This is a comprehensive, step-by-step implementation plan designed to guide you from architectural setup to final deployment. This phased approach prioritizes building the **foundation** before tackling the complex features.

🚀 Step-by-Step Implementation Plan

Phase 1: Architecture & Core Kernel Setup (The Blueprint)

This phase establishes the non-visual Zustand kernel and the core window management logic.

Step	Goal	Details
1. Project Setup	Initialize the React + vite ts project.	Create a new React + vite ts app with TypeScript. Install required dependencies: Zustand and custom scss structure.
2. Define Interfaces	Create the core data contracts.	Define the WindowState and ICommand TypeScript interfaces. This is the **most critical** step for architectural integrity.
3. Zustand Store Setup	Build the OS Kernel.	Create the Zustand store. Define the initial windows slice state (an empty array of WindowState). Set up reducers for the core actions: OPEN_WINDOW, CLOSE_WINDOW, SET_ACTIVE_WINDOW (for zIndex), DRAG_WINDOW, and RESIZE_WINDOW.
4. Basic Layout	Create the visual container.	Create the <Desktop> component to render the wallpaper and the main <OSContainer>. Use the isMobile check here to conditionally render the full OS view or the simplified mobile view.
5. The Taskbar	Create the system tray.	Implement the <Taskbar>

		component, which subscribes to the Zustand windows slice to display icons for all open apps.

Phase 2: The Core Window & First Apps (The Foundation)

This phase builds the single reusable window and the most essential functional app.

Step	Goal	Details
6. The Generic Window	Implement the Window Manager.	Create the <Window> component. It must accept WindowState props and render its children. Implement the Mousedown and Mousemove logic for **drag functionality**.
7. Z-Index Management	Implement layer ordering.	When a user clicks *anywhere* on a window, dispatch the SET_ACTIVE_WINDOW action to assign the highest zIndex to that window, bringing it to the front.
8. The Terminal App	Implement the first interactive app.	Create the <Terminal> component. Define the ICommand[] array containing help and open. Implement the input field logic to execute commands and update the displayed output.
9. Launch Logic	Connect Terminal to the OS.	The Terminal's open <appId> command must dispatch the OPEN_WINDOW Zustand action with the correct WindowState payload.
10. Core Content Apps	Create placeholder content.	Create simple, static components for <AboutMeContent> and <ProjectsContent>. (These are the components rendered *inside* the generic window).

Phase 3: High-Impact Applications (The Simulation)

Focus on the illusion of complexity by leveraging <iframe> and state reading.

Step	Goal	Details
11. The Browser App	Implement web browsing.	Create the <Browser> component. Include an address bar (<input>) and an <iframe>. The address bar value controls the <iframe>'s src. Implement the Google Search URL trick (/search?q=...).
12. Code Viewer (VS Code)	Showcase source code.	Create the <CodeViewer> component. Use an <iframe> set to a GitHub Dev URL (e.g., github.dev/your-repo) or use react-syntax-highlighter to display a raw file.
13. File Explorer	Implement navigation state.	Define the static JavaScript object for the file system. Create the <FileExplorer> component. Use Zustand state to track the user's currentPath and dynamically render folder contents.
14. Task Manager	Implement the Process View.	Create the <TaskManager> component. Use a Zustand selector to **read** the list of all WindowState objects. Add a button that dispatches the CLOSE_WINDOW action.
15. Settings & Theming	Implement state persistence.	Create the <Settings> app. Add a Theme Toggle. Implement a global theme selector in Zustand, and use **localStorage** to persist the theme choice across sessions.

Phase 4: Responsiveness & Final Polish

The final step is to ensure a smooth transition across devices and optimize performance.

Step	Goal	Details
16. Mobile Detection	Implement the UI switch.	Use a React Hook (useMediaQuery) to detect the viewport size (< 768px for portrait/mobile).
17. Mobile UI Implementation	Implement the alternate view.	If isMobile is true, render the simplified **Start Screen**. Use a full-screen modal for the **Start Menu**. Icons on the home screen transition to a full-viewport, scrollable content page when clicked. **(Hide all drag/resize logic).**
18. Polish & Styles	Refine the look and feel.	Apply Windows 11 aesthetics (rounded corners, shadows, subtle transparency/blur effects) using custom scss structure.
19. Documentation & README	Prepare for application.	Write a professional README.md clearly explaining the **Pseudo-OS Architecture**, Zustand usage, and the rationale behind your mobile responsiveness strategy.
20. Deployment	Launch the project.	Deploy the React+vite ts app to **Vercel** or an equivalent service.

