Information Management System

Appendices

Table of contents:

Appendix I: Interviews with the Client

Interview 1: Criterion A - Initial assessment and Defining the Problem

Aim: Understand existing systems and defining its problems
Date: Oct 1, 2024

Name: Isht
Occupation: Student

Describing the existing system:

Please describe your current system of work.

I use many different applications and systems, both physical and digital, to meet the varied needs of my lifestyle.

I currently use pen and paper methods or just rely on my working memory to keep track of my tasks, plans.

As a student I need to be able to keep track of the statuses and tasks on many different things for all the areas of work that I have. If I were to make a general list of those things then it would consist of things like;

- What do I need to do now and later?
- What's the upcoming deadline for xyz subject and xyz component?
- Where did I store an xyz file for the xyz component/subject?
- What did I come up with on the researching spree for xyz topic and where did I store it?
- The What Where Why When Who How of almost every little detail that I will need later on for an xyz purpose.

Aside from keeping track of the many details I need to schedule practice and review sessions for each of the subjects that I have signed up for, with each having its own set of time and effort requirements for effective learning. I manage this with a frankenstein of multiple methods, ranging from calendars and time boxes to impulsive decisions on what to be doing at a particular point of time. But usually I rely upon sheets of paper And carry them around with me, most of the time, and I usually misplace or forget what was written where. And that really bites into my time as I frequently have to reorganise and often rewrite the sheets. Which is also really frustrating in itself as I'd rather spend more time on my work than this pseudo-work.

What do you think is the core problem at hand?

I believe it is the sheer inefficiency of my current system that really acts as a broken pot where I'm losing time, no matter how hard I work I always have to manually adhere to this system to get my tasks done.

Are there any existing solutions that you have given a try?

Yes, Certainly, I have tried to use applications such as Obsidian, Notion, and a few others. But the problem with them is that they do not have what I need, and both require extensive initial setup and time investment, something that I am simply not willing to do.

I see. Do you have a solution in mind already or would you like me to find it out?

I know that a digital system is required. I think a Website would help, because I need to be able to access my sheets and other data from anywhere.

Alright, that's totally ok. Finally, what are your expectations for the outcome of the application? I mean what exactly would you like to see?

I would like to have a website that Isn't much of a hassle to work with, preferably matching my working style and not requiring excessive overhead setup. Most of my work involves note-making, a bunch of todo lists, and also noting down a bunch of ideas that I can't immediately give my attention to. And I also keep a physical note of a bunch of things that I look at when I feel down, it really helps me stay motivated to keep working.

Ok Isht, that's a good starting point for this application, I would like to shadow you during the times you use your pen and paper system, so that I can fully understand it.

Yeah, sure.

Interview 2: Criterion A - Finalising success criteria

Aim: Finalise Success Criteria.
Date: Oct 2, 2024

Good morning. I believe I have collected enough data to understand your system fully. And I have Identified a few key areas that are central requirements: A todoList specific to your style of having multiple lists working in tandem with each other, a place where you can save any important information regarding specific topics that you find interesting but cannot immediately attend to and/or doubts regarding your subjects of study, which, looking at your sheets I can tell that they also contain additional information that needs to be saved with the doubts. I've also noticed that you seem to prefer having many of your sheets in front of you when working, ordering them as per the momentary requirement. I believe if we can digitise these aspects of your workflow then a large portion of the time lost in managing sheets can surely be cut down.

Whoa! That is very insightful. Yeah sure, please commence with development, I'm excited to see the resultant application. Additionally, I would like to be able to use multiple individual accounts for various tasks.

No problem, I will set up an authentication system that can meet your needs.

Oh and I would also like to be able to have all the previously mentioned features in one place… A mission control of sorts!

That's a great addition! It will fit well with your need to have everything laid out in front. I will add them to the plans for the application. Thank you.

Interview 3: Criterion B - Visual and Functional design overviews

Aim: Validating the UI/UX with the client.
Date: Nov 4, 2024

Good Morning! I've made some preliminary designs to get a clear picture about the kind of UI/UX that you prefer. Here, please have a look at these wireframe designs and functionality designs.

I see that the UI is fine, but just to be clear I would like to have something that is elegant to look at, as I'll probably be using it quite often.

No need to worry, I'll make it as per your preferences. I will use a minimalistic UI design that will be easy to look at for long periods of time. Could you please review the representations of the functionality.

Sure. I think this is fine, I could understand most of it and it seems good to go.

That's great to hear! I will start development immediately. I will also be frequently validating the features as I build them so that everything goes smoothly. Thank you.

Interview 4: Criterion E - Evaluation

Aim: Evaluation after final product testing
Date: Feb 16, 2025

Good Morning! I hope the application has stood to your needs. I would love to hear your experience!

I am thrilled by how much this simplifies my work and I especially love that I can switch between multiple accounts and have data to be isolated within them and also the Side-by-Side view, I found it really convenient. It has met my expectations and surpassed them.

I'm glad that it has molded well with your workflow. Any thoughts on the navigation between the pages?

Oh yes I do like it that the interface is super simple and intuitive, it's almost like I've already been using this application for a long time now.

Lovely! Now for the cruxes of the application… How are the features integrating with your workstyle??

The features are working just as we planned, and they also have a very self-explanatory UI.

That's great! Now is there anything else that you might want to expand in this application?

I'm having a great time with the application, and honestly this is really satisfactory, but the application could be made available to a wider audience and maybe add a feature that can track the number of study hours and maybe create analytics for that data, that would really help in accountability and efficient planning. Also, I would like to see more flexibility in terms of Customising the UI. The current one is fantastic, but it would really benefit me to have more freedom over the look of the UI.

Those are fantastic additions to the application! I will continue working on this project whenever I can.

I am very happy with how everything for the application turned out. Thank you for all the effort you put into this project.

I am really glad you liked it and I'm happy to hear that its functionality is easy to navigate and also right for your particular situation. Thank you for the opportunity to create this application for you!

Appendix II: Work Shadowing of Client Workflow

1. Introduction

Purpose: As discussed in Interview 1 (Appendix I), Isht currently employs a paper-based system for managing his academic information and tasks. To gain a deeper, practical understanding of this system and identify specific pain points, a work shadowing session was conducted.

Date and Duration: This work shadowing took place on Oct 2, 2024, and lasted for approximately 0.5 hours.

Methodology: The work shadowing involved direct observation of Isht as he engaged in his daily academic tasks, with concurrent note-taking and opportunities for clarifying questions to understand the rationale behind his actions and system.

2. Methodology in Detail

- **Direct Observation:** Isht was observed directly in his usual study environment (his home study desk) as he worked on typical academic tasks.
- **Non-participatory Observation:** The shadowing was primarily non-participatory. Interaction was limited to brief, clarifying questions to avoid disrupting Isht's natural workflow.
- **Detailed Note-Taking:** Comprehensive notes were taken throughout the session, documenting the mediums used, the processes followed for each task, and any observed organizational strategies or challenges. These notes served as the primary data source for this appendix.
- **Clarifying Questions:** Questions were asked during natural pauses in Isht's workflow to gain insight into his system's logic, the purpose of specific actions, and his perceived difficulties.

3. Observations - Key Areas of Workflow

- **a) Note-Taking/Idea Recording:**
    - **Mediums Used:** Isht was observed using a combination of A4 spiral-bound notebooks for subject-specific notes (e.g., 'Maths Notebook', 'Physics Notes'), loose A4 lined sheets for quick notes and drafts, and small sticky notes for fleeting ideas and reminders.

- **Process:** Note-taking within notebooks was primarily linear and chronological, with notes added sequentially to the next available page. Idea recording was more spontaneous, with ideas jotted down on whichever medium was immediately accessible – often loose sheets or sticky notes, irrespective of the current task.
- **Organisation:** While notebooks provided a degree of subject-based organization, idea capture was less structured. Loose sheets and sticky notes, though convenient for immediate capture, lacked a systematic integration into the main notebooks and were observed to be scattered across his desk.
- **Example:** During the shadowing session, while working on a Maths problem set, Isht had a sudden idea related to his Computer Science IA. He quickly wrote it on a yellow sticky note and adhered it to the edge of his laptop screen, separate from his Maths notebook and task lists.

- **b) Planning and Task Management:**

  - **Tools:** Isht relied on physical to-do lists created on A4 lined paper. Multiple lists were in use concurrently, including subject-specific lists (e.g., 'To-Do: Chemistry IA') and broader lists for general academic tasks ('Weekly Action Items').
  - **Task Breakdown:** Tasks were generally written as brief action phrases (e.g., 'Revise Chapter 3', 'Email advisor about draft'). Subject-specific lists provided some categorization, but there was no central, consolidated overview of all pending tasks across all subjects.
  - **Prioritization/Deadlines:** Task prioritization appeared to be implicitly managed through the order of tasks on each list, and potentially based on immediate deadlines. Deadlines were occasionally noted beside tasks but not consistently across all lists.
  - **Example:** Isht was observed creating a fresh to-do list titled 'Physics Revision - Week of Oct 14th' on a new sheet of paper. This list was separate from another active to-do list labelled 'General To-Do - October', which contained tasks for different subjects and administrative actions.

- **c) Studying and Revision:**

  - **Materials:** Study and revision primarily utilized subject-specific notebooks and textbooks. Revision notes were sometimes created within the same subject notebooks, or occasionally on separate loose sheets for summary purposes.

- **Integration of Notes:** During study and revision, Isht frequently flipped between different notebooks and consulted loose sheets to gather all relevant information on a topic. This demonstrated the dispersed nature of related information across multiple physical locations.
- **Revision Strategies:** Revision methods included re-reading notes and textbook sections, highlighting key passages, and occasionally re-writing core concepts onto new sheets for better retention.
- **Example:** When revising for an upcoming Biology test on 'Cellular Respiration', Isht needed to consult his main Biology notebook, a separate smaller notebook containing experiment notes, and a few loose sheets with diagrams he had drawn – all spread across his desk.

- **d) Self-Motivation/Mental Organisation ("Cookie Jar" Concept):**

  - **Physical "Cookie Jar":** Isht maintains a physical 'Cookie Jar' in the form of a small tin box containing handwritten index cards. Each card contains a personal achievement, positive affirmation, or motivational quote.
  - **Usage:** Isht explained that he accesses his 'Cookie Jar' when feeling unmotivated or stressed. He reads through the cards to remind himself of his successes and maintain a positive mindset. This is a purely motivational tool, entirely separate from his academic workflow.
  - **Integration with Workflow?:** The 'Cookie Jar' is a standalone motivational aid and not integrated into his academic task management, note-taking, or study processes.

- **e) Workspace Organisation - Multiple Sheet Layout:**

  - **Observed Behaviour:** Isht was observed to dynamically arrange several physical sheets of paper on his desk while working. These sheets were often positioned side-by-side or slightly overlapping. The specific configuration changed depending on the nature of the task he was currently focused on.
  - **Purpose:** Isht clarified that this layout was intentional, allowing him to have simultaneous visual access to multiple pieces of related information. He emphasized that having these sheets 'laid out in front of him' aided concentration and facilitated quicker cross-referencing between notes, task lists, and relevant ideas.
  - **Analogy to Digital Workflow:** This physical sheet layout mirrored Isht's digital multitasking habits on his laptop. He mentioned frequently using split-screen views and multiple browser windows to manage different

applications and information streams concurrently, indicating a consistent preference for a visually accessible, multi-window workspace across both physical and digital domains.
    - **Example:** During a period dedicated to ‘IA Project Planning,’ Isht arranged three sheets: one containing brainstorming notes, another with a preliminary project outline, and a third with a subject-specific research article. He actively shifted his focus between these sheets, demonstrating their interconnected relevance to the planning task.
- **f) Digital Platform Preference - Reddit Usage:**
    - **Observed Behaviour:** During a break in the work shadowing session, Isht mentioned that he frequently uses Reddit in his free time for information gathering and general browsing. He elaborated that he finds the platform engaging and spends a considerable amount of his leisure time navigating its content.
    - **Implication for UI Design:** This casual observation provides valuable insight into Isht's digital platform preferences. Reddit's user interface is characterized by a clean, card-based layout, upvoting/downvoting mechanisms for content prioritization, and threaded comment sections for discussion and clarification. Isht's stated familiarity and engagement with Reddit suggests that a user interface incorporating similar design elements might be intuitively comfortable and easily navigable for him. This observation provides a justification for considering a Reddit-like UI aesthetic for modules within the Information Management System, particularly for modules involving content feeds, idea organization, and discussion features.
    - **Integration with Design Rationale:** This observation directly informs the design rationale for the Doubt Tracker and Curiosity Space modules. The decision to adopt a card-based layout, incorporate upvoting/downvoting, and implement comment threads in these modules is strategically aligned with Isht's demonstrated preference for the Reddit platform. By mirroring familiar UI patterns, the application aims to minimize the learning curve and enhance user engagement for these specific modules, contributing to the overall "plug and play” requirement from the client..one

4. Observed Inefficiencies and Problems

The work shadowing session provided direct evidence for the inefficiencies and problems inherent in Isht’s current paper-based system, as initially identified in Criterion A and Interview 1:

- **Data Misplacement:** Work shadowing validated the high risk of data misplacement. Loose sheets and sticky notes, utilized for capturing ideas and quick notes (Observation 3a), were observed to be easily moved and were not systematically stored, increasing their susceptibility to being misplaced or lost – a concern Isht explicitly raised in Interview 1.
- **Redundant Data Entry:** The use of multiple, separate to-do lists (Observation 3b) necessitates redundant data entry. Tasks pertaining to multiple subjects or overarching goals might be rewritten across different lists, increasing workload and the probability of inconsistencies and omissions.
- **Inconsistent Records:** The lack of a centralized system contributes to inconsistent records. Information related to a single subject or project is fragmented across various notebooks and loose sheets (Observation 3c), making it challenging to obtain a comprehensive overview and ensure all information is consistently updated and readily accessible.
- **Time-Consuming "Paper Purges":** The observed scattering of information across various mediums directly corroborates Isht's description of time-consuming “paper purges” (Scenario, Criterion A). The manual process of sorting, consolidating, and reorganizing accumulated papers to regain a semblance of order is clearly an inefficient and recurring time sink, directly impacting his productivity.
- **Cognitive Overload:** The mental effort required to manage and navigate multiple paper-based systems was palpably evident during the shadowing. Isht constantly needed to recall the location of specific notes, mentally track multiple task lists, and remember the purpose of particular sheet arrangements, significantly contributing to cognitive overload as described in Interview 1 and Criterion A.
- **Inefficient Management of Multiple Physical Sheets:** While Isht's strategy of laying out multiple sheets aims to enhance visual accessibility and task focus (Observation 3e), it introduces inefficiencies in a paper-based system. The physical arrangement, while momentarily helpful, is transient and not persistently saved. Reconstructing a specific sheet layout for a recurring task involves manual re-organization each time, adding to 'pseudo-work' and wasted time. Furthermore, updating and maintaining consistency across multiple sheets in such a layout becomes increasingly complex and error-prone, exacerbating data redundancy and inconsistency issues.

The work shadowing session provided invaluable first-hand insights into the practical realities of Isht's paper-based information management system. It definitively confirmed the fragmented nature of his workflow, the inefficiencies arising from disparate mediums, the tangible risk of data loss, and the significant cognitive overload inherent in the system. A particularly insightful observation was Isht's deliberate and dynamic

arrangement of multiple physical sheets to create a task-specific workspace (Observation 3e). This behaviour, mirroring his digital multitasking habits, directly inspired the development of the ‘Stage Manager’ module within the proposed Information Management System. The Stage Manager is designed to digitally replicate this ‘mission control’ workspace concept, allowing Isht to create and save custom layouts of different application modules (represented as 'Spaces' and 'Windows') tailored to specific tasks or workflows. This feature directly addresses the inefficiency of manually recreating paper sheet layouts by providing a persistent, digital solution for organizing and accessing multiple information streams concurrently, mirroring and enhancing Isht's preferred working style and directly mitigating the observed problems identified throughout the work shadowing process and in Criterion A.

