (XP style)
├── hooks/
| ├── useWindowManager.ts // Custom hook for OS logic
| └── useDrag.ts // Hook for handling drag physics
├── store/
| └── systemStore.ts // Zustand/Redux store for global state
└── utils/
└── fileSystem.ts // Virtual file system logic
```

3.2 State Management: The "OS" Store

The state requirements for a window manager are complex. You must track:

1. **Window Registry:** Which apps are currently open?
2. **Window State:** Is App X minimized, maximized, or restored?
3. **Z-Index Stacking:** Which window is on top?

4. **Focus History:** If the top window closes, which one gets focus next?

Recommendation: Zustand While React Context is built-in, it can lead to unnecessary re-renders when managing high-frequency updates (like window dragging coordinates). **Zustand** is recommended for 2025 due to its transient update capabilities and minimal boilerplate.[14]

Table 2: State Model for Window Manager

State Property	Type	Description
windows	Array<WindowObj>	List of open window instances.
activeWindowId	string	ID of the currently focused window (highest Z-index).
windowOrder	Array<string>	An array of IDs representing the Z-stack order.
wallpaper	string	Current background image URL.
theme	ThemeObject	Current active CSS variables (Luna/Silver/Olive).

3.2.1 The WindowObj Interface

TypeScript

```
interface WindowObj {
  id: string;          // Unique UUID
  appId: string;       // ID of the app type (e.g., 'notepad')
  title: string;       // Current window title
  icon: string;        // Window icon
  isOpen: boolean;
  isMinimized: boolean;
  isMaximized: boolean;
  position: { x: number, y: number };
  size: { width: number, height: number };
  zIndex: number;
  contentProps: any;   // Props passed to the internal App component
}
```

3.3 The Window Component: Drag, Resize, and Focus

Implementing drag-and-drop from scratch is error-prone. The react-rnd library is the industry standard for this specific use case, combining react-draggable and react-resizable.[16]

3.3.1 Integrating react-rnd

The Window component wraps the specific App content.

TypeScript

```typescript
import { Rnd } from 'react-rnd';

const Window = ({ windowState, dispatch }) => {
  return (
    <Rnd
      default={{
        x: windowState.position.x,
        y: windowState.position.y,
        width: windowState.size.width,
        height: windowState.size.height,
      }}
      minWidth={200}
      minHeight={150}
      bounds="parent" // Prevents dragging off the desktop
      onDragStart={() => dispatch({ type: 'FOCUS_WINDOW', id: windowState.id })}
      onMouseDown={() => dispatch({ type: 'FOCUS_WINDOW', id: windowState.id })}
      dragHandleClassName="window-title-bar" // Only drag by title bar
    >
      <div className={`xp-window ${isActive? 'active' : 'inactive'}`}>
        <TitleBar />
        <WindowContent>
          {/* App Component Rendered Here */}
        </WindowContent>
      </div>
    </Rnd>
  );
};
```

Key Insight: The onDragStart and onMouseDown handlers are critical. In a real OS, clicking *anywhere* on a window brings it to the front. This must be manually wired to the Z-index state manager.[18]

3.4 The Z-Index Stacking Context Strategy

Z-index management is the most common failure point. A naive approach (z-index: 100 for active) fails when you have multiple windows.

The Stack Algorithm:

1. Maintain an array of window IDs: ``.
2. The index in the array corresponds to the visual stack.
3. When win_A is clicked:
   - Remove win_A from the array.
   - Push win_A to the end of the array.
4. Render windows mapping their array index to their CSS z-index property (plus a base value, e.g., 100).
5. **Taskbar & Start Menu:** Always sit at z-index: 9999 to remain above all windows.[19]

Part 4: System Applications and Portfolio Content

The OS is the container; the Apps are the content. This section details how to populate the portfolio with meaningful demonstrations of skill.

4.1 The "Project Explorer" (The Virtual File System)

Instead of a simple "Projects" page, implement a "File Explorer" that mimics Windows Explorer.

- **Virtual File System (VFS):** Create a JSON object representing the directory structure.
  ```JSON
  {
    "root": {
  ```

```
    "My Documents": {
      "Projects": {
        "ecommerce-app.link": { "type": "link", "url": "..." },
        "backend-api.txt": { "type": "text", "content": "..." }
      },
      "Resume.pdf": { "type": "pdf", "src": "..." }
    }
  }
}
```

- **Navigation Logic:** Breadcrumbs and Back/Forward buttons must traverse this JSON tree.