References - Vova Ushenko, Dustin Brett

Features and Screens

Boot Screen

- Show loading screen similar to windows with progress bar / loader
- Ask for account login - input field that takes in username only (no password required)

- If system admin (Natsu is the system admin), then enable all features. If any other user, allow reading only:
    - About-Me, Contact-Me, Projects, Skills
    - */portfolio* route
    - Browser search
    - Music player
    - Notepad
    - Socials and MS Office without signed in
-

UI Layer

- Desktop environment
    - Icons - vscode, resume (downloadable), projects, About-Me with windows folder icon, notepad, portfolio
    - Personalization:
        - Change background image with upload functionality
        - Change theme -
            - Dark and light
            - Set translucency - keep it simply by using a toggle
            - Custom theme - set background, color, volume level, desktop icon shape (square, circular, water droplet etc) and cursor type (optional)
            - Desktop icon
        - Taskbar
            - Show/hide search
            - Change alignment - snap to bottom, left, top and right
        - Start Menu
            - Layout - grid and list
            - Show / hide recommended apps
        -
    -
- Start Menu
    - Default - about-me, contact-me, projects, skills, portfolio page
    - Recommended - github, vscode, browser, command prompt, notepad
    - A *More* button with embedded links to - calculator, music player, word, powerpoint, outlook, linkedIn, microsoft store. Use microsoft office 365 for easy integration or like vova did, use the corresponding thumbnail
    - Settings - Date & time, Personalization, Accounts, Volume, Display.
    - **Note** - MS Office and socials are optional
- TaskBar
    - Windows icon
    - Search Bar box
    - Pinned apps - file explorer that opens my portfolio (refer Vova's portfolio), browser, vscode, notepad, github
- Layout
    - <OSContainer /> - Contains <Desktop />, <Taskbar /> (depending on where it's positioned)

- Portfolio page - a popular layout
    - Use a good loader (probably an Indic design)
    - Projects
    - Skills
    - About Me
    - Contact Me
    - Links to LinkedIn, Medium, Github, Medium

Shut down Screen

- Display closing each window in a list
- Progress bar / Loader to indicate shutting down
- Display a power button screen which when clicked redirects to login screen
-

Customizations are not persisted for existing user logins in this application.
Customizations are deleted when the user logs out

Architecture and Interface Definitions

State Management

- Auth/User slice
    - `username: string` - Stores the name entered at login screen (admin is *Natsu*)
    - `isAdmin: boolean` - Whether logged in user is admin
    -
- WindowState slice
    - `iWindow: WindowState[]` for any window component that opens within <Desktop />
    - Reducers - to set each property of WindowState interface
- System UI slice
    - `taskbarAlignment: TaskbarAlignmentType` - “top”, “right”, “bottom”, “left”
    - `isSearchVisible: boolean;` - Whether the search bar box in Taskbar is visible
    - `startMenuOpen: boolean;` - Whether start menu is open or not
    - `startMenuLayout: StartMenuLayoutType;` - “grid” | “list”
    - `activeBackground: string;` URL of current background image
    - `showRecommendedApps: boolean;` - Toggles the display of recommended apps within Start Menu
    - `volumeLevel: number;` - Stores the global volume level
    - `showMoreIcons: boolean;` - Windows icons on the right side of taskbar (wifi, sound, etc)
- Boot and Shutdown Slice
    - `bootStatus: BootStatusType;` - When the OS
    - `currentOperation: string;` - Name of operating to be displayed on the screen
    - Reducer to update bootStatus which is essentially a state machine
    -