Appendix III: Complete Codebase

User Authentication

```
==========
Directory: src
==========

File: src\middleware.ts
----------

import { clerkMiddleware, createRouteMatcher } from "@clerk/nextjs/server";

/**
 * Define public routes that don't require authentication.
 * `createRouteMatcher` is used to create a function that checks if a given path matches the provided patterns.
 * In this case, it checks if the path starts with '/sign-in'.
 */
const isPublicRoute = createRouteMatcher(["/sign-in(.*)"]);

/**
 * `clerkMiddleware` is the main middleware that handles authentication for the application.
 * It receives an asynchronous function that takes `auth` and `request` objects.
 *
 * The middleware checks if the current route is public using `isPublicRoute`.
 * If the route is not public, it ensures that the user is authenticated by calling `auth.protect()`.
 * `auth.protect()` will throw an error if the user is not signed in, effectively blocking access to the route.
 */
export default clerkMiddleware(async (auth, request) => {
  if (!isPublicRoute(request)) {
    await auth.protect();
  }
});

/**
```

```
 * Configuration for the middleware to define which paths should be processed by the
middleware.
 * `matcher` array specifies the paths that `clerkMiddleware` should apply to.
 *
 * It uses a negative lookahead regular expression to exclude paths that start with
'_next' (Next.js internals)
 * or contain a file extension (static files), unless they are found in search parameters.
 * It also explicitly includes API routes and trpc routes to ensure they are always
protected.
 */
export const config = {
  matcher: [
    // Skip Next.js internals and all static files, unless found in search params

"/((?!_next|[^?]*\\.(?:html?|css|js(?!on)|jpe?g|webp|png|gif|svg|ttf|woff2?|ico|csv|docx?|xl
sx?|zip|webmanifest)).*)",
    // Always run for API routes
    "/(api|trpc)(.*)",
  ],
};

===========
Directory: src\app
===========

File: src\app\layout.tsx
--------------------

import { ClerkProvider } from "@clerk/nextjs";
import type { Metadata } from "next";
import "./globals.css";
import { Toaster } from "@/components/ui/toaster";

export const metadata: Metadata = {
  title: "IBDP-SIMS",
  description: "Personal Performance Optimization Platform",
};

export default function RootLayout({
  children,
```

```
}: Readonly<{
  children: React.ReactNode;
}>) {
  return (
    <ClerkProvider>
      <html lang="en">
        <body className={`antialiased`}>
            {children}
          <Toaster />
        </body>
      </html>
    </ClerkProvider>
  );
}


===========
Directory: src\app\sign-in\[[...sign-in]]
===========

File: src\app\sign-in\[[...sign-in]]\page.tsx
----------

import { SignIn } from "@clerk/nextjs";

export default function Page() {
  return (
    <div className="min-h-screen flex justify-center items-center">
      <SignIn />
    </div>
  );
}
```

Dashboard and Workstage Layout

```
==========
Directory: src\app\components
==========

File: src\app\components\Dashboard.tsx
--------------------

import {
  ResizableHandle,
  ResizablePanel,
  ResizablePanelGroup,
} from "@/components/ui/resizable";
import { ScrollArea } from "@/components/ui/scroll-area";
import Cookies from "@/components/CookieJar";
import DocumentJar from
"@/components/ContinuousInfoSpace/ContinuousInfoSpaceDocMan";
import ToDoList from "@/components/ToDoList";

export default function Dashboard() {
  return (
    <ResizablePanelGroup
      direction="vertical"
      className="rounded-lg border"
      style={{ width: "1600px" }}
    >
      <ResizablePanel minSize={30}>
        <ResizablePanelGroup
          direction="horizontal"
          className="rounded-lg border"
        >
          <ResizablePanel defaultSize={40} minSize={30}>
            <ScrollArea className="h-full w-full rounded-xl bg-muted/50">
              <Cookies disableEdit disableAdd disableDelete gridCols={3} />
            </ScrollArea>
          </ResizablePanel>
          <ResizableHandle />
          <ResizablePanel>
            <ScrollArea className="h-full w-full rounded-xl bg-muted/50">
```

```
            <ToDoList hideBackButton={true} isDashboard={true} />
          </ScrollArea>
        </ResizablePanel>
      </ResizablePanelGroup>
    </ResizablePanel>
    <ResizableHandle />
    <ResizablePanel>
      <ResizablePanelGroup
        direction="horizontal"
        className="rounded-lg border"
      >
        <ResizablePanel>
          <DocumentJar />
        </ResizablePanel>
      </ResizablePanelGroup>
    </ResizablePanel>
  </ResizablePanelGroup>
 );
}
```

File: src\app\components\SpaceDeleteAlertToast.tsx

```
import { Button } from "@/components/ui/button";
import { useToast } from "@/hooks/use-toast";

interface SpaceDeleteButtonProps {
 spaceId: number;
 onSpaceDelete: (spaceId: number) => void;
}

interface WindowDeleteButtonProps {
 spaceId: number;
 windowId: string;
 windowTitle: string;
 onWindowDelete: (spaceId: number, windowId: string) => void;
}

export function SpaceDeleteToast({
```

```
  spaceId,
  onSpaceDelete,
}: SpaceDeleteButtonProps) {
  const { toast } = useToast();

  const handleDeleteClick = () => {
    toast({
      title: "Delete Space",
      description: "Are you sure you want to delete this space?",
      action: (
        <Button
          variant="outline"
          size="sm"
          onClick={() => {
            onSpaceDelete(spaceId);
            toast({
              title: "Space deleted",
              description: `Space ${
                spaceId + 1
              } has been deleted successfully.`,
            });
          }}
        >
          Delete
        </Button>
      ),
    });
  };

  return (
    <Button variant="destructive" onClick={handleDeleteClick}>
      Delete Space {spaceId + 1}
    </Button>
  );
}

export function WindowDeleteToast({
  spaceId,
  windowId,
  windowTitle,
```

```
  onWindowDelete,
}: WindowDeleteButtonProps) {
  const { toast } = useToast();

  const handleDeleteClick = () => {
    toast({
      title: "Delete Window",
      description: `Are you sure you want to delete "${windowTitle}"?`,
      action: (
        <Button
          variant="outline"
          size="sm"
          onClick={() => {
            onWindowDelete(spaceId, windowId);
            toast({
              title: "Window deleted",
              description: `"${windowTitle}" has been deleted successfully.`,
            });
          }}
        >
          Delete
        </Button>
      ),
    });
  };

  return (
    <div onClick={handleDeleteClick} className="cursor-pointer">
      ✖
    </div>
  );
}
```

File: src\app\components\WorkStage.tsx

```
// app/components/WorkStage.tsx
"use client";
```

```
import { useState, useEffect } from "react";
import { Manager, Space, Spaces, BasicWindow } from
"@/components/stage-manager";
import { Button } from "@/components/ui/button";
import { SpaceDeleteToast } from "./SpaceDeleteAlertToast";
import { db } from "@/lib/firebase";
import { doc, getDoc, setDoc } from "firebase/firestore";
import { useUser } from "@clerk/nextjs";
import { useToast } from "@/hooks/use-toast";

// Interface defining the structure of a draggable/resizable window
interface WindowData {
  id: string;
  title: string;
  content: string;
  position: [number, number];
  size: [number, number];
}

// Interface defining a workspace containing multiple windows
interface SpaceData {
  id: number;
  windows: WindowData[];
}

export default function WorkStage() {
  // User authentication state
  const { user } = useUser();
  // State management for workspaces and current active space
  const [spaces, setSpaces] = useState<SpaceData[]>([]);
  const [currentSpace, setCurrentSpace] = useState(0);
  // Toast notification system
  const { toast } = useToast();

  /**
   * Persists current workspace state to Firestore
   * @param spacesToSave - Array of SpaceData to save
   */
  const saveData = async (spacesToSave: SpaceData[]) => {
    if (!user) return;
```

```
  const docRef = doc(db, "users", user.id, "stageManager", "data");
  await setDoc(docRef, { spaces: spacesToSave });
};

// Load user's workspace data on component mount or user change
useEffect(() => {
  const loadData = async () => {
    if (!user) return;

    const docRef = doc(db, "users", user.id, "stageManager", "data");
    const docSnap = await getDoc(docRef);

    if (docSnap.exists()) {
      const data = docSnap.data();
      // Convert Firestore data to properly typed format
      const formattedSpaces = data.spaces.map((space: any) => ({
        ...space,
        windows: space.windows.map((window: any) => ({
          ...window,
          position: window.position as [number, number],
          size: window.size as [number, number],
          content: window.content,
        })),
      }));
      setSpaces(formattedSpaces);
    }
  };

  loadData();
}, [user]);

/**
 * Creates a new workspace with a default window
 */
const addSpace = () => {
  const newSpace: SpaceData = {
    id: spaces.length,
    windows: [
      {
        id: Date.now().toString(),
```

```
        title: "New Document",
        content: "",
        position: [100, 100],
        size: [400, 300],
      },
    ],
  };

  setSpaces((prev) => {
    const newSpaces = [...prev, newSpace];
    saveData(newSpaces);
    return newSpaces;
  });
  // Switch to the new space after creation
  setCurrentSpace(newSpace.id);
};

/**
 * Removes a workspace and adjusts current space selection
 * @param spaceId - ID of the space to remove
 */
const deleteSpace = (spaceId: number) => {
  setSpaces((prev) => {
    const newSpaces = prev.filter((space) => space.id !== spaceId);
    saveData(newSpaces);
    return newSpaces;
  });
  // Ensure current space stays valid after deletion
  setCurrentSpace((prev) => Math.max(0, prev === spaceId ? prev - 1 : prev));
};

/**
 * Updates properties of a specific window
 * @param spaceId - Parent workspace ID
 * @param windowId - Target window ID
 * @param newData - Partial window data to update
 */
const updateWindow = (
  spaceId: number,
  windowId: string,
```

```
  newData: Partial<WindowData>
) => {
  setSpaces((prev) => {
    const newSpaces = prev.map((space) => {
      if (space.id === spaceId) {
        return {
          ...space,
          windows: space.windows.map((window) =>
            window.id === windowId
              ? {
                  ...window,
                  ...newData,
                  // Ensure content remains if not provided in update
                  content: newData.content || window.content,
                }
              : window
          ),
        };
      }
      return space;
    });
    saveData(newSpaces);
    return newSpaces;
  });
};

/**
 * Adds a new window to the specified workspace
 * @param spaceId - Target workspace ID
 */
const addWindow = (spaceId: number) => {
  setSpaces((prev) => {
    const newSpaces = prev.map((space) => {
      if (space.id === spaceId) {
        const newWindow = {
          id: Date.now().toString(),
          title: "New Document",
          content: "",
          // Stagger window positions for visibility
          position: [
```

```
            100 + space.windows.length * 20,
            100 + space.windows.length * 20,
          ] as [number, number],
          size: [400, 300] as [number, number],
        };
        return {
          ...space,
          windows: [...space.windows, newWindow],
        };
      }
      return space;
    });
    saveData(newSpaces);
    return newSpaces;
  });
};

/**
 * Removes a window from its parent workspace
 * @param spaceId - Parent workspace ID
 * @param windowId - Target window ID to remove
 */
const deleteWindow = (spaceId: number, windowId: string) => {
  setSpaces((prev) => {
    const newSpaces = prev.map((space) => {
      if (space.id === spaceId) {
        return {
          ...space,
          windows: space.windows.filter((window) => window.id !== windowId),
        };
      }
      return space;
    });
    saveData(newSpaces);
    return newSpaces;
  });
};

return (
  <div className="p-4 bg-gray-100">
```

```
{/* Workspace control buttons */}
<div className="mb-4 flex gap-2">
  <Button onClick={addSpace}>Create New Space</Button>
  <Button onClick={() => addWindow(currentSpace)}>Add Window</Button>

  {/* Conditional delete space button with confirmation toast */}
  {spaces.length > 1 && (
    <SpaceDeleteToast
      spaceId={currentSpace}
      onSpaceDelete={deleteSpace}
    />
  )}
</div>

{/* Main workspace visualization component */}
<Manager
  size={[1600, 760]}
  style={{
    margin: "0 auto",
    border: "2px solid #333",
    borderRadius: "8px",
    overflow: "hidden",
  }}
>
  <Spaces
    space={currentSpace}
    onSpaceChange={setCurrentSpace}
    className="bg-white"
  >
    {spaces.map((space) => (
      <Space key={space.id}>
        {space.windows.map((window) => (
          <BasicWindow
            key={window.id}
            title={window.title}
            content={window.content}
            initialPosition={window.position}
            initialSize={window.size}
            style={{ background: "#fff" }}
            onTitleChange={(newTitle) =>
```

```
          updateWindow(space.id, window.id, { title: newTitle })
        }
        onContentChange={(newContent) =>
          updateWindow(space.id, window.id, { content: newContent })
        }
        onPositionChange={(newPosition) =>
          updateWindow(space.id, window.id, { position: newPosition })
        }
        onSizeChange={(newSize) =>
          updateWindow(space.id, window.id, { size: newSize })
        }
        onClose={() => {
          // Toast confirmation flow for window deletion
          const triggerToast = () => {
            toast({
              title: "Delete Window",
              description: `Are you sure you want to delete "${window.title}"?`,
              action: (
                <Button
                  variant="outline"
                  size="sm"
                  onClick={() => {
                    deleteWindow(space.id, window.id);
                    toast({
                      title: "Window deleted",
                      description: `"${window.title}" has been deleted successfully.`,
                    });
                  }}
                >
                  Delete
                </Button>
              ),
            });
          };
          triggerToast();
        }}
      />
    ))}
  </Space>
))}
```