- **Skill Showcase:** This demonstrates knowledge of tree data structures and recursion (for rendering the folder view).[1]

4.2 The Terminal Emulator (Backend Showcase)

For backend engineers, a static portfolio is insufficient. A working Terminal app is the ultimate flex.

- **Command Parsing:** Implement a parser that takes string input, tokenizes it (splitting by spaces), and executes functions mapped to commands (ls, cd, cat, whoami).
- **History Stack:** Use an array to store previous commands, accessible via Up/Down arrow keys (event listeners on keydown).
- **Easter Eggs:** Include sudo (returns "Permission denied: you are not admin"), rm -rf / (triggers a fake blue screen of death or error message).
- **Live API Integration:** If showcasing backend skills, have the terminal actually query your real backend API. curl /api/status could return real-time health checks of your server.[21]

4.3 The "Web Browser" (Iframe Management)

A common desire is to show live web projects inside the OS.

- **The Iframe Problem:** Most modern sites (Google, Facebook, etc.) send X-Frame-Options: DENY headers, preventing embedding.
- **The Solution:**
  1. **Self-Hosted:** Only embed projects you control and have configured to allow framing.
  2. **Screenshot Proxy:** For external links, display a cached screenshot or a "This website prevents embedding" error page with a "Open in New Tab" button.
  3. **Address Bar Logic:** The browser address bar should function, updating the iframe src. This requires validating URLs and handling loading states.[23]

4.4 NotePad / Text Editor

A simple text editor allows you to display your Bio, Experience, or "Readme.txt" files.

- **Rich Text vs. Plain Text:** Stick to plain text (<textarea>) or a simple pre-formatted div to maintain the "Notepad" authenticity. Wordpad would require a Rich Text Editor implementation (like Draft.js), which might be overkill unless targeting a specialized role.[24]

Part 5: Mobile Responsiveness and Adaptive UX

This is the most challenging aspect. A desktop metaphor inherently conflicts with mobile UX patterns (hover states, window management, small screens). Mitch Ivin's portfolio is notable because it "rebuilds everything" for mobile rather than just shrinking it.[3]

5.1 The "Desktop on Mobile" Paradox

Simply shrinking a 1024x768 desktop to a 375px iPhone screen results in unusable touch targets and unreadable text. "Pinching and zooming" is not an acceptable user experience for a professional portfolio.

5.2 Strategy 1: The "Tablet Mode" Transformation (Recommended)

Windows 10/11 handles this by switching to "Tablet Mode." We can emulate this.

- **Windows become Full Screen:** On mobile breakpoints (< 768px), the react-rnd component should disable dragging/resizing and force width: 100%, height: calc(100% - taskbarHeight).
- **Taskbar becomes Bottom Navigation:** The Start Menu becomes a full-screen drawer (like the Android App Drawer or iOS App Library).
- **Window Controls:** The "Minimize/Maximize" buttons can be hidden, leaving only "Close."
- **CSS Media Queries:**

  CSS
  ```css
  @media (max-width: 768px) {
   .xp-window {
     top: 0!important;
     left: 0!important;
     width: 100%!important;
     height: calc(100vh - 40px)!important;
     transform: none!important; /* Disable manual positioning */
   }
   .window-resize-handle {
     display: none; /* Disable resizing */
   }
  }
  ```

5.3 Strategy 2: The "Grid View" Fallback

Instead of a desktop with free-floating icons, mobile users see a responsive grid of large, tappable icons (similar to the iOS home screen).

- **Navigation:** Clicking an icon opens the "App" in a standard modal route, completely bypassing the window manager logic.
- **Pros:** Better usability, adherence to mobile mental models.
- **Cons:** Breaks the "OS" illusion slightly.

5.4 Handling Touch Events

If preserving drag-and-drop on mobile (e.g., for tablets):

- Ensure react-rnd or your drag library supports onTouchStart / onTouchMove.
- **Touch Action:** Use CSS touch-action: none; on draggable elements to prevent the browser from scrolling the page when the user tries to drag a window.[25]

Part 6: Performance Optimization and Engineering Standards

A portfolio that crashes the browser or takes 10 seconds to load is a failed portfolio, regardless of how beautiful it is.

6.1 Bundle Size Management

An OS simulation is asset-heavy.

- **Code Splitting:** Use React.lazy and Suspense. The code for "Minesweeper" or "Winamp" should not be loaded until the user opens that app.