-

Interface definitions in [storeTypes.ts]

```
export interface User {
     isAdmin: boolean; // Whether the user is admin or not
     username: string; // Username of logged in user (e.g. Natsu)
}
```

```
export interface WindowState {  id: string; // Unique identifier for the Window Instance (e.g., browser-3)
      title: string; // Title displayed on window's title bar
      windowName: string; // Identifies specific React component to render inside the window frame (e.g., 'BrowserApp', 'Notepad')
      isMaximized: boolean;
      position: {x: number; y: number}; // x,y co-ordinates
      zIndex: number; // For stacking
      size: {width: number; height: number}; // Width of the window and height of the window
      customTheme?: CustomTheme; // For personalization
      isOpen: boolean; // Whether the window is opened or not
      snapPosition?: SnapPositionType; // Mobile-only
}
```

```
export type IconShapeType = "water-droplet" | "circle" | "square";
export type SnapPositionType =
  | 'fullscreen'
  | 'left-half'
  | 'right-half'
  | 'grid-cell'
  | 'top-half'
  | 'bottom-half';

export interface CustomTheme {
      fontColor: string; // rgba
      iconShape: IconShapeType;
      bgColor: string; // rgba
}
```

```
export interface TaskbarAlignmentType = "top" | "right" | "bottom" | "left"
export interface StartMenuLayoutType = “grid”  | “list”;
export interface BootStatusType = “ON” | “DISPLAY_SHUTDOWN_SCREEN” |
“DISPLAY_BOOT_SCREEN” | “OFF”;
```

Constant definitions

```
export const COMMANDS = {
"help": "help", // Display info about all commands; args: name of the
command
"open": "open", // Open a window; args: an easily readable window name
"close": "close", // Close a window; args: an easily readable window name
"Hide": "hide", // Minimize a window; args: an easily readable window name
"Show": "show", // Maximize a window; args: an easily readable window name
"Activate": "activate" // Set a window's zIndex to the highest; args: an
easily readable window name
"Resize": "resize" // Resize a window; args: an easily readable window
name, x coordinate, y coordinate
"Shut-down": "shutdown", // Close all active windows and shut down the OS ;
args: none
"Reboot": "reboot", // Close all active windows and display login screen
}
```

Easily readable window name format:
`<appCategory>-<appName>-<windowId>`
If user enters either <appCategory> or <appCategory>-<appName> then display a list of open windows in that category with that name and window id. Then suggest the user to pick one.

Apps list for commands:

App Category	App Name
browser	firefox
browser	edge
terminal	cp
file-explorer	natsu (with icon for Windows Folder)
tasks	task-manager

settings	settings
notes	notepad
portfolio	about-us
portfolio	contact-me
portfolio	projects
portfolio	skills
music	music-player

Resizing and Dragging logic

- Desktop + Tablet-Landscape: Dragging/resizing (bigger screens, mouse/trackpad)
- Tablet-Portrait + Mobile: Snapping is better UX (smaller screens, touch frustration)

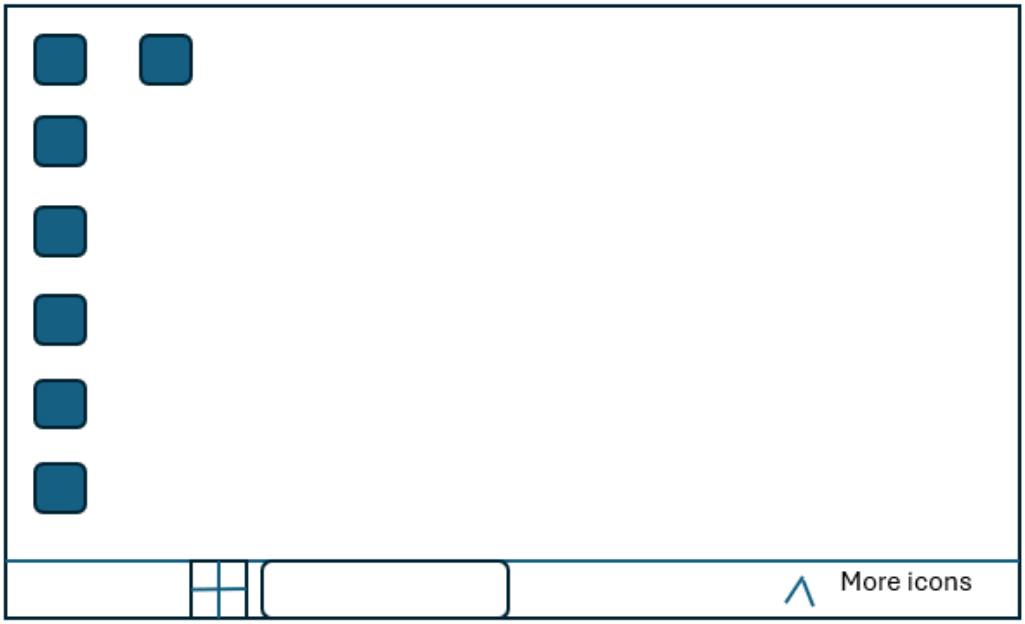
This approach improves performance in mobile devices in lighthouse. The `useMediaQuery` hook, which you planned in your architecture, is the key to toggling between these two modes.

```
const isMobile = useMediaQuery('(max-width: 768px)');
```