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

Cookie Jar Module

```
==========
Directory: src\components
==========

File: src\components\CookieJar.tsx
----------
```

```
"use client";

import React, { useState, useEffect } from "react";
import { Plus, Trash2, Edit } from "lucide-react";
import {
  DndContext,
  closestCenter,
  KeyboardSensor,
  PointerSensor,
  useSensor,
  useSensors,
  DragEndEvent,
  DragStartEvent,
  useDraggable,
  useDroppable,
} from "@dnd-kit/core";
import {
  arrayMove,
  sortableKeyboardCoordinates,
  rectSortingStrategy,
  SortableContext,
  useSortable,
} from "@dnd-kit/sortable";
import { CSS } from "@dnd-kit/utilities";
import { Button } from "@/components/ui/button";
import {
  readCookies,
  updateCookiePositions,
  deleteCookie,
  updateCookie,
  Cookie,
```

```
  addCookie,
} from "@/lib/cookies-actions";

interface CookiesProps {
  disableEdit?: boolean;
  disableAdd?: boolean;
  disableDelete?: boolean;
  gridCols?: number;
}

/**
 * TrashZone Component: Represents the droppable area for deleting cookies.
 * It becomes visible at the bottom of the screen during drag operations when delete is
enabled.
 */
function TrashZone({ disabled }: { disabled: boolean }) {
  const { isOver, setNodeRef } = useDroppable({
    id: "trash",
    disabled,
  });

  // If deletion is disabled, don't render the trash zone
  if (disabled) return null;

  return (
    <div
      ref={setNodeRef}
      className={`fixed bottom-4 left-1/2 -translate-x-1/2 bg-red-500 text-white p-4
rounded-full
        transition-all duration-300 ${
          isOver ? "scale-110 bg-red-600" : "scale-100"
        }`}
    >
      <Trash2 className="h-6 w-6" />
    </div>
  );
}

/**
```

```
 * SortableCookieCard Component: Represents a single cookie card that can be
dragged and sorted.
 * It includes functionalities for editing and displaying cookie information.
 */
function SortableCookieCard({
  cookie,
  onEdit,
  isDragging,
  disableEdit,
}: {
  cookie: Cookie;
  onEdit: (cookie: Cookie) => void;
  isDragging: boolean;
  disableEdit: boolean;
}) {
  // Hooks for sortable functionality from dnd-kit
  const {
    attributes,
    listeners,
    setNodeRef: setSortableNodeRef,
    transform,
    transition,
  } = useSortable({ id: cookie.id });

  // Hooks for draggable functionality from dnd-kit - nested draggable within sortable
  const {
    attributes: dragAttributes,
    listeners: dragListeners,
    setNodeRef: setDraggableNodeRef,
    transform: dragTransform,
  } = useDraggable({
    id: cookie.id,
    data: { type: "cookie" }, // Data to identify the type of draggable item
  });

  // Style to apply during drag operation for visual feedback
  const style = {
    transform: CSS.Transform.toString(transform || dragTransform),
    transition,
    opacity: isDragging ? 0.5 : 1, // Make card slightly transparent during drag
```

```
  };

  return (
    <div
      ref={(node) => {
        setSortableNodeRef(node);
        setDraggableNodeRef(node);
      }}
      style={style}
      {...attributes}
      {...listeners}
      {...dragAttributes}
      {...dragListeners}
      className="bg-white p-4 rounded-lg shadow-md cursor-move relative group
min-h-[120px] flex flex-col border border-gray-200" // Changed background and added
border
    >
      <h3 className="text-gray-800 font-bold text-lg mb-2 break-words">
        {cookie.name}
      </h3>{" "}
      {/* Changed text color */}
      <p className="text-gray-700 text-sm break-words overflow-y-auto max-h-24">
        {cookie.description}
      </p>{" "}
      {/* Changed text color */}
      {!disableEdit && (
        <div className="absolute top-2 right-2 flex gap-2 opacity-0
group-hover:opacity-100 transition-opacity">
          <button
            onClick={(e) => {
              e.stopPropagation(); // Prevent card click from triggering parent actions
              onEdit(cookie); // Callback to handle cookie editing
            }}
            className="text-gray-600 hover:text-gray-800" // Changed text color
          >
            <Edit className="h-4 w-4" />
          </button>
        </div>
      )}
    </div>
```

```
  );
}

/**
 * CookieJar Component: Main component for displaying and managing cookies in a draggable grid.
 * It handles cookie data fetching, adding, editing, deleting, and reordering.
 */
export default function CookieJar({
  disableEdit = false,
  disableAdd = false,
  disableDelete = false,
  gridCols = 4,
}: CookiesProps) {
  const [isModalOpen, setIsModalOpen] = useState(false); // State for controlling the cookie modal
  const [cookies, setCookies] = useState<Cookie[]>([]); // State to store the list of cookies
  const [newCookie, setNewCookie] = useState<Partial<Cookie>>({
    name: "",
    description: "",
  }); // State to manage new cookie input values in the modal
  const [isEditing, setIsEditing] = useState(false); // State to track if the modal is in edit mode
  const [editingCookieId, setEditingCookieId] = useState<string | null>(null); // State to hold the ID of the cookie being edited
  const [activeId, setActiveId] = useState<string | null>(null); // State to track the currently dragged cookie ID

  // Configure sensors for drag and drop interactions (Pointer and Keyboard)
  const sensors = useSensors(
    useSensor(PointerSensor, {
      activationConstraint: {
        distance: 30, // Drag activation distance threshold
      },
    }),
    useSensor(KeyboardSensor, {
      coordinateGetter: sortableKeyboardCoordinates, // Use keyboard coordinates for accessibility
    })
```

```
  );

  // useEffect hook to fetch cookies on component mount
  useEffect(() => {
    async function fetchCookies() {
      const fetchedCookies = await readCookies(); // Call to fetch cookies from database
      setCookies(fetchedCookies); // Update cookies state with fetched data
    }
    fetchCookies();
  }, []);

  const toggleModal = () => setIsModalOpen(!isModalOpen); // Function to toggle modal
visibility

  // Handler for input changes in the modal form
  const handleInputChange = (
    e: React.ChangeEvent<HTMLInputElement | HTMLTextAreaElement>
  ) => {
    setNewCookie((prev) => ({
      ...prev,
      [e.target.id]: e.target.value, // Dynamically update the corresponding field in
newCookie state
    }));
  };

  // Handler to prepare modal for editing an existing cookie
  const handleEdit = (cookie: Cookie) => {
    if (disableEdit) return; // Prevent edit if editing is disabled
    setNewCookie(cookie); // Populate modal form with cookie data
    setIsEditing(true); // Set modal to edit mode
    setEditingCookieId(cookie.id); // Store the ID of the cookie being edited
    setIsModalOpen(true); // Open the modal
  };

  // Handler for form submission (Add or Edit Cookie)
  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();

    if (isEditing && editingCookieId) {
      // Handle cookie update
```

```
    if (disableEdit) return; // Prevent update if editing is disabled

    const updatedCookie: Cookie = {
      ...newCookie,
      id: editingCookieId,
    } as Cookie;

    try {
      await updateCookie(updatedCookie); // Call to update cookie in database
      const updatedCookies = await readCookies(); // Re-fetch cookies to reflect
changes
      setCookies(updatedCookies); // Update local cookie state
    } catch (error) {
      console.error("Failed to update cookie:", error);
    } finally {
      setIsEditing(false); // Reset edit mode state
      setEditingCookieId(null); // Clear editing cookie ID
      setNewCookie({ name: "", description: "" }); // Reset new cookie input state
      setIsModalOpen(false); // Close the modal
    }
  } else {
    // Handle new cookie addition
    if (disableAdd) return; // Prevent add if adding is disabled

    try {
      if (!newCookie.name || !newCookie.description) {
        console.error("Name and description are required");
        return;
      }

      const cookieToAdd = {
        name: newCookie.name,
        description: newCookie.description,
      };

      await addCookie(cookieToAdd, gridCols); // Call to add new cookie to database
      const updatedCookies = await readCookies(); // Re-fetch cookies to reflect addition
      setCookies(updatedCookies); // Update local cookie state
      setNewCookie({ name: "", description: "" }); // Reset new cookie input state
      setIsModalOpen(false); // Close the modal
```

```
    } catch (error) {
      console.error("Failed to add cookie:", error);
    }
  }
};

// Handler for drag start event
const handleDragStart = (event: DragStartEvent) => {
  setActiveId(event.active.id as string); // Set active ID to the ID of the dragged cookie
};

// Handler for drag end event
const handleDragEnd = async (event: DragEndEvent) => {
  const { over, active } = event;
  setActiveId(null); // Reset active ID after drag end

  if (!disableDelete && over?.id === "trash" && active.id) {
    // Handle cookie deletion when dropped on trash zone
    await deleteCookie(String(active.id)); // Call to delete cookie from database
    const updatedCookies = await readCookies(); // Re-fetch cookies to reflect deletion
    setCookies(updatedCookies); // Update local cookie state
    return;
  }

  setCookies((items) => {
    const oldIndex = items.findIndex((item) => item.id === active.id); // Find the original
index of the dragged cookie
    const newIndex = items.findIndex((item) => item.id === over?.id); // Find the new
index based on drop target

    if (oldIndex === -1 || newIndex === -1) {
      return items; // Return original items if indices are invalid
    }

    const reorderedItems = arrayMove(items, oldIndex, newIndex); // Reorder items
array

    // Update positions based on new order in the grid
    const updatedItemsWithPositions = reorderedItems.map((item, index) => ({
      ...item,
```

```
        position: {
          x: index % gridCols, // Calculate x position based on column
          y: Math.floor(index / gridCols), // Calculate y position based on row
        },
      }));

      updateCookiePositions(updatedItemsWithPositions); // Call to update cookie positions in database
      return updatedItemsWithPositions; // Return updated items array
    });
  };

  // Dynamic grid classes based on props
  const gridClasses = `grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 xl:grid-cols-${gridCols} gap-4`;

  return (
    <div className="p-6">
      {!disableAdd && (
        <Button
          onClick={() => {
            setNewCookie({ name: "", description: "" }); // Reset input fields
            setIsEditing(false); // Ensure modal is in add mode
            toggleModal(); // Open the modal
          }}
          variant="outline"
          size="default"
          className="mb-4"
        >
          <Plus className="h-4 w-4" />
          Add Cookie
        </Button>
      )}

      {/* DndContext Provider for drag and drop functionality */}
      <DndContext
        sensors={sensors}
        collisionDetection={closestCenter} // Strategy for detecting collisions during drag
        onDragStart={handleDragStart} // Handler for drag start event
        onDragEnd={handleDragEnd} // Handler for drag end event
```

```
>
  {/* SortableContext to manage sortable items */}
  <SortableContext
    items={cookies.map((a) => a.id)} // Array of cookie IDs for sortable context
    strategy={rectSortingStrategy} // Strategy for sorting (rectangle sorting)
  >
    <div className={gridClasses}>
      {cookies.map((cookie) => (
        <SortableCookieCard
          key={cookie.id}
          cookie={cookie}
          onEdit={handleEdit}
          isDragging={activeId === cookie.id} // Pass dragging state to card for styling
          disableEdit={disableEdit}
        />
      ))}
    </div>
  </SortableContext>

  {/* TrashZone component for deleting cookies */}
  <TrashZone disabled={disableDelete} />
</DndContext>

{/* Modal for adding/editing cookies */}
{isModalOpen && (
  <div className="fixed inset-0 bg-black/50 flex items-center justify-center">
    <div className="bg-background p-6 rounded-lg w-full max-w-md">
      <form onSubmit={handleSubmit}>
        <div className="mb-4">
          <label
            htmlFor="name"
            className="block text-sm font-medium mb-2 text-gray-700"
          >
            {" "}
            {/* Changed text color */}
            Name
          </label>
          <input
            type="text"
            id="name"
```

```
              value={newCookie.name}
              onChange={handleInputChange}
              className="w-full px-4 py-2 rounded-md bg-gray-100 text-gray-800
focus:outline-none focus:ring-2 focus:ring-green-400 border border-gray-200" //
Changed background and text color, added border
              placeholder="Enter cookie name"
              required
            />
          </div>

          <div className="mb-4">
            <label
              htmlFor="description"
              className="block text-sm font-medium mb-2 text-gray-700"
            >
              {" "}
              {/* Changed text color */}
              Description
            </label>
            <textarea
              id="description"
              value={newCookie.description}
              onChange={handleInputChange}
              className="w-full px-4 py-2 rounded-md bg-gray-100 text-gray-800
focus:outline-none focus:ring-2 focus:ring-green-400 border border-gray-200" //
Changed background and text color, added border
              rows={3}
              placeholder="Enter cookie description"
              required
            />
          </div>

          <div className="flex justify-end gap-4">
            <Button
              type="button"
              onClick={() => {
                toggleModal(); // Close the modal
                setIsEditing(false); // Reset edit mode
              }}
              variant="secondary"
```

```
                size="default"
              >
                Cancel
              </Button>

              <Button type="submit" variant="default" size="default">
                {isEditing ? "Update" : "Submit"}
              </Button>
            </div>
          </form>
        </div>
      </div>
    )}
  </div>
 );
}


===========
Directory: src\lib
===========

File: src\lib\cookies-actions.ts
--------------------

"use server";

import { z } from "zod";
import { db } from "@/lib/firebase";
import {
 doc,
 getDoc,
 setDoc,
} from "firebase/firestore";
import { auth } from "@clerk/nextjs/server";

/**
 * Zod schema to validate the structure of a Cookie object.
 * Ensures that cookie data conforms to the expected format before being processed or
stored.
```

```
 * Defines the shape of a cookie with properties like id, name, description, and an
optional position.
 */
const CookieSchema = z.object({
  id: z.string(),
  name: z.string().min(1, "Name is required"),
  description: z.string().min(1, "Description is required"),
  position: z
    .object({
      x: z.number(),
      y: z.number(),
    })
    .optional(),
});
export type Cookie = z.infer<typeof CookieSchema>;

/**
 * Interface defining the structure of the cookieJar document in Firestore.
 * It specifies that the document should contain a 'cookies' field, which is a record
(object)
 * where keys are cookie IDs (strings) and values are Cookie objects (without the 'id'
field, as the ID is the key).
 */
interface CookieJarDocument {
  cookies: Record<string, Omit<Cookie, 'id'>>;
}

// --- Firestore Interaction Functions ---

/**
 * Asynchronously retrieves a DocumentReference to the user's cookieJar document in
Firestore.
 * This function handles authentication to ensure only logged-in users can access their
cookie jar.
 * It constructs the Firestore path to the cookieJar document, which is nested under the
user's document.
 * @returns {Promise<DocumentReference>} - A promise that resolves to a
DocumentReference for the cookieJar.
 * @throws {Error} - If the user is not authenticated, an error is thrown.
 */
```

```
async function getCookieJarDocRef() {
  const { userId } = await auth();
  if (!userId) {
    throw new Error("User not authenticated");
  }
  return doc(db, "users", userId, "cookieJar", "cookies");
}

/**
 * Reads and retrieves all cookies from the user's cookieJar in Firestore.
 * Fetches the cookieJar document and extracts the cookie data, then transforms it into an array of Cookie objects.
 * It also sorts the cookies based on their 'position' property to maintain visual order, defaulting to position (0,0) if not set.
 * @returns {Promise<Cookie[]>} - A promise that resolves to an array of Cookie objects. Returns an empty array if no cookies are found or in case of errors.
 */
export async function readCookies(): Promise<Cookie[]> {
  try {
    const cookieJarDocRef = await getCookieJarDocRef();
    const docSnap = await getDoc(cookieJarDocRef);

    // If the cookieJar document does not exist, return an empty array.
    if (!docSnap.exists()) {
      return [];
    }

    const data = docSnap.data() as CookieJarDocument;
    // Convert the record of cookies into an array of Cookie objects, including the ID from the document key.
    return Object.entries(data.cookies).map(([id, cookie]) => ({
      id,
      ...cookie
    })).sort((a, b) => {
      // Sort cookies primarily by y-position (row) and then by x-position (column) to maintain grid order.
      if (a.position?.y === b.position?.y) {
        return (a.position?.x || 0) - (b.position?.x || 0); // Default to 0 if position is not defined.
      }
```

```
      return (a.position?.y || 0) - (b.position?.y || 0); // Default to 0 if position is not defined.
    });
  } catch (error) {
    console.error("Error reading cookies from Firestore:", error);
    return []; // Return an empty array in case of an error to prevent app crashes.
  }
}

/**
 * Adds a new cookie to the user's cookieJar in Firestore.
 * Generates a unique ID for the new cookie and sets its initial position to the top-left of the grid (0,0).
 * It merges the new cookie into the existing cookieJar document, preserving existing cookies.
 * @param {Omit<Cookie, "id">} newCookie - An object containing the new cookie's data (name and description), excluding the ID.
 * @param {number} [gridCols=4] - Optional parameter for the number of columns in the cookie grid (currently not directly used in addCookie but might be relevant for position calculation in future).
 * @returns {Promise<void>} - A promise that resolves when the cookie is successfully added to Firestore.
 */
export async function addCookie(
  newCookie: Omit<Cookie, "id">,
  gridCols: number = 4, // gridCols parameter is defined but not currently used in positioning logic
): Promise<void> {
  try {
    const cookieJarDocRef = await getCookieJarDocRef();
    const docSnap = await getDoc(cookieJarDocRef);
    const id = crypto.randomUUID(); // Generate a unique ID for the new cookie using crypto.randomUUID().

    // Determine the existing cookie data. If the document doesn't exist yet, start with an empty cookies object.
    const existingData = docSnap.exists()
      ? (docSnap.data() as CookieJarDocument)
      : { cookies: {} };

    await setDoc(cookieJarDocRef, {
```

```
      cookies: {
        ...existingData.cookies, // Merge with existing cookies
        [id]: {
          name: newCookie.name,
          description: newCookie.description,
          position: { x: 0, y: 0 } // Set initial position for new cookies to (0,0)
        }
      }
    });
  } catch (error) {
    console.error("Error adding cookie to Firestore:", error);
  }
}

/**
 * Deletes a cookie from the user's cookieJar in Firestore.
 * Removes the cookie with the given ID from the 'cookies' record in the cookieJar
document.
 * @param {string} id - The ID of the cookie to delete.
 * @returns {Promise<void>} - A promise that resolves when the cookie is successfully
deleted from Firestore.
 */
export async function deleteCookie(id: string): Promise<void> {
  try {
    const cookieJarDocRef = await getCookieJarDocRef();
    const docSnap = await getDoc(cookieJarDocRef);

    // If the cookieJar document does not exist, there's nothing to delete, so just return.
    if (!docSnap.exists()) return;

    const data = docSnap.data() as CookieJarDocument;
    // Use destructuring to remove the cookie with the specified ID from the cookies
record.
    const { [id]: deletedCookie, ...remainingCookies } = data.cookies;

    await setDoc(cookieJarDocRef, {
      cookies: remainingCookies // Update Firestore with the remaining cookies.
    });
  } catch (error) {
    console.error("Error deleting cookie from Firestore:", error);
```

```
  }
}

/**
 * Updates an existing cookie's data in Firestore.
 * Overwrites the data for the cookie with the matching ID in the cookieJar document.
 * @param {Cookie} updatedCookie - The complete Cookie object with updated properties (name, description, position).
 * @returns {Promise<void>} - A promise that resolves when the cookie is successfully updated in Firestore.
 */
export async function updateCookie(updatedCookie: Cookie): Promise<void> {
  try {
    const cookieJarDocRef = await getCookieJarDocRef();
    const docSnap = await getDoc(cookieJarDocRef);

    // If the cookieJar document does not exist, return as there's nothing to update.
    if (!docSnap.exists()) return;

    const data = docSnap.data() as CookieJarDocument;

    await setDoc(cookieJarDocRef, {
      cookies: {
        ...data.cookies, // Keep existing cookies
        [updatedCookie.id]: { // Overwrite the cookie with the given ID
          name: updatedCookie.name,
          description: updatedCookie.description,
          position: updatedCookie.position // Update position as well
        }
      }
    });
  } catch (error) {
    console.error("Error updating cookie in Firestore:", error);
  }
}

/**
 * Updates the positions of multiple cookies in Firestore.
 * Efficiently updates the 'position' field for each cookie in the provided array within the cookieJar document.
```

```
 * This is typically used after drag-and-drop operations to persist the new cookie layout.
 * @param {Cookie[]} updatedCookies - An array of Cookie objects, each containing an updated 'position' property.
 * @returns {Promise<void>} - A promise that resolves when all cookie positions are successfully updated in Firestore.
 */
export async function updateCookiePositions(
  updatedCookies: Cookie[]
): Promise<void> {
  try {
    const cookieJarDocRef = await getCookieJarDocRef();
    const docSnap = await getDoc(cookieJarDocRef);

    // If the cookieJar document does not exist, return as there's nothing to update.
    if (!docSnap.exists()) return;

    const data = docSnap.data() as CookieJarDocument;
    const updatedData = { ...data.cookies }; // Create a shallow copy of the existing cookies data.

    updatedCookies.forEach(cookie => {
      // Only update if the cookie exists in the current data.
      if (updatedData[cookie.id]) {
        updatedData[cookie.id] = {
          ...updatedData[cookie.id],
          position: cookie.position // Update only the position, preserving other cookie properties.
        };
      }
    });

    await setDoc(cookieJarDocRef, {
      cookies: updatedData // Update Firestore with the modified cookies data containing new positions.
    });
  } catch (error) {
    console.error("Error updating cookie positions in Firestore:", error);
  }
}
```

```
===========
Directory: src\app\CookieJar
===========

File: src\app\CookieJar\page.tsx
-------------------

"use client";

import React from "react";
import CookieJar from "@/components/CookieJar";
import { BackButton } from "@/components/ui/custom-ui/back-button";

const CookieJarPage: React.FC = () => {
  return (
    <div className="flex flex-col h-screen">
      <header className="flex items-center justify-between p-4 border-b">
        <BackButton />
      </header>
      <main className="flex-1 overflow-auto">
        <CookieJar />
      </main>
    </div>
  );
};

export default CookieJarPage;
```