  TypeScript
  ```
  const Winamp = React.lazy(() => import('./apps/Winamp'));
  ```
- **Asset Optimization:** Convert wallpapers to WebP/AVIF formats. Compress audio files aggressively.
- **Virtualization:** If your "File Explorer" displays hundreds of items, use react-window to virtualize the list, ensuring 60fps scrolling performance even with large DOM trees.[26]

6.2 Accessibility (A11y)

Simulated UIs are often accessibility nightmares. To be a "Pro Developer," you must address this.

- **Keyboard Navigation:** Users should be able to tab through desktop icons and Taskbar entries.
- **Focus Management:** When a window opens, focus should move to it. When it closes, focus should return to the triggering icon. Use the useFocusTrap hook for active windows

to keep keyboard navigation contained.
- **ARIA Roles:** Use role="application" for the desktop container, role="button" for icons, and aria-label for icon-only buttons (like Start).[8]

Part 7: Implementation Roadmap

Phase 1: Setup & Foundations (Week 1)

1. **Initialize Project:** React + TypeScript + Vite (faster than CRA).
2. **Asset Gathering:** Download Windows XP assets (icons, sounds, wallpapers). Process them (SVG conversion, compression).
3. **Global Styles:** Set up CSS Variables for the Luna theme (Table 1).
4. **Store Setup:** Configure Zustand store for window management.

Phase 2: Core OS Mechanics (Week 2)

1. **Desktop Component:** Grid layout for icons, background image rendering.
2. **Taskbar Component:** Clock (using setInterval), Start Button toggle logic.
3. **Window Manager:** Implement react-rnd, Z-index sorting logic, and Min/Max/Close state toggling.

Phase 3: Application Development (Week 3)

1. **File Explorer:** Build the recursive folder viewer.
2. **Content Population:** Create the "Resume" and "Projects" data structures.
3. **Special Apps:** Build the Terminal (if Backend) or Paint (if Frontend).

Phase 4: Polish & Optimization (Week 4)

1. **Mobile Response:** Implement the "Tablet Mode" CSS overrides.
2. **Boot Sequence:** Create a fake BIOS/Login screen overlay.
3. **Sound Integration:** Wire up click/startup sounds with user interaction checks.
4. **Testing:** Audit with Lighthouse. Fix A11y contrast issues.

Part 8: Conclusion

The creation of a Windows XP-style OS portfolio is a significant undertaking that transcends standard web design. It requires the developer to wear multiple hats: **System Architect** (designing the window manager), **UI Designer** (recreating the Luna aesthetic), and **Product Engineer** (ensuring performance and responsiveness).

By strictly adhering to the "Pro Developer" mindset—prioritizing architectural robustness, type safety with TypeScript, and thoughtful mobile adaptation—this project becomes more than a novelty. It becomes a definitive proof of competence. The result is a portfolio that does not merely *tell* a recruiter you can build complex web applications; it *shows* them by becoming one itself.

The success of this project lies not in the 100% pixel-perfect recreation of every Windows quirk, but in the intelligent adaptation of those metaphors to the web medium. It is a balance of nostalgia and modern usability, proving that you understand the past well enough to reinvent it for the future.

Technical Appendix

Appendix A: Key Libraries

- **Window Management:** react-rnd (Best for resize/drag), dnd-kit (Alternative).
- **State:** zustand (Recommended), redux-toolkit.
- **Styling:** styled-components (Component-scoped styles) or Tailwind CSS (Utility-first).
- **Icons:** tabler-icons (General), react95 (Specific retro assets).
- **Virtualization:** react-window (For file lists).