Package - react-rnd

UI

Basic Layout



Design System - Fluent

https://github.com/Codrax/Codrut-Fluent-Design-System

Link - https://fluent2.microsoft.design/

- Color
  - Neutral colors - Use for visual hierarchy
  - Shared colors - MS has this but I don't know whether I need it.
  - Semantic colors - Communicate feedback, status or urgency – at-a-glance information.
  - Brand colors - In my case, for MS suite.
  - Interaction states - Generally, a component will get darker as someone interacts with it, from the lightest rest state, to a darker hover, all the way to the darkest selected state. For focus states, the color of the control does not change, but the container gets a thicker stroke to create clear visual distinctions between mouse and keyboard interactions.
  - **To Do** -
    - Get all the rgba values of colors for this project with appropriate sass variable names
    - For light and dark themes
    - For customizable themes - I don't know how to implement this. If I offer a color palette, how'd it change the complexity of the project. Or perhaps,

        - some other themes beyond light and dark? Help me here
        - Solved by fluent package https://www.npmjs.com/package/@fluentui/tokens/v/0.0.0-nightly-20250103-0406.1
    -
- Elevation using shadows
    - $Shadow-2 has 2 pixel blur
    - $shadow-64 has 64 pixel blur
    - **To Do:**
        - What shadows are relevant for this project?
        - Shadows in a theme
        - Solved by fluent package https://www.npmjs.com/package/@fluentui/tokens/v/0.0.0-nightly-20250103-0406.1
        -
    -

- Iconography - https://github.com/microsoft/fluentui-system-icons
    - **To Do**:
        - What icons for this project?
        - Regular / filled - which one is required?
        - Are System Icons (mentioned in https://fluent2.microsoft.design/iconography) required?
        - How many File type icons to use?
    -
- Layout
    - Spacing and proximity – Elements in a design that are in close proximity are seen as being meaningfully related. As more space is added between elements, their perceived relationship weakens.
    - Create / use the existing global spacing ramp
    - Grid / Flexbox - I don't think so many grid types which are mentioned in that Fluent design link required for this project. Just stick to grid / flexbox
    - Responsive design (while these are -
        - $breakpoint-small <= 479 px
        - $breakpoint-medium <= 639 px
        - $breakpoint-large <= 1023 px
        - $breakpint-x-large >= 1024 px
        - $breakpint-xx-large >= 1366 px
        - $breakpint-xxx-large >= 1920 px
    - **To Do:**
        - Layout for all screens in this project
        - Layout for <Window /> component - window instance indicating an active app
        - Layout of <StartMenu />
        -
    - **Note**
        - Using the breakpoints defined in my previous projects as these are tied to actual device sizes and testing tools. **This is better than abstract**

        - **numbers given above** - pragmatic choice that serves this project well ().
        -
    -
- Material
    - Solid - Opaque material that uses color and varying elevation. Use this as backup if mica doesn't work. It was in windows 10 and lower.
    - Mica - for open windows in Windows 11. Try it for this project.
    - Acrylic - A semi-transparent material that replicates the effect of frosted glass. Use it for transient, light-dismiss surfaces like popover and menus. Use it for right-click popups.
    - Smoke - Smoke emphasizes an important UI surface by dimming the surfaces beneath so that they recede into the background. Smoke is used to signal blocking interaction below a modal UI such as a dialog. Modal's outer layer + inner dialog window conveying the information.
    - **To Do** - Absolutely no idea. You only have to help me!
    - Solved by fluent package https://www.npmjs.com/package/@fluentui/tokens/v/0.0.0-nightly-20250103-0406.1
    - https://learn.microsoft.com/en-us/windows/apps/design/style/acrylic