Doubt Tracker Module

```
==========
Directory: src\components
==========

File: src\components\DoubtTracker.tsx
----------
```

```
"use client";
import React, { useState, useEffect } from "react";
import DoubtList from "@/components/doubts-tracker/DoubtList";
import NewDoubtForm from "@/components/doubts-tracker/NewDoubtForm";
import { Button } from "@/components/ui/button";
import { Tabs, TabsContent, TabsList, TabsTrigger } from "@/components/ui/tabs";
import {
  Dialog,
  DialogContent,
  DialogDescription,
  DialogHeader,
  DialogTitle,
  DialogTrigger,
} from "@/components/ui/dialog";
import { Plus } from "lucide-react";
import { useUser } from "@clerk/nextjs";
import { db } from "@/lib/firebase";
import {
  collection,
  addDoc,
  updateDoc,
  deleteDoc,
  query,
  orderBy,
  onSnapshot,
  increment,
  serverTimestamp,
  doc,
} from "firebase/firestore";

export interface Comment {
```

```
  id: string;
  text: string;
  parentId?: string | null;
  replies: Comment[];
}

export interface Doubt {
  id: string;
  title: string;
  description: string;
  upvotes: number;
  downvotes: number;
  resolved: boolean;
  solution?: string;
  comments: Comment[];
}

export default function DoubtTracker() {
  // User authentication state
  const { user } = useUser();
  // State to hold the list of doubts.
  const [doubts, setDoubts] = useState<Doubt[]>([]);

  // useEffect hook to fetch doubts from Firestore when the component mounts or when
the user changes.
  useEffect(() => {
    if (!user) return;
    // Reference to the 'nugget' collection within the user's document in Firestore.
    const nuggetRef = collection(db, "users", user.id, "nugget");
    // Create a query to fetch doubts, ordered by creation time in descending order.
    const q = query(nuggetRef, orderBy("createdAt", "desc"));
    // Subscribe to real-time updates from Firestore using onSnapshot.
    const unsubscribe = onSnapshot(q, (snapshot) => {
      const loadedDoubts: Doubt[] = [];
      // Iterate over each document in the snapshot.
      snapshot.forEach((docSnap) => {
        // Extract data from each document and shape it into an Doubt object.
        loadedDoubts.push({
          id: docSnap.id,
          title: docSnap.data().title,
```

```
          description: docSnap.data().description,
          upvotes: docSnap.data().upvotes,
          downvotes: docSnap.data().downvotes,
          resolved: docSnap.data().resolved,
          solution: docSnap.data().solution,
          comments: [], // Initialize comments as an empty array, they are fetched
separately if needed.
        });
      });
      // Update the doubts state with the fetched doubts.
      setDoubts(loadedDoubts);
    });
    // Return the unsubscribe function to detach the listener when the component
unmounts.
    return () => unsubscribe();
  }, [user]); // Dependency array ensures this effect runs when 'user' object changes.

  // Function to add a new doubt to Firestore.
  const addDoubt = async (title: string, description: string) => {
    if (!user) return;
    try {
      // Get reference to the 'nugget' collection for the current user.
      const nuggetRef = collection(db, "users", user.id, "nugget");
      // Add a new document to the 'nugget' collection with the provided title and
description.
      await addDoc(nuggetRef, {
        title,
        description,
        upvotes: 0,
        downvotes: 0,
        resolved: false,
        solution: "",
        createdAt: serverTimestamp(), // Use Firestore server timestamp for creation time.
      });
    } catch (error) {
      console.error("Error creating doubt:", error);
    }
  };

  // Function to resolve an doubt in Firestore.
```

```
  const resolveDoubt = async (id: string, solution: string) => {
    if (!user) return;
    try {
      // Get reference to the specific doubt document within the user's 'nugget' collection.
      const doubtRef = doc(db, "users", user.id, "nugget", id);
      // Update the doubt document to mark it as resolved and add the provided solution.
      await updateDoc(doubtRef, { resolved: true, solution });
    } catch (error) {
      console.error("Error resolving doubt:", error);
    }
  };

  // Function to reopen a resolved doubt in Firestore.
  const reopenDoubt = async (id: string) => {
    if (!user) return;
    try {
      // Get reference to the specific doubt document.
      const doubtRef = doc(db, "users", user.id, "nugget", id);
      // Update the doubt document to mark it as not resolved and clear the solution.
      await updateDoc(doubtRef, { resolved: false, solution: "" });
    } catch (error) {
      console.error("Error reopening doubt:", error);
    }
  };

  // Function to upvote an doubt in Firestore.
  const upvoteDoubt = async (id: string) => {
    if (!user) return;
    try {
      // Get reference to the specific doubt document.
      const doubtRef = doc(db, "users", user.id, "nugget", id);
      // Increment the upvotes count for the doubt document using Firestore's increment
operator.
      await updateDoc(doubtRef, { upvotes: increment(1) });
    } catch (error) {
      console.error("Error upvoting doubt:", error);
    }
  };

  // Function to downvote an doubt in Firestore.
```

```
  const downvoteDoubt = async (id: string) => {
    if (!user) return;
    try {
      // Get reference to the specific doubt document.
      const doubtRef = doc(db, "users", user.id, "nugget", id);
      // Increment the downvotes count for the doubt document using Firestore's
increment operator.
      await updateDoc(doubtRef, { downvotes: increment(1) });
    } catch (error) {
      console.error("Error downvoting doubt:", error);
    }
  };

  // Function to add a comment to an doubt in Firestore.
  const addComment = async (
    doubtId: string,
    commentText: string,
    parentId?: string
  ) => {
    if (!user) return;
    try {
      // Get reference to the 'comments' subcollection within the specific doubt document.
      const commentsRef = collection(
        db,
        "users",
        user.id,
        "nugget",
        doubtId,
        "comments"
      );
      // Add a new comment document to the 'comments' subcollection.
      await addDoc(commentsRef, {
        text: commentText,
        parentId: parentId || null, // Store parentId if it's a reply, otherwise null for top-level
comments.
        createdAt: serverTimestamp(), // Use Firestore server timestamp for comment
creation time.
      });
    } catch (error) {
      console.error("Error adding comment:", error);
```

```
    }
  };

  // Function to edit an existing comment in Firestore.
  const editComment = async (
    doubtId: string,
    commentId: string,
    newText: string
  ) => {
    if (!user) return;
    try {
      // Get reference to the specific comment document within the 'comments'
subcollection.
      const commentRef = doc(
        db,
        "users",
        user.id,
        "nugget",
        doubtId,
        "comments",
        commentId
      );
      // Update the text field of the comment document with the new text.
      await updateDoc(commentRef, { text: newText });
    } catch (error) {
      console.error("Error editing comment:", error);
    }
  };

  // Function to delete a comment from Firestore.
  const deleteComment = async (doubtId: string, commentId: string) => {
    if (!user) return;
    try {
      // Get reference to the specific comment document to be deleted.
      const commentRef = doc(
        db,
        "users",
        user.id,
        "nugget",
        doubtId,
```

```
      "comments",
      commentId
    );
    // Delete the comment document from Firestore.
    await deleteDoc(commentRef);
  } catch (error) {
    console.error("Error deleting comment:", error);
  }
};

// Function to edit an doubt's title and description in Firestore.
const editDoubt = async (id: string, title: string, description: string) => {
  if (!user) return;
  try {
    // Get reference to the specific doubt document.
    const doubtRef = doc(db, "users", user.id, "nugget", id);
    // Update the title and description fields of the doubt document.
    await updateDoc(doubtRef, { title, description });
  } catch (error) {
    console.error("Error editing doubt:", error);
  }
};

// Function to edit the solution of a resolved doubt in Firestore.
const editSolution = async (id: string, newSolution: string) => {
  if (!user) return;
  try {
    // Get reference to the specific doubt document.
    const doubtRef = doc(db, "users", user.id, "nugget", id);
    // Update the solution field of the doubt document with the new solution.
    await updateDoc(doubtRef, { solution: newSolution });
  } catch (error) {
    console.error("Error editing solution:", error);
  }
};

// Function to delete an doubt from Firestore.
const deleteDoubt = async (id: string) => {
  if (!user) return;
  try {
```

```
      // Get reference to the specific doubt document to be deleted.
      const doubtRef = doc(db, "users", user.id, "nugget", id);
      // Delete the doubt document from Firestore.
      await deleteDoc(doubtRef);
    } catch (error) {
      console.error("Error deleting doubt:", error);
    }
  };

  return (
    <div className="container mx-auto p-4 max-w-4xl">
      {/* Header section of the Doubt Tracker, containing the title and 'New Doubt' button.
*/}
      <div className="flex justify-between items-center mb-4">
        <h1 className="text-3xl font-bold text-orange-600">Doubts Tracker</h1>
        {/* Dialog component for creating a new doubt, triggered by the 'New Doubt'
button. */}
        <Dialog>
          <DialogTrigger asChild>
            <Button>
              <Plus className="mr-2 h-4 w-4" /> New Doubt
            </Button>
          </DialogTrigger>
          <DialogContent>
            <DialogHeader>
              <DialogTitle>Create a New Doubt</DialogTitle>
              <DialogDescription>
                Add a new doubt to track and resolve.
              </DialogDescription>
            </DialogHeader>
            {/* NewDoubtForm component for handling the input and submission of new
doubts. */}
            <NewDoubtForm onSubmit={addDoubt} />
          </DialogContent>
        </Dialog>
      </div>
      {/* Tabs component to separate 'Open Doubts' and 'Resolved Doubts'. */}
      <Tabs defaultValue="open" className="mt-4">
        <TabsList className="grid w-full grid-cols-2">
          <TabsTrigger value="open">Open Doubts</TabsTrigger>
```

```
          <TabsTrigger value="resolved">Resolved Doubts</TabsTrigger>
        </TabsList>
        {/* Content for the 'Open Doubts' tab. */}
        <TabsContent value="open">
          {/* DoubtList component to display the list of open doubts. */}
          <DoubtList
            doubts={doubts.filter((d) => !d.resolved)} // Filter doubts to show only open ones.
            onResolve={resolveDoubt}
            onReopen={reopenDoubt}
            onUpvote={upvoteDoubt}
            onDownvote={downvoteDoubt}
            onAddComment={addComment}
            onEditDoubt={editDoubt}
            onEditComment={editComment}
            onDeleteComment={deleteComment}
            onEditSolution={editSolution}
            onDeleteDoubt={deleteDoubt}
          />
        </TabsContent>
        {/* Content for the 'Resolved Doubts' tab. */}
        <TabsContent value="resolved">
          {/* DoubtList component to display the list of resolved doubts. */}
          <DoubtList
            doubts={doubts.filter((d) => d.resolved)} // Filter doubts to show only resolved
ones.
            onResolve={resolveDoubt}
            onReopen={reopenDoubt}
            onUpvote={upvoteDoubt}
            onDownvote={downvoteDoubt}
            onAddComment={addComment}
            onEditDoubt={editDoubt}
            onEditComment={editComment}
            onDeleteComment={deleteComment}
            onEditSolution={editSolution}
            onDeleteDoubt={deleteDoubt}
          />
        </TabsContent>
      </Tabs>
    </div>
  );
```

```
}


==========
Directory: src\app\DoubtTracker
==========

File: src\app\DoubtTracker\page.tsx
----------

"use client";

import React from "react";
import DoubtTracker from "@/components/DoubtTracker";
import { BackButton } from "@/components/ui/custom-ui/back-button";

const DoubtTrackerPage: React.FC = () => {
  return (
    <div className="flex flex-col h-screen">
      <header className="flex items-center justify-between p-4 border-b">
        <BackButton />
      </header>
      <main className="flex-1 overflow-auto">
        <DoubtTracker />
      </main>
    </div>
  );
};

export default DoubtTrackerPage;


==========
Directory: src\components\doubts-tracker
==========

File: src\components\doubts-tracker\CommentSection.tsx
----------

import React, { useState, useRef, useEffect } from "react";
```

```
import { Button } from "@/components/ui/button";
import { Input } from "@/components/ui/input";
import {
  MessageSquare,
  Edit,
  Save,
  Reply,
  Trash2,
  MoreVertical,
  X,
} from "lucide-react";
import {
  DropdownMenu,
  DropdownMenuContent,
  DropdownMenuItem,
  DropdownMenuTrigger,
} from "@/components/ui/dropdown-menu";
import { useUser } from "@clerk/nextjs";

// Define the Comment interface to structure comment data
export interface Comment {
  id: string;
  text: string;
  parentId?: string | null;
  replies: Comment[];
}

// Define the props for the CommentSection component
interface CommentSectionProps {
  doubtId: string; // ID of the doubt to which the comments belong
  comments: Comment[]; // Array of comments to display
  onAddComment: (text: string, parentId?: string) => void; // Function to handle adding a
new comment
  onEditComment: (commentId: string, newText: string) => void; // Function to handle
editing an existing comment
  onDeleteComment: (commentId: string) => void; // Function to handle deleting a
comment
}
```

```
// Define the props for the CommentItem component, which represents a single comment
interface CommentItemProps {
  doubtId: string; // ID of the doubt, passed down for comment actions
  comment: Comment; // The comment object to render
  onAddComment: (text: string, parentId: string) => void; // Function to add a reply to a comment
  onEditComment: (commentId: string, newText: string) => void; // Function to edit a comment (passed down)
  onDeleteComment: (commentId: string) => void; // Function to delete a comment (passed down)
}

/**
 * CommentItem Component: Represents a single comment and its replies.
 * Handles rendering comment text, edit/delete options, and reply functionality.
 */
const CommentItem: React.FC<CommentItemProps> = ({
  doubtId,
  comment,
  onAddComment,
  onEditComment,
  onDeleteComment,
}) => {
  // User authentication state from Clerk
  const { user } = useUser();
  // State to manage whether the comment is in edit mode
  const [isEditing, setIsEditing] = useState(false);
  // State to hold the text being edited in the comment
  const [editedText, setEditedText] = useState(comment.text);
  // State to manage whether the reply input is shown for this comment
  const [isReplying, setIsReplying] = useState(false);
  // State to hold the text for a new reply
  const [replyText, setReplyText] = useState("");
  // useRef to manage focus outside of the reply input to close it
  const replyRef = useRef<HTMLDivElement>(null);
  // useRef to manage focus outside of the edit input to exit edit mode
  const editRef = useRef<HTMLDivElement>(null);
  // State to track if there are unsaved changes in the edit input
  const [hasChanges, setHasChanges] = useState(false);
```

```
  /**
   * Handler for saving the edited comment text.
   * Calls the onEditComment prop function to update the comment and exits edit mode.
   */
  const handleSaveEdit = () => {
    onEditComment(comment.id, editedText);
    setIsEditing(false);
  };

  /**
   * Handler for cancelling comment editing.
   * Resets the edited text to the original comment text and exits edit mode.
   */
  const handleCancelEdit = () => {
    setEditedText(comment.text);
    setIsEditing(false);
  };

  /**
   * Handler for adding a reply to the comment.
   * Calls the onAddComment prop function with the reply text and the current
comment's ID as the parent.
   * Resets the reply input and closes the reply input area.
   */
  const handleAddReply = () => {
    if (replyText) {
      onAddComment(replyText, comment.id);
      setIsReplying(false);
      setReplyText("");
    }
  };

  /**
   * Handler for input changes in the edit comment input.
   * Updates the editedText state and checks if changes have been made compared to
the original text.
   * @param {React.ChangeEvent<HTMLInputElement>} e - The input change event.
   */
  const handleEditTextChange = (e: React.ChangeEvent<HTMLInputElement>) => {
```

```
    setEditedText(e.target.value);
    setHasChanges(e.target.value !== comment.text);
  };

  /**
   * useEffect hook to handle closing the reply input when clicking outside.
   * Adds a mousedown event listener to the document to detect clicks outside the reply input area.
   */
  useEffect(() => {
    const handleClickOutsideReply = (event: MouseEvent) => {
      if (
        replyRef.current &&
        !replyRef.current.contains(event.target as Node)
      ) {
        if (!replyText) setIsReplying(false);
      }
    };
    document.addEventListener("mousedown", handleClickOutsideReply);
    return () =>
      document.removeEventListener("mousedown", handleClickOutsideReply);
  }, [replyText]); // Dependency array includes replyText to re-run effect when reply text changes.

  /**
   * useEffect hook to handle exiting edit mode when clicking outside the edit input area, if no changes are made.
   * Adds a mousedown event listener to the document to detect clicks outside the edit input area.
   */
  useEffect(() => {
    const handleClickOutsideEdit = (event: MouseEvent) => {
      if (
        !hasChanges &&
        editRef.current &&
        !editRef.current.contains(event.target as Node)
      ) {
        setIsEditing(false);
      }
    };
```

```
    document.addEventListener("mousedown", handleClickOutsideEdit);
    return () =>
      document.removeEventListener("mousedown", handleClickOutsideEdit);
  }, [hasChanges]); // Dependency array includes hasChanges to re-run effect when
changes status is updated.

  return (
    <div className="mt-2 pl-4 border-l-2 border-gray-200 group">
      {/* Conditional rendering: Edit mode input or comment text display */}
      {isEditing ? (
        <div ref={editRef} className="flex items-center space-x-2">
          <Input
            value={editedText}
            onChange={handleEditTextChange}
            className="flex-grow"
          />
          <Button size="sm" onClick={handleSaveEdit} title="Save">
            <Save className="h-4 w-4" />
          </Button>
          <Button
            size="sm"
            variant="ghost"
            onClick={handleCancelEdit}
            title="Cancel"
          >
            <X className="h-4 w-4" />
          </Button>
        </div>
      ) : (
        <div className="flex items-center justify-between">
          <p className="text-sm whitespace-pre-wrap break-words max-w-[90%]">
            {comment.text}
          </p>
          {/* Conditional rendering: Action buttons (Reply, Edit, Delete) - Shown on group
hover */}
          <div className="flex space-x-2 opacity-0 group-hover:opacity-100
transition-opacity">
            {/* Reply button to toggle reply input */}
            <Button
              size="sm"
```

```
        variant="ghost"
        onClick={() => setIsReplying(true)}
        title="Reply"
      >
        <Reply className="h-4 w-4" />
      </Button>
      {/* Dropdown menu for Edit and Delete actions */}
      <DropdownMenu>
        <DropdownMenuTrigger asChild>
          <Button size="sm" variant="ghost" title="More options">
            <MoreVertical className="h-4 w-4" />
          </Button>
        </DropdownMenuTrigger>
        <DropdownMenuContent>
          <DropdownMenuItem onClick={() => setIsEditing(true)}>
            <Edit className="h-4 w-4 mr-2" />
            Edit
          </DropdownMenuItem>
          <DropdownMenuItem
            className="text-red-600"
            onClick={() => onDeleteComment(comment.id)}
          >
            <Trash2 className="h-4 w-4 mr-2" />
            Delete
          </DropdownMenuItem>
        </DropdownMenuContent>
      </DropdownMenu>
    </div>
  </div>
)}
{/* Conditional rendering: Reply input area */}
{isReplying && (
  <div ref={replyRef} className="mt-2 flex items-center space-x-2">
    <Input
      value={replyText}
      onChange={(e) => setReplyText(e.target.value)}
      placeholder="Write a reply..."
      className="flex-grow"
    />
    <Button size="sm" onClick={handleAddReply}>
```

```
            Reply
          </Button>
        </div>
      )}
      {/* Recursive rendering of replies */}
      {comment.replies &&
        comment.replies.map((reply) => (
          <CommentItem
            key={reply.id}
            doubtId={doubtId}
            comment={reply}
            onAddComment={onAddComment}
            onEditComment={onEditComment}
            onDeleteComment={onDeleteComment}
          />
        ))}
    </div>
  );
};

/**
 * CommentSection Component: Renders the comment section, including the list of comments and the new comment input form.
 * Manages the overall display of comments and handles adding new top-level comments.
 */
export default function CommentSection({
  doubtId,
  comments,
  onAddComment,
  onEditComment,
  onDeleteComment,
}: CommentSectionProps) {
  // State to hold the text for a new top-level comment
  const [newComment, setNewComment] = useState("");
  // State to manage the expanded/collapsed state of the comment section
  const [expanded, setExpanded] = useState(false);

  /**
   * Handler for submitting a new top-level comment.
```

```
   * Prevents default form submission and calls the onAddComment prop function to add the new comment.
   * Resets the new comment input area after submission.
   * @param {React.FormEvent} e - The form submit event.
   */
  const handleSubmit = (e: React.FormEvent) => {
    e.preventDefault();
    if (newComment) {
      onAddComment(newComment);
      setNewComment("");
    }
  };

  /**
   * Helper function to recursively count all replies for a given comment and its nested replies.
   * @param {Comment} comment - The comment object to start counting replies from.
   * @returns {number} - The total count of replies for the comment and all its nested replies.
   */
  const countReplies = (comment: Comment): number => {
    return comment.replies.reduce(
      (acc, reply) => acc + 1 + countReplies(reply),
      0
    );
  };

  // Calculate total comment count (including nested replies) for display purposes
  const totalCommentCount = comments.reduce(
    (acc, c) => acc + 1 + countReplies(c),
    0
  );

  return (
    <div className="w-full">
      {/* Button to toggle the comment section's expanded state */}
      <Button
        variant="ghost"
        className="p-0 h-auto text-gray-500 hover:text-gray-700"
        onClick={() => setExpanded(!expanded)}
```

```
      >
        <MessageSquare className="h-4 w-4 mr-2" />
        {expanded ? "Hide Comments" : `${totalCommentCount} Comments`}
      </Button>
      {/* Conditional rendering: Comment list and new comment form - Shown when
expanded is true */}
      {expanded && (
        <div className="mt-4 space-y-4">
          {/* Map through comments and render each as a CommentItem */}
          {comments.map((comment) => (
            <CommentItem
              key={comment.id}
              doubtId={doubtId}
              comment={comment}
              onAddComment={onAddComment}
              onEditComment={onEditComment}
              onDeleteComment={onDeleteComment}
            />
          ))}
          {/* Form for adding a new top-level comment */}
          <form onSubmit={handleSubmit} className="flex space-x-2">
            <Input
              value={newComment}
              onChange={(e) => setNewComment(e.target.value)}
              placeholder="Add a comment"
              className="flex-grow"
            />
            <Button type="submit">Comment</Button>
          </form>
        </div>
      )}
    </div>
  );
}

// Helper function to count nested replies for a comment
function countReplies(comment: Comment): number {
  return comment.replies.reduce(
    (acc, reply) => acc + 1 + countReplies(reply),
    0
```

```
  );
}
```