Appendix B: Folder Structure Example

```
/src
 /assets/icons/xp/ (SVG converted icons)
 /components/os/WindowManager/
   Window.tsx
   TitleBar.tsx
   WindowContext.ts
 /hooks/
   useOS.ts (Zustand hook)
   useTime.ts (Clock logic)
 /apps/
   Terminal/
     Terminal.tsx
     commands.ts
```

Works cited

1. [Hindi (auto-generated)] Stop Copying Portfolios — Learn How to Think Like a Pro Developer [DownSub.com].srt

2. MitchIvin XP - UI UX Showcase, accessed January 24, 2026, https://uiuxshowcase.com/portfolio/mitchivin-xp/
3. Show HN: I recreated Windows XP as my portfolio - Brian Lovin, accessed January 24, 2026, https://brianlovin.com/hn/45154609
4. Windows XP Color Palette, accessed January 24, 2026, https://www.color-hex.com/color-palette/44345
5. Windows System Colours (by OS) - GitHub Gist, accessed January 24, 2026, https://gist.github.com/zaxbux/64b5a88e2e390fb8f8d24eb1736f71e0
6. Default Fonts in Desktop & Mobile Operating Systems | Scott Granneman, accessed January 24, 2026, https://granneman.com/webdev/coding/css/fonts-and-formatting/default-fonts
7. Windows XP Default Font? - Dell Technologies, accessed January 24, 2026, https://www.dell.com/community/en/conversations/windows-general/windows-xp-default-font/647e42d0f4ccf8a8dede7c09
8. Fonts - Win32 apps - Microsoft Learn, accessed January 24, 2026, https://learn.microsoft.com/en-us/windows/win32/uxguide/vis-fonts
9. Free Windows Xp Icons - Free Download in SVG, PNG - IconScout, accessed January 24, 2026, https://iconscout.com/icons/windows-xp?price=free
10. Windows Xp Vector SVG Icon - SVG Repo, accessed January 24, 2026, https://www.svgrepo.com/svg/306974/windows-xp
11. 15 Backend Project Ideas to Enhance Your Developer Portfolio - AlmaBetter, accessed January 24, 2026, https://www.almabetter.com/bytes/articles/backend-projects
12. Windows XP (FREE sound fx WAV files) : r/SP404 - Reddit, accessed January 24, 2026, https://www.reddit.com/r/SP404/comments/1ij7p2l/windows_xp_free_sound_fx_wav_files/
13. Windows Startsounds - Winhistory.de, accessed January 24, 2026, https://www.winhistory.de/more/winstart/winstart_en.htm
14. Redux vs. Context.Provider: Choosing State Management in React Applications, accessed January 24, 2026, https://dev.to/dosht/redux-vs-contextprovider-choosing-state-management-in-react-applications-6bm
15. Advanced State Management in React: When to Use Context, Redux, or Zustand, accessed January 24, 2026, https://dev.to/johnschibelli/advanced-state-management-in-react-when-to-use-context-redux-or-zustand-kgo
16. Top 5 Drag-and-Drop Libraries for React in 2026 - Puck, accessed January 24, 2026, https://puckeditor.com/blog/top-5-drag-and-drop-libraries-for-react
17. react-rnd - NPM, accessed January 24, 2026, https://www.npmjs.com/package/react-rnd
18. JavaScript multiple draggable DIV windows, zIndex on focus - Stack Overflow, accessed January 24, 2026, https://stackoverflow.com/questions/8673768/javascript-multiple-draggable-div-windows-zindex-on-focus

19. Manage layers and z-index in React applications effectively - GitHub, accessed January 24, 2026, https://github.com/giuseppeg/react-layers-manager
20. How to manage Z-index with 4 layers in React? - Stack Overflow, accessed January 24, 2026, https://stackoverflow.com/questions/62331872/how-to-manage-z-index-with-4-layers-in-react
21. iamdhakrey/terminal-portfolio: A modern, terminal-style portfolio website that's fully configurable through a single configuration file. Perfect for developers who want to showcase their skills and projects in a unique way. - GitHub, accessed January 24, 2026, https://github.com/iamdhakrey/terminal-portfolio
22. Standing Out with Terminal-Based SSH Portfolio | by Kaustubh Patange - Medium, accessed January 24, 2026, https://kaustubhpatange.medium.com/standing-out-with-terminal-based-ssh-portfolio-aa7b6a2eb1ad
23. Windows XP - A React recreation of iconic operating system. - GitHub, accessed January 24, 2026, https://github.com/shalkauskas/windows
24. qqkookie/XPAccApps: Simple Windows XP Accessories ported from React OS application like calculator, notepad, taskmgr - GitHub, accessed January 24, 2026, https://github.com/qqkookie/XPAccApps
25. HTML Drag And Drop On Mobile Devices - Stack Overflow, accessed January 24, 2026, https://stackoverflow.com/questions/3172100/html-drag-and-drop-on-mobile-devices
26. Getting to Grips with react-window - DigitalOcean, accessed January 24, 2026, https://www.digitalocean.com/community/tutorials/react-react-window
27. bvaughn/react-window: React components for efficiently rendering large lists and tabular data - GitHub, accessed January 24, 2026, https://github.com/bvaughn/react-window
28. Roving focus in react with custom hooks - DEV Community, accessed January 24, 2026, https://dev.to/rafi993/roving-focus-in-react-with-custom-hooks-1ln