- Motion
    - Container Transform - for resize and drag
    - **To Do** - Not sure what type of animation to use for other stuff. I don't even know what elements of my project require animation. Help me with this
- Shapes - defining properties are form, border radius and stroke
    - Forms - Use fill or borders to distinguish form elements from their surroundings
        - Rectangle - basic shape for most common components and containers, like buttons, textareas, menus, cards, and images
        - Circle - used for avatars and other components displaying or representing people
        - Pill - represent paths like a slider's track or a toggle's channel. Pills can also show tags, keywords, or a selection in a list.
        - Break - Tooltips
    - Border Radius
        - 4 px by default on rectangle
        - For shapes smaller than 32 pixels, the corner angle is reduced to 2 pixels. For large and extra-large components, 8 pixel and 12 pixel angles are used.
        - Fluent uses rounded corner styles in three areas of UI components: rectangular elements, flyout elements, and pill-shaped or round elements
    - Stroke - Apply stroke properties to create lines, arrows, borders, and vector networks.
        - Stroke thickness
        - Stroke dashed
        - Stroke caps
    - **To Do**
        - In my project, which components use Rectangle, Pill and Circle? List them

- Shall I stick to 4px border radius in web and mobile?
- What is color of all types of strokes?
  -

Corner radius tokens

Use these radius tokens to change the corner radius on elements.

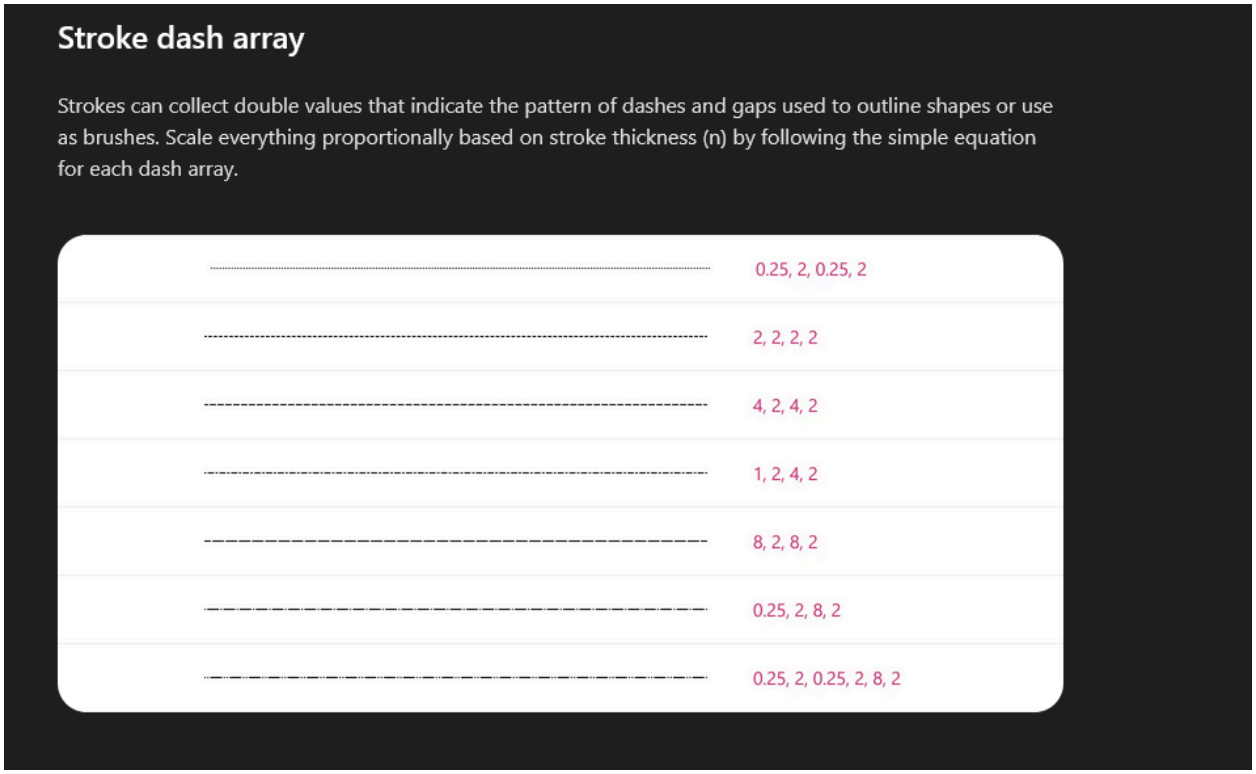
Token	Usage	Value
None	Navigation bars, tab bars	0 pixels
Small	Small badges	2 pixels
Medium	Buttons, dropdown	4 pixels
Large	Large buttons	8 pixels
X-Large	Button sheets, popovers	12 pixels
Circle	Personas	50%