File: src\components\doubts-tracker\DoubtCard.tsx

```
import React, { useState, useRef, useEffect } from "react";
import { useUser } from "@clerk/nextjs";
import { db } from "@/lib/firebase";
import { collection, query, orderBy, onSnapshot } from "firebase/firestore";
import {
  Card,
  CardContent,
  CardFooter,
  CardHeader,
  CardTitle,
} from "@/components/ui/card";
import { Button } from "@/components/ui/button";
import { Input } from "@/components/ui/input";
import { Textarea } from "@/components/ui/textarea";
import {
  ArrowBigUp,
  ArrowBigDown,
  CheckCircle2,
  Edit,
  RefreshCw,
  Save,
  Trash2,
} from "lucide-react";
import CommentSection, { Comment } from "./CommentSection";
import {
  Dialog,
  DialogContent,
  DialogDescription,
  DialogHeader,
  DialogTitle,
  DialogTrigger,
} from "@/components/ui/dialog";
```

```
interface Doubt {
  id: string;
  title: string;
  description: string;
  upvotes: number;
  downvotes: number;
  resolved: boolean;
  solution?: string;
  comments: Comment[];
}

interface DoubtCardProps {
  doubt: Doubt;
  onResolve: (id: string, solution: string) => void;
  onReopen: (id: string) => void;
  onUpvote: (id: string) => void;
  onDownvote: (id: string) => void;
  onAddComment: (
    doubtId: string,
    commentText: string,
    parentCommentId?: string
  ) => void;
  onEditDoubt: (id: string, title: string, description: string) => void;
  onEditComment: (doubtId: string, commentId: string, newText: string) => void;
  onDeleteComment: (doubtId: string, commentId: string) => void;
  onEditSolution: (id: string, newSolution: string) => void;
  onDeleteDoubt: (id: string) => void;
}

export default function DoubtCard({
  doubt,
  onResolve,
  onReopen,
  onUpvote,
  onDownvote,
  onAddComment,
  onEditDoubt,
  onEditComment,
  onDeleteComment,
  onEditSolution,
```

```
  onDeleteDoubt,
}: DoubtCardProps) {
  // User authentication state
  const { user } = useUser();
  // State to manage whether the full description is expanded or collapsed
  const [expanded, setExpanded] = useState(false);
  // State to hold the solution text when resolving an doubt
  const [solution, setSolution] = useState("");
  // State for editing the doubt title, initialized with the current doubt title
  const [editTitle, setEditTitle] = useState(doubt.title);
  // State for editing the doubt description, initialized with the current doubt description
  const [editDescription, setEditDescription] = useState(doubt.description);
  // State to track if the solution is currently being edited
  const [isEditingSolution, setIsEditingSolution] = useState(false);
  // State to hold the edited solution text, initialized with the current solution
  const [editedSolution, setEditedSolution] = useState(doubt.solution || "");
  // State to track if the doubt itself is being edited (title and description)
  const [isEditing, setIsEditing] = useState(false);
  // State to control the visibility of the delete confirmation dialog
  const [showDeleteConfirm, setShowDeleteConfirm] = useState(false);
  // State to track if there are changes in the edit form to prevent accidental navigation
away
  const [hasEditChanges, setHasEditChanges] = useState(false);
  // This state holds the nested comment tree built from the flat list of comments fetched
from Firestore.
  const [nestedComments, setNestedComments] = useState<Comment[]>([]);
  // useRef for the title edit input to detect outside clicks for cancelling edit mode
  const editTitleRef = useRef<HTMLDivElement>(null);
  // useRef for the solution edit input to detect outside clicks for cancelling edit mode
  const editSolutionRef = useRef<HTMLDivElement>(null);

  // Load flat comments from Firestore and build the nested comment tree structure.
  useEffect(() => {
    if (!user) return;
    // Reference to the comments subcollection for this doubt
    const commentsRef = collection(
      db,
      "users",
      user.id,
      "nugget",
```

```
      doubt.id,
      "comments"
    );
    // Query to fetch comments ordered by creation time
    const q = query(commentsRef, orderBy("createdAt", "asc"));
    // Subscribe to comment updates using onSnapshot to get real-time updates
    const unsubscribe = onSnapshot(q, (snapshot) => {
      const loadedComments: Comment[] = [];
      snapshot.forEach((docSnap) => {
        loadedComments.push({
          id: docSnap.id,
          text: docSnap.data().text,
          parentId: docSnap.data().parentId || null,
          replies: [], // Initialize replies as empty array, to be populated in nesting logic
        });
      });
      // Build nested structure from flat comments
      const commentMap = new Map<string, Comment>();
      const topLevelComments: Comment[] = [];
      loadedComments.forEach((comment) => {
        commentMap.set(comment.id, { ...comment, replies: [] }); // Store comment in map
for easy access
      });
      commentMap.forEach((comment) => {
        if (comment.parentId) {
          // If comment has a parentId, find the parent comment and add this comment as a
reply
          const parent = commentMap.get(comment.parentId);
          if (parent) {
            parent.replies.push(comment);
          } else {
            // If parent not found (e.g., orphaned comment), treat as top-level
            topLevelComments.push(comment);
          }
        } else {
          // If no parentId, it's a top-level comment
          topLevelComments.push(comment);
        }
      });
      setNestedComments(topLevelComments); // Update state with nested comments
```

```
    });
    return () => unsubscribe(); // Unsubscribe from snapshot listener when component
unmounts
  }, [user, doubt.id]);

  // Handler for input changes in the title edit field
  const handleTitleChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    setEditTitle(e.target.value);
    setHasEditChanges(e.target.value !== doubt.title); // Track changes for "are you
sure?" logic
  };

  // Handler for textarea changes in the description edit field
  const handleDescriptionChange = (
    e: React.ChangeEvent<HTMLTextAreaElement>
  ) => {
    setEditDescription(e.target.value);
    setHasEditChanges(e.target.value !== doubt.description); // Track changes for "are
you sure?" logic
  };

  // useEffect hook to handle clicks outside the edit title/solution area to cancel editing
  useEffect(() => {
    const handleClickOutside = (event: MouseEvent) => {
      if (!hasEditChanges) {
        // If no changes, clicking outside should cancel edit mode
        if (
          editTitleRef.current &&
          !editTitleRef.current.contains(event.target as Node)
        ) {
          setIsEditing(false); // Exit title edit mode
        }
        if (
          editSolutionRef.current &&
          !editSolutionRef.current.contains(event.target as Node)
        ) {
          setIsEditingSolution(false); // Exit solution edit mode
        }
      }
    };
```

```
  document.addEventListener("mousedown", handleClickOutside);
  return () => document.removeEventListener("mousedown", handleClickOutside);
}, [hasEditChanges]);

// Function to handle resolving an doubt - calls the onResolve prop function
const handleResolve = () => {
  onResolve(doubt.id, solution);
  setSolution(""); // Clear the solution input after resolving
};

// Function to handle initiating the edit mode for the doubt (title/description)
const handleEdit = () => {
  onEditDoubt(doubt.id, editTitle, editDescription);
};

// Function to handle saving the edited solution
const handleEditSolution = () => {
  onEditSolution(doubt.id, editedSolution);
  setIsEditingSolution(false); // Exit solution edit mode after saving
};

return (
  <Card className={`${doubt.resolved ? "bg-gray-50" : "bg-white"}`}>
    <CardHeader className="flex flex-row items-start space-x-4 pb-2">
      {/* Upvote/Downvote buttons and score display */}
      <div className="flex flex-col items-center space-y-1">
        <Button
          variant="ghost"
          size="sm"
          className="px-0"
          onClick={() => onUpvote(doubt.id)}
        >
          <ArrowBigUp
            className={`h-5 w-5 ${
              doubt.upvotes > doubt.downvotes
                ? "text-orange-500"
                : "text-gray-500"
            }`}
          />
        </Button>
```

```
  <span className="text-sm font-bold">
   {doubt.upvotes - doubt.downvotes}
  </span>
  <Button
   variant="ghost"
   size="sm"
   className="px-0"
   onClick={() => onDownvote(doubt.id)}
  >
   <ArrowBigDown
    className={`h-5 w-5 ${
     doubt.downvotes > doubt.upvotes
      ? "text-blue-500"
      : "text-gray-500"
    }`}
   />
  </Button>
 </div>
 <div className="flex-grow">
  <CardTitle className="text-lg break-all overflow-wrap-anywhere">
   {doubt.title}
  </CardTitle>
  <p className="text-sm text-gray-500">
   {doubt.resolved ? "Resolved" : "Open"} •{" "}
   {nestedComments.reduce((acc, c) => acc + 1 + countReplies(c), 0)}{" "}
   comments
  </p>
 </div>
 <div className="flex space-x-2">
  {/* Edit Doubt Dialog */}
  <Dialog>
   <DialogTrigger asChild>
    <Button variant="outline" size="sm">
     <Edit className="h-4 w-4 mr-2" />
     Edit
    </Button>
   </DialogTrigger>
   <DialogContent>
    <DialogHeader>
     <DialogTitle>Edit Doubt</DialogTitle>
```

```
      <DialogDescription>
        Make changes to your doubt here. Click save when you're done.
      </DialogDescription>
    </DialogHeader>
    <div className="space-y-4 py-4">
      <div className="space-y-2">
        <Input
          id="title"
          value={editTitle}
          onChange={handleTitleChange}
          placeholder="Doubt title"
        />
      </div>
      <div className="space-y-2">
        <Textarea
          id="description"
          value={editDescription}
          onChange={handleDescriptionChange}
          placeholder="Detailed description"
        />
      </div>
    </div>
    <Button onClick={handleEdit}>Save Changes</Button>
  </DialogContent>
</Dialog>
{/* Delete Doubt Dialog */}
<Dialog open={showDeleteConfirm} onOpenChange={setShowDeleteConfirm}>
  <DialogTrigger asChild>
    <Button
      variant="ghost"
      size="sm"
      className="text-red-600 hover:text-red-700 hover:bg-red-50"
    >
      <Trash2 className="h-4 w-4" />
    </Button>
  </DialogTrigger>
  <DialogContent>
    <DialogHeader>
      <DialogTitle>Delete Doubt</DialogTitle>
      <DialogDescription>
```

```
              Are you sure you want to delete this doubt? This action cannot
              be undone.
            </DialogDescription>
          </DialogHeader>
          <div className="flex justify-end space-x-2">
            <Button
              variant="outline"
              onClick={() => setShowDeleteConfirm(false)}
            >
              Cancel
            </Button>
            <Button
              variant="destructive"
              onClick={() => {
                onDeleteDoubt(doubt.id);
                setShowDeleteConfirm(false);
              }}
            >
              Delete
            </Button>
          </div>
        </DialogContent>
      </Dialog>
    </div>
    {/* Reopen Doubt Button - only shown if the doubt is resolved */}
    {doubt.resolved ? (
      <Button variant="outline" size="sm" onClick={() => onReopen(doubt.id)}>
        <RefreshCw className="h-4 w-4 mr-2" />
        Reopen
      </Button>
    ) : null}
  </CardHeader>
  <CardContent>
    <p className="text-sm whitespace-pre-wrap break-words">
      {/* Conditionally render full description or truncated description based on
expanded state */}
      {expanded ? doubt.description : `${doubt.description.slice(0, 200)}...`}
      {doubt.description.length > 200 && (
        <Button
          variant="link"
```

```
            className="p-0 h-auto text-blue-500"
            onClick={() => setExpanded(!expanded)}
          >
            {expanded ? "Read Less" : "Read More"}
          </Button>
        )}
      </p>
      {/* Solution display - only shown if the doubt is resolved and has a solution */}
      {doubt.resolved && doubt.solution && (
        <div className="mt-2 p-2 bg-green-50 rounded border border-green-200">
          {/* Conditionally render solution text or solution edit input based on
isEditingSolution state */}
          {isEditingSolution ? (
            <div className="flex space-x-2">
              <Input
                value={editedSolution}
                onChange={(e) => setEditedSolution(e.target.value)}
                className="flex-grow"
              />
              <Button size="sm" onClick={handleEditSolution}>
                <Save className="h-4 w-4" />
              </Button>
            </div>
          ) : (
            <div className="flex justify-between items-start">
              <p className="text-sm font-medium text-green-800 whitespace-pre-wrap
break-words">
                Solution: {doubt.solution}
              </p>
              <Button
                variant="ghost"
                size="sm"
                onClick={() => setIsEditingSolution(true)}
              >
                <Edit className="h-4 w-4" />
              </Button>
            </div>
          )}
        </div>
      )}
```

```
      </CardContent>
      <CardFooter className="flex flex-col items-start space-y-2">
       {/* Resolve input and button - only shown if the doubt is not resolved */}
       {!doubt.resolved && (
         <div className="flex w-full space-x-2">
           <Input
             value={solution}
             onChange={(e) => setSolution(e.target.value)}
             placeholder="Enter solution"
           />
           <Button onClick={handleResolve} disabled={!solution}>
             <CheckCircle2 className="h-4 w-4 mr-2" />
             Resolve
           </Button>
         </div>
       )}
       {/* Comment section component for displaying and adding comments */}
       <CommentSection
         doubtId={doubt.id}
         comments={nestedComments}
         onAddComment={(text, parentId) =>
           onAddComment(doubt.id, text, parentId)
         }
         onEditComment={(commentId, newText) =>
           onEditComment(doubt.id, commentId, newText)
         }
         onDeleteComment={(commentId) => onDeleteComment(doubt.id, commentId)}
       />
      </CardFooter>
    </Card>
  );
}

// Helper function to recursively count nested replies for a comment
function countReplies(comment: Comment): number {
  return comment.replies.reduce(
    (acc, reply) => acc + 1 + countReplies(reply),
    0
  );
}
```

File: src\components\doubts-tracker\DoubtList.tsx

```
import React from "react";
import DoubtCard from "./DoubtCard";

// Define the Comment interface to structure comment data.
interface Comment {
  id: string;
  text: string;
  replies: Comment[];
}

// Define the Doubt interface to structure doubt data, including comments.
interface Doubt {
  id: string;
  title: string;
  description: string;
  upvotes: number;
  downvotes: number;
  resolved: boolean;
  solution?: string;
  comments: Comment[];
}

// Define the props for the DoubtList component.
interface DoubtListProps {
  doubts: Doubt[];
  onResolve: (id: string, solution: string) => void;
  onReopen: (id: string) => void;
  onUpvote: (id: string) => void;
  onDownvote: (id: string) => void;
  onAddComment: (
    doubtId: string,
    commentText: string,
    parentCommentId?: string
  ) => void;
  onEditDoubt: (id: string, title: string, description: string) => void;
```

```
  onEditComment: (doubtId: string, commentId: string, newText: string) => void;
  onDeleteComment: (doubtId: string, commentId: string) => void;
  onEditSolution: (id: string, newSolution: string) => void;
  onDeleteDoubt: (id: string) => void;
}

// DoubtList component: Displays a list of DoubtCards.
export default function DoubtList({
  doubts, // Array of doubt objects to be displayed.
  onResolve, // Function to handle resolving an doubt.
  onReopen, // Function to handle reopening a resolved doubt.
  onUpvote, // Function to handle upvoting an doubt.
  onDownvote, // Function to handle downvoting an doubt.
  onAddComment, // Function to handle adding a comment to an doubt.
  onEditDoubt, // Function to handle editing an doubt's details.
  onEditComment, // Function to handle editing a comment.
  onDeleteComment, // Function to handle deleting a comment.
  onEditSolution, // Function to handle editing the solution of an doubt.
  onDeleteDoubt, // Function to handle deleting an doubt.
}: DoubtListProps) {
  // Sort doubts based on vote difference (upvotes - downvotes) in descending order,
  // and then by ID in descending order as a secondary sort.
  const sortedDoubts = [...doubts].sort(
    (a, b) =>
      b.upvotes - b.downvotes - (a.upvotes - a.downvotes) ||
      Number(b.id) - Number(a.id)
  );

  return (
    <div className="space-y-4">
      {/* Map through the sorted doubts and render an DoubtCard for each doubt. */}
      {sortedDoubts.map((doubt) => (
        <DoubtCard
          key={doubt.id} // Unique key for each DoubtCard, using doubt ID.
          doubt={doubt} // Pass the current doubt object as props to DoubtCard.
          onResolve={onResolve} // Pass the resolve handler.
          onReopen={onReopen} // Pass the reopen handler.
          onUpvote={onUpvote} // Pass the upvote handler.
          onDownvote={onDownvote} // Pass the downvote handler.
          onAddComment={onAddComment} // Pass the add comment handler.
```

```
          onEditDoubt={onEditDoubt} // Pass the edit doubt handler.
          onEditComment={onEditComment} // Pass the edit comment handler.
          onDeleteComment={onDeleteComment} // Pass the delete comment handler.
          onEditSolution={onEditSolution} // Pass the edit solution handler.
          onDeleteDoubt={onDeleteDoubt} // Pass the delete doubt handler.
        />
      ))}
    </div>
  );
}
```

File: src\components\doubts-tracker\NewDoubtForm.tsx

```
import React, { useState } from "react";
import { Button } from "@/components/ui/button";
import { Input } from "@/components/ui/input";
import { Textarea } from "@/components/ui/textarea";

interface NewDoubtFormProps {
  onSubmit: (title: string, description: string) => void;
}

/**
 * NewDoubtForm Component
 *
 * This component renders a form for creating a new doubt.
 * It includes input fields for the doubt's title and description,
 * and a submit button to trigger the creation process.
 */
export default function NewDoubtForm({ onSubmit }: NewDoubtFormProps) {
  // State variables to manage the input values for the doubt's title and description.
  const [title, setTitle] = useState("");
  const [description, setDescription] = useState("");

  /**
   * handleSubmit function
   *
   * Handles the form submission event.
```

```
   * It prevents the default form submission behavior, checks if both title and description are filled,
   * and then calls the `onSubmit` prop function passed from the parent component to handle doubt creation.
   * After successful submission, it resets the title and description input fields.
   *
   * @param {React.FormEvent} e - The form submission event.
   */
  const handleSubmit = (e: React.FormEvent) => {
    e.preventDefault();
    if (title && description) {
      onSubmit(title, description);
      setTitle("");
      setDescription("");
    }
  };

  return (
    // Form element that handles the submission of new doubt data.
    <form onSubmit={handleSubmit} className="space-y-4">
      {/* Container for the title input field */}
      <div className="space-y-2">
        {/* Input field for the title of the doubt */}
        <Input
          value={title}
          onChange={(e) => setTitle(e.target.value)}
          placeholder="Doubt title"
          required
        />
      </div>
      {/* Container for the description textarea */}
      <div className="space-y-2">
        {/* Textarea for the detailed description of the doubt */}
        <Textarea
          value={description}
          onChange={(e) => setDescription(e.target.value)}
          placeholder="Detailed description"
          required
        />
      </div>
```

```
      {/* Button to submit the form and create a new doubt */}
      <Button type="submit">Create Doubt</Button>
    </form>
  );
}
```

Curiosity Space (Idea Tracker) Module

```
==========
Directory: src\components
==========

File: src\components\IdeaTracker.tsx
--------------------

"use client";
import React, { useState, useEffect } from "react";
import IdeaList from "@/components/curiosity-space/IdeaList";
import NewIdeaForm from "@/components/curiosity-space/NewIdeaForm";
import { Button } from "@/components/ui/button";
import { Tabs, TabsContent, TabsList, TabsTrigger } from "@/components/ui/tabs";
import {
  Dialog,
  DialogContent,
  DialogDescription,
  DialogHeader,
  DialogTitle,
  DialogTrigger,
} from "@/components/ui/dialog";
import { Plus } from "lucide-react";
import { useUser } from "@clerk/nextjs";
import { db } from "@/lib/firebase";
import {
  collection,
  addDoc,
  updateDoc,
  deleteDoc,
  query,
  orderBy,
  onSnapshot,
  increment,
  serverTimestamp,
  doc,
} from "firebase/firestore";

export interface Comment {
```

```
  id: string;
  text: string;
  parentId?: string | null;
  replies: Comment[];
}

export interface Idea {
  id: string;
  title: string;
  description: string;
  upvotes: number;
  downvotes: number;
  resolved: boolean;
  solution?: string;
  comments: Comment[];
}

export default function IdeaTracker() {
  // User authentication state
  const { user } = useUser();
  // State to hold the list of ideas.
  const [ideas, setIdeas] = useState<Idea[]>([]);

  // useEffect hook to fetch ideas from Firestore when the component mounts or when
the user changes.
  useEffect(() => {
    if (!user) return;
    // Reference to the 'ideas' collection within the user's document in Firestore.
    const ideasRef = collection(db, "users", user.id, "ideas");
    // Create a query to fetch ideas, ordered by creation time in descending order.
    const q = query(ideasRef, orderBy("createdAt", "desc"));
    // Subscribe to real-time updates from Firestore using onSnapshot.
    const unsubscribe = onSnapshot(q, (snapshot) => {
      const loadedIdeas: Idea[] = [];
      // Iterate over each document in the snapshot.
      snapshot.forEach((docSnap) => {
        // Extract data from each document and shape it into an Idea object.
        loadedIdeas.push({
          id: docSnap.id,
          title: docSnap.data().title,
```

```
          description: docSnap.data().description,
          upvotes: docSnap.data().upvotes,
          downvotes: docSnap.data().downvotes,
          resolved: docSnap.data().resolved,
          solution: docSnap.data().solution,
          comments: [], // Initialize comments as an empty array, they are fetched
separately if needed.
        });
      });
      // Update the ideas state with the fetched ideas.
      setIdeas(loadedIdeas);
    });
    // Return the unsubscribe function to detach the listener when the component
unmounts.
    return () => unsubscribe();
  }, [user]); // Dependency array ensures this effect runs when 'user' object changes.

  // Function to add a new idea to Firestore.
  const addIdea = async (title: string, description: string) => {
    if (!user) return;
    try {
      // Get reference to the 'ideas' collection for the current user.
      const ideasRef = collection(db, "users", user.id, "ideas");
      // Add a new document to the 'ideas' collection with the provided title and
description.
      await addDoc(ideasRef, {
        title,
        description,
        upvotes: 0,
        downvotes: 0,
        resolved: false,
        solution: "",
        createdAt: serverTimestamp(), // Use Firestore server timestamp for creation time.
      });
    } catch (error) {
      console.error("Error creating idea:", error);
    }
  };

  // Function to resolve an idea in Firestore.
```

```
  const resolveIdea = async (id: string, solution: string) => {
    if (!user) return;
    try {
      // Get reference to the specific idea document within the user's 'ideas' collection.
      const ideaRef = doc(db, "users", user.id, "ideas", id);
      // Update the idea document to mark it as resolved and add the provided solution.
      await updateDoc(ideaRef, { resolved: true, solution });
    } catch (error) {
      console.error("Error resolving idea:", error);
    }
  };

  // Function to reopen a resolved idea in Firestore.
  const reopenIdea = async (id: string) => {
    if (!user) return;
    try {
      // Get reference to the specific idea document.
      const ideaRef = doc(db, "users", user.id, "ideas", id);
      // Update the idea document to mark it as not resolved and clear the solution.
      await updateDoc(ideaRef, { resolved: false, solution: "" });
    } catch (error) {
      console.error("Error reopening idea:", error);
    }
  };

  // Function to upvote an idea in Firestore.
  const upvoteIdea = async (id: string) => {
    if (!user) return;
    try {
      // Get reference to the specific idea document.
      const ideaRef = doc(db, "users", user.id, "ideas", id);
      // Increment the upvotes count for the idea document using Firestore's increment
operator.
      await updateDoc(ideaRef, { upvotes: increment(1) });
    } catch (error) {
      console.error("Error upvoting idea:", error);
    }
  };

  // Function to downvote an idea in Firestore.
```

```
  const downvoteIdea = async (id: string) => {
    if (!user) return;
    try {
      // Get reference to the specific idea document.
      const ideaRef = doc(db, "users", user.id, "ideas", id);
      // Increment the downvotes count for the idea document using Firestore's increment
operator.
      await updateDoc(ideaRef, { downvotes: increment(1) });
    } catch (error) {
      console.error("Error downvoting idea:", error);
    }
  };

  // Function to add a comment to an idea in Firestore.
  const addComment = async (
    ideaId: string,
    commentText: string,
    parentId?: string
  ) => {
    if (!user) return;
    try {
      // Get reference to the 'comments' subcollection within the specific idea document.
      const commentsRef = collection(
        db,
        "users",
        user.id,
        "ideas",
        ideaId,
        "comments"
      );
      // Add a new comment document to the 'comments' subcollection.
      await addDoc(commentsRef, {
        text: commentText,
        parentId: parentId || null, // Store parentId if it's a reply, otherwise null for top-level
comments.
        createdAt: serverTimestamp(), // Use Firestore server timestamp for comment
creation time.
      });
    } catch (error) {
      console.error("Error adding comment:", error);
```

```
    }
  };

  // Function to edit an existing comment in Firestore.
  const editComment = async (
    ideaId: string,
    commentId: string,
    newText: string
  ) => {
    if (!user) return;
    try {
      // Get reference to the specific comment document within the 'comments'
subcollection.
      const commentRef = doc(
        db,
        "users",
        user.id,
        "ideas",
        ideaId,
        "comments",
        commentId
      );
      // Update the text field of the comment document with the new text.
      await updateDoc(commentRef, { text: newText });
    } catch (error) {
      console.error("Error editing comment:", error);
    }
  };

  // Function to delete a comment from Firestore.
  const deleteComment = async (ideaId: string, commentId: string) => {
    if (!user) return;
    try {
      // Get reference to the specific comment document to be deleted.
      const commentRef = doc(
        db,
        "users",
        user.id,
        "ideas",
        ideaId,
```

```
      "comments",
      commentId
    );
    // Delete the comment document from Firestore.
    await deleteDoc(commentRef);
  } catch (error) {
    console.error("Error deleting comment:", error);
  }
};

// Function to edit an idea's title and description in Firestore.
const editIdea = async (id: string, title: string, description: string) => {
  if (!user) return;
  try {
    // Get reference to the specific idea document.
    const ideaRef = doc(db, "users", user.id, "ideas", id);
    // Update the title and description fields of the idea document.
    await updateDoc(ideaRef, { title, description });
  } catch (error) {
    console.error("Error editing idea:", error);
  }
};

// Function to edit the solution of a resolved idea in Firestore.
const editSolution = async (id: string, newSolution: string) => {
  if (!user) return;
  try {
    // Get reference to the specific idea document.
    const ideaRef = doc(db, "users", user.id, "ideas", id);
    // Update the solution field of the idea document with the new solution.
    await updateDoc(ideaRef, { solution: newSolution });
  } catch (error) {
    console.error("Error editing solution:", error);
  }
};

// Function to delete an idea from Firestore.
const deleteIdea = async (id: string) => {
  if (!user) return;
  try {
```

```
      // Get reference to the specific idea document to be deleted.
      const ideaRef = doc(db, "users", user.id, "ideas", id);
      // Delete the idea document from Firestore.
      await deleteDoc(ideaRef);
    } catch (error) {
      console.error("Error deleting idea:", error);
    }
  };

  return (
    <div className="container mx-auto p-4 max-w-4xl">
      {/* Header section of the Idea Tracker, containing the title and 'New Idea' button. */}
      <div className="flex justify-between items-center mb-4">
        <h1 className="text-3xl font-bold text-orange-600">Ideas Tracker</h1>
        {/* Dialog component for creating a new idea, triggered by the 'New Idea' button. */}
        <Dialog>
          <DialogTrigger asChild>
            <Button>
              <Plus className="mr-2 h-4 w-4" /> New Idea
            </Button>
          </DialogTrigger>
          <DialogContent>
            <DialogHeader>
              <DialogTitle>Create a New Idea</DialogTitle>
              <DialogDescription>
                Add a new idea to track and resolve.
              </DialogDescription>
            </DialogHeader>
            {/* NewIdeaForm component for handling the input and submission of new
ideas. */}
            <NewIdeaForm onSubmit={addIdea} />
          </DialogContent>
        </Dialog>
      </div>
      {/* Tabs component to separate 'Open Ideas' and 'Resolved Ideas'. */}
      <Tabs defaultValue="open" className="mt-4">
        <TabsList className="grid w-full grid-cols-2">
          <TabsTrigger value="open">Open Ideas</TabsTrigger>
          <TabsTrigger value="resolved">Resolved Ideas</TabsTrigger>
        </TabsList>
```

```
        {/* Content for the 'Open Ideas' tab. */}
        <TabsContent value="open">
          {/* IdeaList component to display the list of open ideas. */}
          <IdeaList
            ideas={ideas.filter((d) => !d.resolved)} // Filter ideas to show only open ones.
            onResolve={resolveIdea}
            onReopen={reopenIdea}
            onUpvote={upvoteIdea}
            onDownvote={downvoteIdea}
            onAddComment={addComment}
            onEditIdea={editIdea}
            onEditComment={editComment}
            onDeleteComment={deleteComment}
            onEditSolution={editSolution}
            onDeleteIdea={deleteIdea}
          />
        </TabsContent>
        {/* Content for the 'Resolved Ideas' tab. */}
        <TabsContent value="resolved">
          {/* IdeaList component to display the list of resolved ideas. */}
          <IdeaList
            ideas={ideas.filter((d) => d.resolved)} // Filter ideas to show only resolved ones.
            onResolve={resolveIdea}
            onReopen={reopenIdea}
            onUpvote={upvoteIdea}
            onDownvote={downvoteIdea}
            onAddComment={addComment}
            onEditIdea={editIdea}
            onEditComment={editComment}
            onDeleteComment={deleteComment}
            onEditSolution={editSolution}
            onDeleteIdea={deleteIdea}
          />
        </TabsContent>
      </Tabs>
    </div>
  );
}
```

==========
Directory: src\app\CuriositySpace
==========

File: src\app\CuriositySpace\page.tsx

```
"use client";

import React from "react";
import IdeaTracker from "@/components/IdeaTracker";
import { BackButton } from "@/components/ui/custom-ui/back-button";

const IdeaTrackerPage: React.FC = () => {
  return (
    <div className="flex flex-col h-screen">
      <header className="flex items-center justify-between p-4 border-b">
        <BackButton />
      </header>
      <main className="flex-1 overflow-auto">
        <IdeaTracker />
      </main>
    </div>
  );
};

export default IdeaTrackerPage;
```

==========
Directory: src\components\curiosity-space
==========

File: src\components\curiosity-space\CommentSection.tsx

```
import React, { useState, useRef, useEffect } from "react";
import { Button } from "@/components/ui/button";
import { Input } from "@/components/ui/input";
import {
```

```
  MessageSquare,
  Edit,
  Save,
  Reply,
  Trash2,
  MoreVertical,
  X,
} from "lucide-react";
import {
  DropdownMenu,
  DropdownMenuContent,
  DropdownMenuItem,
  DropdownMenuTrigger,
} from "@/components/ui/dropdown-menu";
import { useUser } from "@clerk/nextjs";

// Define the Comment interface to structure comment data
export interface Comment {
  id: string;
  text: string;
  parentId?: string | null;
  replies: Comment[];
}

// Define the props for the CommentSection component
interface CommentSectionProps {
  ideaId: string; // ID of the idea to which the comments belong
  comments: Comment[]; // Array of comments to display
  onAddComment: (text: string, parentId?: string) => void; // Function to handle adding a
new comment
  onEditComment: (commentId: string, newText: string) => void; // Function to handle
editing an existing comment
  onDeleteComment: (commentId: string) => void; // Function to handle deleting a
comment
}

// Define the props for the CommentItem component, which represents a single
comment
interface CommentItemProps {
  ideaId: string; // ID of the idea, passed down for comment actions
```

```
  comment: Comment; // The comment object to render
  onAddComment: (text: string, parentId: string) => void; // Function to add a reply to a
comment
  onEditComment: (commentId: string, newText: string) => void; // Function to edit a
comment (passed down)
  onDeleteComment: (commentId: string) => void; // Function to delete a comment
(passed down)
}

/**
 * CommentItem Component: Represents a single comment and its replies.
 * Handles rendering comment text, edit/delete options, and reply functionality.
 */
const CommentItem: React.FC<CommentItemProps> = ({
  ideaId,
  comment,
  onAddComment,
  onEditComment,
  onDeleteComment,
}) => {
  // User authentication state from Clerk
  const { user } = useUser();
  // State to manage whether the comment is in edit mode
  const [isEditing, setIsEditing] = useState(false);
  // State to hold the text being edited in the comment
  const [editedText, setEditedText] = useState(comment.text);
  // State to manage whether the reply input is shown for this comment
  const [isReplying, setIsReplying] = useState(false);
  // State to hold the text for a new reply
  const [replyText, setReplyText] = useState("");
  // useRef to manage focus outside of the reply input to close it
  const replyRef = useRef<HTMLDivElement>(null);
  // useRef to manage focus outside of the edit input to exit edit mode
  const editRef = useRef<HTMLDivElement>(null);
  // State to track if there are unsaved changes in the edit input
  const [hasChanges, setHasChanges] = useState(false);

  /**
   * Handler for saving the edited comment text.
   * Calls the onEditComment prop function to update the comment and exits edit mode.
```

```
   */
  const handleSaveEdit = () => {
    onEditComment(comment.id, editedText);
    setIsEditing(false);
  };

  /**
   * Handler for cancelling comment editing.
   * Resets the edited text to the original comment text and exits edit mode.
   */
  const handleCancelEdit = () => {
    setEditedText(comment.text);
    setIsEditing(false);
  };

  /**
   * Handler for adding a reply to the comment.
   * Calls the onAddComment prop function with the reply text and the current
comment's ID as the parent.
   * Resets the reply input and closes the reply input area.
   */
  const handleAddReply = () => {
    if (replyText) {
      onAddComment(replyText, comment.id);
      setIsReplying(false);
      setReplyText("");
    }
  };

  /**
   * Handler for input changes in the edit comment input.
   * Updates the editedText state and checks if changes have been made compared to
the original text.
   * @param {React.ChangeEvent<HTMLInputElement>} e - The input change event.
   */
  const handleEditTextChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    setEditedText(e.target.value);
    setHasChanges(e.target.value !== comment.text);
  };
```

```
  /**
   * useEffect hook to handle closing the reply input when clicking outside.
   * Adds a mousedown event listener to the document to detect clicks outside the reply input area.
   */
  useEffect(() => {
    const handleClickOutsideReply = (event: MouseEvent) => {
      if (
        replyRef.current &&
        !replyRef.current.contains(event.target as Node)
      ) {
        if (!replyText) setIsReplying(false);
      }
    };
    document.addEventListener("mousedown", handleClickOutsideReply);
    return () =>
      document.removeEventListener("mousedown", handleClickOutsideReply);
  }, [replyText]); // Dependency array includes replyText to re-run effect when reply text changes.

  /**
   * useEffect hook to handle exiting edit mode when clicking outside the edit input area, if no changes are made.
   * Adds a mousedown event listener to the document to detect clicks outside the edit input area.
   */
  useEffect(() => {
    const handleClickOutsideEdit = (event: MouseEvent) => {
      if (
        !hasChanges &&
        editRef.current &&
        !editRef.current.contains(event.target as Node)
      ) {
        setIsEditing(false);
      }
    };
    document.addEventListener("mousedown", handleClickOutsideEdit);
    return () =>
      document.removeEventListener("mousedown", handleClickOutsideEdit);
```

```
  }, [hasChanges]); // Dependency array includes hasChanges to re-run effect when changes status is updated.

  return (
    <div className="mt-2 pl-4 border-l-2 border-gray-200 group">
      {/* Conditional rendering: Edit mode input or comment text display */}
      {isEditing ? (
        <div ref={editRef} className="flex items-center space-x-2">
          <Input
            value={editedText}
            onChange={handleEditTextChange}
            className="flex-grow"
          />
          <Button size="sm" onClick={handleSaveEdit} title="Save">
            <Save className="h-4 w-4" />
          </Button>
          <Button
            size="sm"
            variant="ghost"
            onClick={handleCancelEdit}
            title="Cancel"
          >
            <X className="h-4 w-4" />
          </Button>
        </div>
      ) : (
        <div className="flex items-center justify-between">
          <p className="text-sm whitespace-pre-wrap break-words max-w-[90%]">
            {comment.text}
          </p>
          {/* Conditional rendering: Action buttons (Reply, Edit, Delete) - Shown on group hover */}
          <div className="flex space-x-2 opacity-0 group-hover:opacity-100 transition-opacity">
            {/* Reply button to toggle reply input */}
            <Button
              size="sm"
              variant="ghost"
              onClick={() => setIsReplying(true)}
              title="Reply"
```

```
      >
        <Reply className="h-4 w-4" />
      </Button>
      {/* Dropdown menu for Edit and Delete actions */}
      <DropdownMenu>
        <DropdownMenuTrigger asChild>
          <Button size="sm" variant="ghost" title="More options">
            <MoreVertical className="h-4 w-4" />
          </Button>
        </DropdownMenuTrigger>
        <DropdownMenuContent>
          <DropdownMenuItem onClick={() => setIsEditing(true)}>
            <Edit className="h-4 w-4 mr-2" />
            Edit
          </DropdownMenuItem>
          <DropdownMenuItem
            className="text-red-600"
            onClick={() => onDeleteComment(comment.id)}
          >
            <Trash2 className="h-4 w-4 mr-2" />
            Delete
          </DropdownMenuItem>
        </DropdownMenuContent>
      </DropdownMenu>
    </div>
  </div>
)}
{/* Conditional rendering: Reply input area */}
{isReplying && (
  <div ref={replyRef} className="mt-2 flex items-center space-x-2">
    <Input
      value={replyText}
      onChange={(e) => setReplyText(e.target.value)}
      placeholder="Write a reply..."
      className="flex-grow"
    />
    <Button size="sm" onClick={handleAddReply}>
      Reply
    </Button>
  </div>
```

```
      )}
      {/* Recursive rendering of replies */}
      {comment.replies &&
        comment.replies.map((reply) => (
          <CommentItem
            key={reply.id}
            ideaId={ideaId}
            comment={reply}
            onAddComment={onAddComment}
            onEditComment={onEditComment}
            onDeleteComment={onDeleteComment}
          />
        ))}
    </div>
  );
};

/**
 * CommentSection Component: Renders the comment section, including the list of comments and the new comment input form.
 * Manages the overall display of comments and handles adding new top-level comments.
 */
export default function CommentSection({
  ideaId,
  comments,
  onAddComment,
  onEditComment,
  onDeleteComment,
}: CommentSectionProps) {
  // State to hold the text for a new top-level comment
  const [newComment, setNewComment] = useState("");
  // State to manage the expanded/collapsed state of the comment section
  const [expanded, setExpanded] = useState(false);

  /**
   * Handler for submitting a new top-level comment.
   * Prevents default form submission and calls the onAddComment prop function to add the new comment.
   * Resets the new comment input area after submission.
```

```
   * @param {React.FormEvent} e - The form submit event.
   */
  const handleSubmit = (e: React.FormEvent) => {
    e.preventDefault();
    if (newComment) {
      onAddComment(newComment);
      setNewComment("");
    }
  };

  /**
   * Helper function to recursively count all replies for a given comment and its nested
replies.
   * @param {Comment} comment - The comment object to start counting replies from.
   * @returns {number} - The total count of replies for the comment and all its nested
replies.
   */
  const countReplies = (comment: Comment): number => {
    return comment.replies.reduce(
      (acc, reply) => acc + 1 + countReplies(reply),
      0
    );
  };

  // Calculate total comment count (including nested replies) for display purposes
  const totalCommentCount = comments.reduce(
    (acc, c) => acc + 1 + countReplies(c),
    0
  );

  return (
    <div className="w-full">
      {/* Button to toggle the comment section's expanded state */}
      <Button
        variant="ghost"
        className="p-0 h-auto text-gray-500 hover:text-gray-700"
        onClick={() => setExpanded(!expanded)}
      >
        <MessageSquare className="h-4 w-4 mr-2" />
        {expanded ? "Hide Comments" : `${totalCommentCount} Comments`}
```

```
      </Button>
      {/* Conditional rendering: Comment list and new comment form - Shown when
expanded is true */}
      {expanded && (
        <div className="mt-4 space-y-4">
          {/* Map through comments and render each as a CommentItem */}
          {comments.map((comment) => (
            <CommentItem
              key={comment.id}
              ideaId={ideaId}
              comment={comment}
              onAddComment={onAddComment}
              onEditComment={onEditComment}
              onDeleteComment={onDeleteComment}
            />
          ))}
          {/* Form for adding a new top-level comment */}
          <form onSubmit={handleSubmit} className="flex space-x-2">
            <Input
              value={newComment}
              onChange={(e) => setNewComment(e.target.value)}
              placeholder="Add a comment"
              className="flex-grow"
            />
            <Button type="submit">Comment</Button>
          </form>
        </div>
      )}
    </div>
  );
}

// Helper function to count nested replies for a comment
function countReplies(comment: Comment): number {
  return comment.replies.reduce(
    (acc, reply) => acc + 1 + countReplies(reply),
    0
  );
}
```