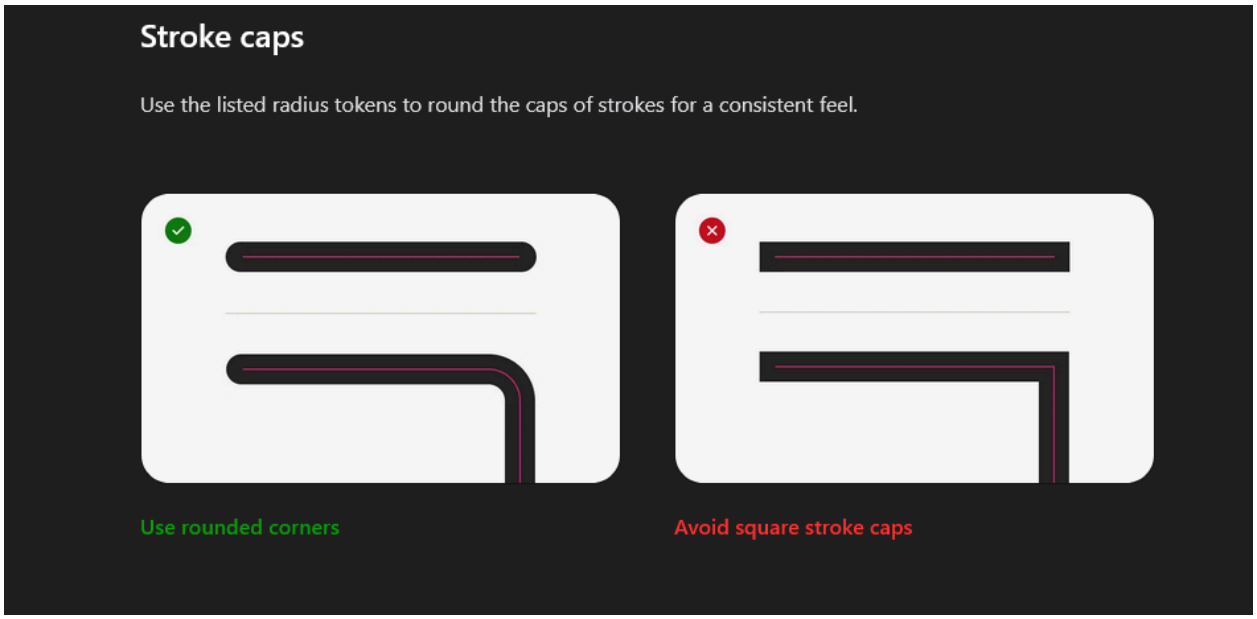
Stroke Thickness

Change line thickness with stroke tokens.

Token	Web	Mobile
Thin	1 pixel	1 pixel
Thick	2 pixels	2 pixels
Thicker	3 pixels	4 pixels
Thickest	4 pixels	6 pixels

To achieve correct visual weight, consider the size of the element when determining the stroke thickness. For example, on an avatar activity ring, strokes scale from a 2 pixel ring for smaller avatars to a 4 pixel ring for larger avatars. The visual weight feels consistent as the element scales up or down.





- Typography
    - Segoe UI Variable in windows/web
    - San Francisco Pro Display macOs / iOS - Not sure if NextJs provides a way to check the device to implement this in macOs and iOS. If not, keep it simple and default to Segoe
    - Roboto - Android. Again, if difficult to switch fonts depending on device then use Segoe
    - Use *Sentence case* when writing paras.

- Fluent uses base alignment to distribute vertical space.
- For left-to-right (LTR) languages like English, use left-alignment
- For right-to-left (RTL) languages like Arabic or Hebrew, use right-alignment
- Applying color can emphasize or de-emphasize text. Using a primary color on text will give it more visual prominence than the standard text color. In contrast, using a lighter neutral than standard text will subdue the text and place it lower in a visual hierarchy.