File: src\components\curiosity-space\IdeaCard.tsx

```
import React, { useState, useRef, useEffect } from "react";
import { useUser } from "@clerk/nextjs";
import { db } from "@/lib/firebase";
import { collection, query, orderBy, onSnapshot } from "firebase/firestore";
import {
  Card,
  CardContent,
  CardFooter,
  CardHeader,
  CardTitle,
} from "@/components/ui/card";
import { Button } from "@/components/ui/button";
import { Input } from "@/components/ui/input";
import { Textarea } from "@/components/ui/textarea";
import {
  ArrowBigUp,
  ArrowBigDown,
  CheckCircle2,
  Edit,
  RefreshCw,
  Save,
  Trash2,
} from "lucide-react";
import CommentSection, { Comment } from "./CommentSection";
import {
  Dialog,
  DialogContent,
  DialogDescription,
  DialogHeader,
  DialogTitle,
  DialogTrigger,
} from "@/components/ui/dialog";

interface Idea {
  id: string;
  title: string;
```

```
  description: string;
  upvotes: number;
  downvotes: number;
  resolved: boolean;
  solution?: string;
  comments: Comment[];
}

interface IdeaCardProps {
  idea: Idea;
  onResolve: (id: string, solution: string) => void;
  onReopen: (id: string) => void;
  onUpvote: (id: string) => void;
  onDownvote: (id: string) => void;
  onAddComment: (
    ideaId: string,
    commentText: string,
    parentCommentId?: string
  ) => void;
  onEditIdea: (id: string, title: string, description: string) => void;
  onEditComment: (ideaId: string, commentId: string, newText: string) => void;
  onDeleteComment: (ideaId: string, commentId: string) => void;
  onEditSolution: (id: string, newSolution: string) => void;
  onDeleteIdea: (id: string) => void;
}

export default function IdeaCard({
  idea,
  onResolve,
  onReopen,
  onUpvote,
  onDownvote,
  onAddComment,
  onEditIdea,
  onEditComment,
  onDeleteComment,
  onEditSolution,
  onDeleteIdea,
}: IdeaCardProps) {
  // User authentication state
```

```
  const { user } = useUser();
  // State to manage whether the full description is expanded or collapsed
  const [expanded, setExpanded] = useState(false);
  // State to hold the solution text when resolving an idea
  const [solution, setSolution] = useState("");
  // State for editing the idea title, initialized with the current idea title
  const [editTitle, setEditTitle] = useState(idea.title);
  // State for editing the idea description, initialized with the current idea description
  const [editDescription, setEditDescription] = useState(idea.description);
  // State to track if the solution is currently being edited
  const [isEditingSolution, setIsEditingSolution] = useState(false);
  // State to hold the edited solution text, initialized with the current solution
  const [editedSolution, setEditedSolution] = useState(idea.solution || "");
  // State to track if the idea itself is being edited (title and description)
  const [isEditing, setIsEditing] = useState(false);
  // State to control the visibility of the delete confirmation dialog
  const [showDeleteConfirm, setShowDeleteConfirm] = useState(false);
  // State to track if there are changes in the edit form to prevent accidental navigation
away
  const [hasEditChanges, setHasEditChanges] = useState(false);
  // This state holds the nested comment tree built from the flat list of comments fetched
from Firestore.
  const [nestedComments, setNestedComments] = useState<Comment[]>([]);
  // useRef for the title edit input to detect outside clicks for cancelling edit mode
  const editTitleRef = useRef<HTMLDivElement>(null);
  // useRef for the solution edit input to detect outside clicks for cancelling edit mode
  const editSolutionRef = useRef<HTMLDivElement>(null);

  // Load flat comments from Firestore and build the nested comment tree structure.
  useEffect(() => {
    if (!user) return;
    // Reference to the comments subcollection for this idea
    const commentsRef = collection(
      db,
      "users",
      user.id,
      "ideas",
      idea.id,
      "comments"
    );
```

```
    // Query to fetch comments ordered by creation time
    const q = query(commentsRef, orderBy("createdAt", "asc"));
    // Subscribe to comment updates using onSnapshot to get real-time updates
    const unsubscribe = onSnapshot(q, (snapshot) => {
      const loadedComments: Comment[] = [];
      snapshot.forEach((docSnap) => {
        loadedComments.push({
          id: docSnap.id,
          text: docSnap.data().text,
          parentId: docSnap.data().parentId || null,
          replies: [], // Initialize replies as empty array, to be populated in nesting logic
        });
      });
      // Build nested structure from flat comments
      const commentMap = new Map<string, Comment>();
      const topLevelComments: Comment[] = [];
      loadedComments.forEach((comment) => {
        commentMap.set(comment.id, { ...comment, replies: [] }); // Store comment in map
for easy access
      });
      commentMap.forEach((comment) => {
        if (comment.parentId) {
          // If comment has a parentId, find the parent comment and add this comment as a
reply
          const parent = commentMap.get(comment.parentId);
          if (parent) {
            parent.replies.push(comment);
          } else {
            // If parent not found (e.g., orphaned comment), treat as top-level
            topLevelComments.push(comment);
          }
        } else {
          // If no parentId, it's a top-level comment
          topLevelComments.push(comment);
        }
      });
      setNestedComments(topLevelComments); // Update state with nested comments
    });
    return () => unsubscribe(); // Unsubscribe from snapshot listener when component
unmounts
```

```
  }, [user, idea.id]);

  // Handler for input changes in the title edit field
  const handleTitleChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    setEditTitle(e.target.value);
    setHasEditChanges(e.target.value !== idea.title); // Track changes for "are you sure?"
logic
  };

  // Handler for textarea changes in the description edit field
  const handleDescriptionChange = (
    e: React.ChangeEvent<HTMLTextAreaElement>
  ) => {
    setEditDescription(e.target.value);
    setHasEditChanges(e.target.value !== idea.description); // Track changes for "are
you sure?" logic
  };

  // useEffect hook to handle clicks outside the edit title/solution area to cancel editing
  useEffect(() => {
    const handleClickOutside = (event: MouseEvent) => {
      if (!hasEditChanges) {
        // If no changes, clicking outside should cancel edit mode
        if (
          editTitleRef.current &&
          !editTitleRef.current.contains(event.target as Node)
        ) {
          setIsEditing(false); // Exit title edit mode
        }
        if (
          editSolutionRef.current &&
          !editSolutionRef.current.contains(event.target as Node)
        ) {
          setIsEditingSolution(false); // Exit solution edit mode
        }
      }
    };
    document.addEventListener("mousedown", handleClickOutside);
    return () => document.removeEventListener("mousedown", handleClickOutside);
  }, [hasEditChanges]);
```

```
// Function to handle resolving an idea - calls the onResolve prop function
const handleResolve = () => {
  onResolve(idea.id, solution);
  setSolution(""); // Clear the solution input after resolving
};

// Function to handle initiating the edit mode for the idea (title/description)
const handleEdit = () => {
  onEditIdea(idea.id, editTitle, editDescription);
};

// Function to handle saving the edited solution
const handleEditSolution = () => {
  onEditSolution(idea.id, editedSolution);
  setIsEditingSolution(false); // Exit solution edit mode after saving
};

return (
  <Card className={`${idea.resolved ? "bg-gray-50" : "bg-white"}`}>
    <CardHeader className="flex flex-row items-start space-x-4 pb-2">
      {/* Upvote/Downvote buttons and score display */}
      <div className="flex flex-col items-center space-y-1">
        <Button
          variant="ghost"
          size="sm"
          className="px-0"
          onClick={() => onUpvote(idea.id)}
        >
          <ArrowBigUp
            className={`h-5 w-5 ${
              idea.upvotes > idea.downvotes
                ? "text-orange-500"
                : "text-gray-500"
            }`}
          />
        </Button>
        <span className="text-sm font-bold">
          {idea.upvotes - idea.downvotes}
        </span>
```

```
  <Button
    variant="ghost"
    size="sm"
    className="px-0"
    onClick={() => onDownvote(idea.id)}
  >
    <ArrowBigDown
      className={`h-5 w-5 ${
        idea.downvotes > idea.upvotes
          ? "text-blue-500"
          : "text-gray-500"
      }`}
    />
  </Button>
</div>
<div className="flex-grow">
  <CardTitle className="text-lg break-all overflow-wrap-anywhere">
    {idea.title}
  </CardTitle>
  <p className="text-sm text-gray-500">
    {idea.resolved ? "Resolved" : "Open"} •{" "}
    {nestedComments.reduce((acc, c) => acc + 1 + countReplies(c), 0)}{" "}
    comments
  </p>
</div>
<div className="flex space-x-2">
  {/* Edit Idea Dialog */}
  <Dialog>
    <DialogTrigger asChild>
      <Button variant="outline" size="sm">
        <Edit className="h-4 w-4 mr-2" />
        Edit
      </Button>
    </DialogTrigger>
    <DialogContent>
      <DialogHeader>
        <DialogTitle>Edit Idea</DialogTitle>
        <DialogDescription>
          Make changes to your idea here. Click save when you're done.
        </DialogDescription>
```

```
    </DialogHeader>
    <div className="space-y-4 py-4">
     <div className="space-y-2">
      <Input
       id="title"
       value={editTitle}
       onChange={handleTitleChange}
       placeholder="Idea title"
      />
     </div>
     <div className="space-y-2">
      <Textarea
       id="description"
       value={editDescription}
       onChange={handleDescriptionChange}
       placeholder="Detailed description"
      />
     </div>
    </div>
    <Button onClick={handleEdit}>Save Changes</Button>
   </DialogContent>
  </Dialog>
  {/* Delete Idea Dialog */}
  <Dialog open={showDeleteConfirm} onOpenChange={setShowDeleteConfirm}>
   <DialogTrigger asChild>
    <Button
     variant="ghost"
     size="sm"
     className="text-red-600 hover:text-red-700 hover:bg-red-50"
    >
     <Trash2 className="h-4 w-4" />
    </Button>
   </DialogTrigger>
   <DialogContent>
    <DialogHeader>
     <DialogTitle>Delete Idea</DialogTitle>
     <DialogDescription>
      Are you sure you want to delete this idea? This action cannot
      be undone.
     </DialogDescription>
```

```
            </DialogHeader>
            <div className="flex justify-end space-x-2">
              <Button
                variant="outline"
                onClick={() => setShowDeleteConfirm(false)}
              >
                Cancel
              </Button>
              <Button
                variant="destructive"
                onClick={() => {
                  onDeleteIdea(idea.id);
                  setShowDeleteConfirm(false);
                }}
              >
                Delete
              </Button>
            </div>
          </DialogContent>
        </Dialog>
      </div>
      {/* Reopen Idea Button - only shown if the idea is resolved */}
      {idea.resolved ? (
        <Button variant="outline" size="sm" onClick={() => onReopen(idea.id)}>
          <RefreshCw className="h-4 w-4 mr-2" />
          Reopen
        </Button>
      ) : null}
    </CardHeader>
    <CardContent>
      <p className="text-sm whitespace-pre-wrap break-words">
        {/* Conditionally render full description or truncated description based on
expanded state */}
        {expanded ? idea.description : `${idea.description.slice(0, 200)}...`}
        {idea.description.length > 200 && (
          <Button
            variant="link"
            className="p-0 h-auto text-blue-500"
            onClick={() => setExpanded(!expanded)}
          >
```

```
            {expanded ? "Read Less" : "Read More"}
          </Button>
        )}
      </p>
      {/* Solution display - only shown if the idea is resolved and has a solution */}
      {idea.resolved && idea.solution && (
        <div className="mt-2 p-2 bg-green-50 rounded border border-green-200">
          {/* Conditionally render solution text or solution edit input based on
isEditingSolution state */}
          {isEditingSolution ? (
            <div className="flex space-x-2">
              <Input
                value={editedSolution}
                onChange={(e) => setEditedSolution(e.target.value)}
                className="flex-grow"
              />
              <Button size="sm" onClick={handleEditSolution}>
                <Save className="h-4 w-4" />
              </Button>
            </div>
          ) : (
            <div className="flex justify-between items-start">
              <p className="text-sm font-medium text-green-800 whitespace-pre-wrap
break-words">
                Solution: {idea.solution}
              </p>
              <Button
                variant="ghost"
                size="sm"
                onClick={() => setIsEditingSolution(true)}
              >
                <Edit className="h-4 w-4" />
              </Button>
            </div>
          )}
        </div>
      )}
    </CardContent>
    <CardFooter className="flex flex-col items-start space-y-2">
      {/* Resolve input and button - only shown if the idea is not resolved */}
```

```
        {!idea.resolved && (
          <div className="flex w-full space-x-2">
            <Input
              value={solution}
              onChange={(e) => setSolution(e.target.value)}
              placeholder="Enter solution"
            />
            <Button onClick={handleResolve} disabled={!solution}>
              <CheckCircle2 className="h-4 w-4 mr-2" />
              Resolve
            </Button>
          </div>
        )}
        {/* Comment section component for displaying and adding comments */}
        <CommentSection
          ideaId={idea.id}
          comments={nestedComments}
          onAddComment={(text, parentId) =>
            onAddComment(idea.id, text, parentId)
          }
          onEditComment={(commentId, newText) =>
            onEditComment(idea.id, commentId, newText)
          }
          onDeleteComment={(commentId) => onDeleteComment(idea.id, commentId)}
        />
      </CardFooter>
    </Card>
  );
}

// Helper function to recursively count nested replies for a comment
function countReplies(comment: Comment): number {
  return comment.replies.reduce(
    (acc, reply) => acc + 1 + countReplies(reply),
    0
  );
}
```