Packages - @microsoft/fluentui-tokens and `@fluentui/react-icons` (tree-shaken)

Type ramp

Web Windows macOS iOS Android

The Fluent type ramp for web builds from a long history of design at Microsoft. The type options pr
clear style direction and semantic roles for creating scannable hierarchies and a sense of balance. Th
uses Segoe UI as the default typeface.

Name	Weight	Size / Line-height
Caption 2	Regular	10px / 14px
Caption 2 Strong	Semibold	10px / 14px
Caption 1	Regular	12px / 16px
Caption 1 Stronger	Bold	12px / 16px
Body 1	Regular	14px / 20px
Body 1 Strong	Semibold	14px / 20px
Body 1 Stronger	Bold	14px / 20px
Subtitle 2	Semibold	16px / 22px
Subtitle 2 Stronger	Bold	16px / 22px
Subtitle 1	Semibold	20px / 26px
Title 3	Semibold	24px / 32px
Title 2	Semibold	28px / 36px
Title 1	Semibold	32px / 40px
Large Title	Semibold	40px / 52px
Display	Semibold	68px / 92px

Breakpoints and Responsive Layout

Shutdown and Booting Screens

List of operations being done when powering ON or powering OFF
DISPLAY_SHUTDOWN_SCREEN and DISPLAY_BOOT_SCREEN should display progress bar while each operating is being displayed.

Powering ON operations:

Pre-Login Boot (DISPLAY_BOOT_SCREEN)

Step	Operation Text (Displayed on Loader)	Technical Action (Zustand)
1.	Initializing Virtual Hardware...	Start CSS animations/loading GIF.
2.	Loading System UI Kernel...	Initialize the useSystemStore (Taskbar alignment, default theme settings).
3.	Applying Default Configuration...	Apply the universal default light/dark theme and wallpaper.
4.	Preparing Application Registry...	Map and load the static APP_REGISTRY constant into memory.
5.	**System Ready.** Launching Login Screen...	Set bootStatus to 'DISPLAY_LOGIN_SCREEN'.

Transition to login screen

Post-login Initialization - after successful login

Step	Operation Text (Displayed on Loader)	Technical Action (Zustand)
1.	Authenticating User: **[Username]**...	Set username and isAdmin state based on the input.
2.	Loading User Profile and Permissions...	Apply the **read-only vs. full-access** logic based on the isAdmin state.
3.	Applying Default User Theme...	Dispatch the action to apply the **default** custom colors, wallpaper, and the isTranslucent setting (not saved settings).
4.	**Welcome, [Username].** Launching Desktop Environment...	Set bootStatus to 'ON' (Desktop is visible).

Shutdown Operations:

Step	Operation Text (Displayed on Loader)	Technical Action (Zustand)
1.	Shutting down system processes...	Disable all user input listeners.
2.	Closing **[X]** active windows...	Iterate through the windows array and close them all.

3.	Disconnecting User Profile...	Clear the username and set isLoggedIn: false in the store.
4.	**Shutdown Complete.**	Set bootStatus to 'DISPLAY_POWER_SCREEN'.

Components

<AppIcon />

- Variant = taskbar, desktop, mobile, start-menu
- Shape = circular, square, water-droplet
- appId = id of the application from AppMetaData
- Taskbar right click
    - New window with app's icon
    - Unpin from taskbar
    - Close window if it's a single active instance
    - Close all windows if there are multiple active instances
- Desktop right click
    - Double click opens new window
    - Pin to taskbar (if not pinned)
    - Unpin from taskbar (if pinned)
    - Properties (displays app info modal)
- Start Menu right click
    - Pin/unpin to/from taskbar
    - Open a new window
-

<Taskbar />

- Windows image
- Search input + search magnifying glass
- Pinned apps
- Static icons - date & time, wifi, battery, language

<WindowContainer />

Handling window display types:
1. Minimize - Hide with CSS (display: none or visibility: hidden)
2. Maximize - Full desktop size (width: 100%, height: 100%)

3. Normal - Fixed dimensions (width: 45vw, height: 35vh or whatever defaults) For the restore functionality, the key insight is:

- Clicking maximize when **normal** → sets to **maximized**
- Clicking maximize when **maximized** → restores to **normal**
- Clicking **minimize** → sets to **minimized**
- Clicking taskbar icon when **minimized** → restores to previous state (either **normal** or **maximized**)

<Desktop />