File: src\components\curiosity-space\IdeaList.tsx

```
import React from "react";
import IdeaCard from "./IdeaCard";

// Define the Comment interface to structure comment data.
interface Comment {
  id: string;
  text: string;
  replies: Comment[];
}

// Define the Idea interface to structure idea data, including comments.
interface Idea {
  id: string;
  title: string;
  description: string;
  upvotes: number;
  downvotes: number;
  resolved: boolean;
  solution?: string;
  comments: Comment[];
}

// Define the props for the IdeaList component.
interface IdeaListProps {
  ideas: Idea[];
  onResolve: (id: string, solution: string) => void;
  onReopen: (id: string) => void;
  onUpvote: (id: string) => void;
  onDownvote: (id: string) => void;
  onAddComment: (
    ideaId: string,
    commentText: string,
    parentCommentId?: string
  ) => void;
  onEditIdea: (id: string, title: string, description: string) => void;
  onEditComment: (ideaId: string, commentId: string, newText: string) => void;
  onDeleteComment: (ideaId: string, commentId: string) => void;
  onEditSolution: (id: string, newSolution: string) => void;
```

```
  onDeleteIdea: (id: string) => void;
}

// IdeaList component: Displays a list of IdeaCards.
export default function IdeaList({
  ideas, // Array of idea objects to be displayed.
  onResolve, // Function to handle resolving an idea.
  onReopen, // Function to handle reopening a resolved idea.
  onUpvote, // Function to handle upvoting an idea.
  onDownvote, // Function to handle downvoting an idea.
  onAddComment, // Function to handle adding a comment to an idea.
  onEditIdea, // Function to handle editing an idea's details.
  onEditComment, // Function to handle editing a comment.
  onDeleteComment, // Function to handle deleting a comment.
  onEditSolution, // Function to handle editing the solution of an idea.
  onDeleteIdea, // Function to handle deleting an idea.
}: IdeaListProps) {
  // Sort ideas based on vote difference (upvotes - downvotes) in descending order,
  // and then by ID in descending order as a secondary sort.
  const sortedIdeas = [...ideas].sort(
    (a, b) =>
      b.upvotes - b.downvotes - (a.upvotes - a.downvotes) ||
      Number(b.id) - Number(a.id)
  );

  return (
    <div className="space-y-4">
      {/* Map through the sorted ideas and render an IdeaCard for each idea. */}
      {sortedIdeas.map((idea) => (
        <IdeaCard
          key={idea.id} // Unique key for each IdeaCard, using idea ID.
          idea={idea} // Pass the current idea object as props to IdeaCard.
          onResolve={onResolve} // Pass the resolve handler.
          onReopen={onReopen} // Pass the reopen handler.
          onUpvote={onUpvote} // Pass the upvote handler.
          onDownvote={onDownvote} // Pass the downvote handler.
          onAddComment={onAddComment} // Pass the add comment handler.
          onEditIdea={onEditIdea} // Pass the edit idea handler.
          onEditComment={onEditComment} // Pass the edit comment handler.
          onDeleteComment={onDeleteComment} // Pass the delete comment handler.
```

```
          onEditSolution={onEditSolution} // Pass the edit solution handler.
          onDeleteIdea={onDeleteIdea} // Pass the delete idea handler.
        />
      ))}
    </div>
  );
}

File: src\components\curiosity-space\NewIdeaForm.tsx
--------------------

import React, { useState } from "react";
import { Button } from "@/components/ui/button";
import { Input } from "@/components/ui/input";
import { Textarea } from "@/components/ui/textarea";

interface NewIdeaFormProps {
  onSubmit: (title: string, description: string) => void;
}

/**
 * NewIdeaForm Component
 *
 * This component renders a form for creating a new idea.
 * It includes input fields for the idea's title and description,
 * and a submit button to trigger the creation process.
 */
export default function NewIdeaForm({ onSubmit }: NewIdeaFormProps) {
  // State variables to manage the input values for the idea's title and description.
  const [title, setTitle] = useState("");
  const [description, setDescription] = useState("");

  /**
   * handleSubmit function
   *
   * Handles the form submission event.
   * It prevents the default form submission behavior, checks if both title and description are filled,
   * and then calls the `onSubmit` prop function passed from the parent component to handle idea creation.
```

```
 * After successful submission, it resets the title and description input fields.
 *
 * @param {React.FormEvent} e - The form submission event.
 */
const handleSubmit = (e: React.FormEvent) => {
  e.preventDefault();
  if (title && description) {
    onSubmit(title, description);
    setTitle("");
    setDescription("");
  }
};

return (
  // Form element that handles the submission of new idea data.
  <form onSubmit={handleSubmit} className="space-y-4">
    {/* Container for the title input field */}
    <div className="space-y-2">
      {/* Input field for the title of the idea */}
      <Input
        value={title}
        onChange={(e) => setTitle(e.target.value)}
        placeholder="Idea title"
        required
      />
    </div>
    {/* Container for the description textarea */}
    <div className="space-y-2">
      {/* Textarea for the detailed description of the idea */}
      <Textarea
        value={description}
        onChange={(e) => setDescription(e.target.value)}
        placeholder="Detailed description"
        required
      />
    </div>
    {/* Button to submit the form and create a new idea */}
    <Button type="submit">Create Idea</Button>
  </form>
);
```

```
}
```

To-Do List Module

```
==========
Directory: src\components
==========

File: src\components\ToDoList.tsx
----------
```

```
"use client";

import React, { useEffect, useState } from "react";
import { useRouter } from "next/navigation";
import { Archive, Plus, X, ArrowLeft } from "lucide-react";
import { Button } from "@/components/ui/button";
import { useUser } from "@clerk/nextjs";
import { doc, getDoc, setDoc, collection } from "firebase/firestore";
import { db } from "@/lib/firebase";
import {
  Sheet,
  SheetContent,
  SheetHeader,
  SheetTitle,
  SheetTrigger,
} from "@/components/ui/sheet";
import { Card, CardContent, CardHeader, CardTitle } from "@/components/ui/card";
import {
  DndContext,
  DragOverlay,
  useSensors,
  useSensor,
  PointerSensor,
  closestCorners,
  DragStartEvent,
  DragEndEvent,
  useDroppable,
  useDraggable,
} from "@dnd-kit/core";
import { useToast } from "@/hooks/use-toast";
import { ToastAction } from "@/components/ui/toast";
```

```
import { BackButton } from "@/components/ui/custom-ui/back-button";

// ——— DATA STRUCTURES ———

// Core data structures for managing todo list state
interface Note {
  id: string;
  content: string;
  checked: boolean;
  // Each note can have nested subtasks.
  subtasks?: Note[];
}

interface Column {
  id: string;
  title: string;
  notes: Note[];
  isEditing?: boolean;
}

interface Section {
  id: string;
  title: string;
  columns: Column[];
  isEditing: boolean;
  isArchived?: boolean;
  isPreviewingArchived?: boolean;
}

interface DragData {
  type: "note" | "column" | "subtask";
  sectionId: string;
  columnId?: string;
  noteId?: string;
  // For subtasks:
  parentNoteId?: string;
  subtaskId?: string;
}

interface ToDoListProps {
```

```
  hideBackButton?: boolean;
  isDashboard?: boolean;
}

// DRAGGABLE/DROPPABLECOMPONENTS

// Core drag and drop components

// Columns
const DraggableColumn = ({
  column,
  section,
  children,
}: {
  column: Column;
  section: Section;
  children: React.ReactNode;
}) => {
  // Enables column reordering through drag and drop
  // Maintains column position state and handles drag events
  const { attributes, listeners, setNodeRef, transform } = useDraggable({
    id: `column-${column.id}`,
    data: {
      type: "column",
      sectionId: section.id,
      columnId: column.id,
    },
  });

  const style = transform
    ? { transform: `translate3d(${transform.x}px, ${transform.y}px, 0)` }
    : undefined;

  return (
    <div
      ref={setNodeRef}
      style={style}
      {...listeners}
      {...attributes}
      className="flex-none w-80"
```

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

const DroppableColumn = ({
  column,
  section,
  children,
}: {
  column: Column;
  section: Section;
  children: React.ReactNode;
}) => {
  // Defines drop zone for notes and other columns
  // Visual feedback during drag operations
  const { setNodeRef, isOver } = useDroppable({
    id: `droppable-${column.id}`,
    data: { type: "column", sectionId: section.id, columnId: column.id },
  });

  return (
    <div
      ref={setNodeRef}
      className={`rounded-lg p-4 ${isOver ? "bg-gray-100" : "bg-gray-50"}`}
    >
      {children}
    </div>
  );
};

// Main notes (tasks)
const DraggableNote = ({
  note,
  children,
}: {
  note: Note;
  children: React.ReactNode;
}) => {
```

```
  // Enables notes to be dragged between columns
  // Maintains position state during drag operations
  const { attributes, listeners, setNodeRef, transform } = useDraggable({
    id: note.id,
    data: { type: "note", noteId: note.id },
  });

  const style = transform
    ? { transform: `translate3d(${transform.x}px, ${transform.y}px, 0)` }
    : undefined;

  return (
    <div
      ref={setNodeRef}
      style={style}
      {...listeners}
      {...attributes}
      className="bg-[#fff9e6] rounded-lg p-4 cursor-move group"
    >
      {children}
    </div>
  );
};

// Subtasks – similar to DraggableNote but with extra drag data.
const DraggableSubtask = ({
  subtask,
  sectionId,
  columnId,
  parentNoteId,
  children,
}: {
  subtask: Note;
  sectionId: string;
  columnId: string;
  parentNoteId: string;
  children: React.ReactNode;
}) => {
  // Enables subtasks to be reordered within a parent note
  // Can also be moved between different parent notes
```

```
  const { attributes, listeners, setNodeRef, transform } = useDraggable({
    id: `subtask-${subtask.id}`,
    data: {
      type: "subtask",
      sectionId,
      columnId,
      parentNoteId,
      subtaskId: subtask.id,
    },
  });

  const style = transform
    ? { transform: `translate3d(${transform.x}px, ${transform.y}px, 0)` }
    : undefined;

  return (
    <div
      ref={setNodeRef}
      style={style}
      {...listeners}
      {...attributes}
      className="bg-[#e6f7ff] rounded-lg p-2 cursor-move group"
    >
      {children}
    </div>
  );
};

const DroppableSubtaskContainer = ({
  parentNoteId,
  columnId,
  children,
}: {
  parentNoteId: string;
  columnId: string;
  children: React.ReactNode;
}) => {
  // Defines drop zone for subtasks within a parent note
  // Handles subtask reordering and moving between parents
  const { setNodeRef, isOver } = useDroppable({
```

```
    id: `droppable-subtasks-${parentNoteId}`,
    data: { type: "subtask-container", parentNoteId, columnId },
  });

  return (
    <div
      ref={setNodeRef}
      className={`ml-4 mt-2 p-2 rounded-lg ${isOver ? "bg-blue-50" : ""}`}
    >
      {children}
    </div>
  );
};

// ——— THE MAIN COMPONENT ———

// Main ToDo list component that orchestrates all functionality
const ToDoList = ({
  hideBackButton = false,
  isDashboard = false,
}: ToDoListProps) => {
  // User authentication state
  const { user } = useUser();
  const { toast } = useToast();
  const router = useRouter();
  const [sections, setSections] = useState<Section[]>([]);
  const [activeDragItem, setActiveDragItem] = useState<{
    type: "note" | "subtask" | "column";
    item: Note | Column;
    extraData?: any;
  } | null>(null);

  const sensors = useSensors(
    useSensor(PointerSensor, { activationConstraint: { distance: 8 } })
  );

  // ——— FETCH / SYNC DATA ———

  // Synchronizes local state with Firestore database
  useEffect(() => {
```

```
if (!user) return;
const fetchTodoListData = async () => {
  try {
    const userDocRef = doc(db, "users", user.id);
    const todoListRef = doc(collection(userDocRef, "todoList"), "data");
    const todoListDoc = await getDoc(todoListRef);
    if (todoListDoc.exists()) {
      const data = todoListDoc.data();
      const processSections = (sections: Section[]) =>
        sections.map((section) => ({
          ...section,
          columns: section.columns.map((column) => ({
            ...column,
            notes: column.notes.map((note) => ({
              ...note,
              checked:
                note.hasOwnProperty("checked") &&
                typeof note.checked === "boolean"
                  ? note.checked
                  : false,
              subtasks: note.subtasks
                ? note.subtasks.map((subtask: Note) => ({
                    ...subtask,
                    checked:
                      subtask.hasOwnProperty("checked") &&
                      typeof subtask.checked === "boolean"
                        ? subtask.checked
                        : false,
                  }))
                : [],
            })),
          })),
        }));
      setSections(processSections(data.sections || []));
    } else {
      await setDoc(todoListRef, {
        sections: [],
        lastUpdated: new Date(),
      });
    }
```

```
    } catch (error) {
      console.error("Error fetching todoList data:", error);
    }
  };
  fetchTodoListData();
}, [user]);

// Debounced save of changes to Firestore
useEffect(() => {
  if (!user) return;
  const timeoutId = setTimeout(async () => {
    try {
      const userDocRef = doc(db, "users", user.id);
      const todoListRef = doc(collection(userDocRef, "todoList"), "data");
      await setDoc(todoListRef, { sections, lastUpdated: new Date() });
    } catch (error) {
      console.error("Error updating data:", error);
    }
  }, 1000);
  return () => clearTimeout(timeoutId);
}, [sections, user]);

// ——— SECTION / COLUMN / NOTE / SUBTASK FUNCTIONS ———

const archiveSection = (sectionId: string) => {
  setSections(sections.filter((s) => s.id !== sectionId));
};

// Core state modification functions
const addSection = () => {
  // Creates new section with default column
  const newSection: Section = {
    id: Date.now().toString(),
    title: "New Section",
    isEditing: true,
    columns: [
      {
        id: `col-${Date.now()}`,
        title: "New Column",
        isEditing: false,
```

```
        notes: [],
      },
    ],
  };
  setSections([...sections, newSection]);
};

const addColumn = (sectionId: string) => {
  setSections(
    sections.map((section) =>
      section.id === sectionId
        ? {
            ...section,
            columns: [
              ...section.columns,
              {
                id: `col-${Date.now()}`,
                title: "New Column",
                isEditing: true,
                notes: [],
              },
            ],
          }
        : section
    )
  );
};

const addNote = (sectionId: string, columnId: string) => {
  setSections(
    sections.map((section) =>
      section.id === sectionId
        ? {
            ...section,
            columns: section.columns.map((column) =>
              column.id === columnId
                ? {
                    ...column,
                    notes: [
                      ...column.notes,
```

```
                {
                  id: `note-${Date.now()}`,
                  content: "New note",
                  checked: false,
                  subtasks: [],
                  isEditing: true,
                },
              ],
            }
          : column
      ),
    }
  : section
)
);
};

const addSubtask = (
  sectionId: string,
  columnId: string,
  parentNoteId: string
) => {
  setSections((prevSections) =>
    prevSections.map((section) =>
      section.id !== sectionId
        ? section
        : {
            ...section,
            columns: section.columns.map((column) =>
              column.id !== columnId
                ? column
                : {
                    ...column,
                    notes: column.notes.map((note) =>
                      note.id !== parentNoteId
                        ? note
                        : {
                            ...note,
                            subtasks: note.subtasks
                              ? [
```

```
                                ...note.subtasks,
                                {
                                  id: `subtask-${Date.now()}`,
                                  content: "New Subtask",
                                  checked: false,
                                  subtasks: [],
                                },
                              ]
                            : [
                                {
                                  id: `subtask-${Date.now()}`,
                                  content: "New Subtask",
                                  checked: false,
                                  subtasks: [],
                                },
                              ],
                        }
                      ),
                    }
                  ),
                }
              )
            );
          };

          const moveNote = (
            fromSectionId: string,
            fromColumnId: string,
            toSectionId: string,
            toColumnId: string,
            noteId: string
          ) => {
            // Handles moving notes between columns and sections
            // Maintains note data integrity during moves
            const noteToMove = sections
              .find((s) => s.id === fromSectionId)
              ?.columns.find((c) => c.id === fromColumnId)
              ?.notes.find((n) => n.id === noteId);
            if (!noteToMove) return;
            setSections(
```

```
    sections.map((section) => {
      if (section.id === fromSectionId) {
        return {
          ...section,
          columns: section.columns.map((column) => {
            if (column.id === fromColumnId) {
              return {
                ...column,
                notes: column.notes.filter((note) => note.id !== noteId),
              };
            }
            if (column.id === toColumnId && fromSectionId === toSectionId) {
              return { ...column, notes: [...column.notes, noteToMove] };
            }
            return column;
          }),
        };
      }
      if (section.id === toSectionId) {
        return {
          ...section,
          columns: section.columns.map((column) => {
            if (column.id === toColumnId) {
              return { ...column, notes: [...column.notes, noteToMove] };
            }
            return column;
          }),
        };
      }
      return section;
    })
  );
};

const moveColumn = (
  sectionId: string,
  fromIndex: number,
  toIndex: number
) => {
  setSections(
```

```
    sections.map((section) => {
      if (section.id === sectionId) {
        const newColumns = [...section.columns];
        const [movedColumn] = newColumns.splice(fromIndex, 1);
        newColumns.splice(toIndex, 0, movedColumn);
        return { ...section, columns: newColumns };
      }
      return section;
    })
  );
};

// Move a subtask within the same parent note.
const moveSubtaskWithinParent = (
  sectionId: string,
  columnId: string,
  parentNoteId: string,
  fromIndex: number,
  toIndex: number
) => {
  setSections((prevSections) =>
    prevSections.map((section) => {
      if (section.id !== sectionId) return section;
      return {
        ...section,
        columns: section.columns.map((column) => {
          if (column.id !== columnId) return column;
          return {
            ...column,
            notes: column.notes.map((note) => {
              if (note.id !== parentNoteId || !note.subtasks) return note;
              const newSubtasks = [...note.subtasks];
              const [moved] = newSubtasks.splice(fromIndex, 1);
              newSubtasks.splice(toIndex, 0, moved);
              return { ...note, subtasks: newSubtasks };
            }),
          };
        }),
      };
    })
```

```
    );
  };

  // Move a subtask from one parent note to another (and possibly into a different
column).
  const moveSubtaskToAnotherParent = (
    sectionId: string,
    fromColumnId: string,
    fromParentNoteId: string,
    subtaskId: string,
    toParentNoteId: string,
    toColumnId: string
  ) => {
    // Handles moving subtasks between different parent notes
    // Updates both source and target parent notes
    let movedSubtask: Note | null = null;
    setSections((prevSections) =>
      prevSections.map((section) => {
        if (section.id !== sectionId) return section;
        return {
          ...section,
          columns: section.columns.map((column) => {
            // Remove the subtask from its original parent in the source column.
            if (column.id === fromColumnId) {
              return {
                ...column,
                notes: column.notes.map((note) => {
                  if (note.id === fromParentNoteId && note.subtasks) {
                    const newSubtasks = note.subtasks.filter((subtask) => {
                      if (subtask.id === subtaskId) {
                        movedSubtask = subtask;
                        return false;
                      }
                      return true;
                    });
                    return { ...note, subtasks: newSubtasks };
                  }
                  return note;
                }),
              };
```

```
        }
        // In the target column, add the subtask to the target parent.
        if (column.id === toColumnId) {
          return {
            ...column,
            notes: column.notes.map((note) => {
              if (note.id === toParentNoteId) {
                return {
                  ...note,
                  subtasks: note.subtasks
                    ? [...note.subtasks, movedSubtask!]
                    : [movedSubtask!],
                };
              }
              return note;
            }),
          };
        }
        return column;
      }),
    };
  })
 );
};

const deleteSection = (sectionId: string) => {
  const section = sections.find((s) => s.id === sectionId);
  setSections(sections.filter((section) => section.id !== sectionId));
  toast({
    title: "Section deleted",
    description: `${section?.title || "Section"} has been deleted.`,
    action: (
      <ToastAction
        altText="Undo"
        onClick={() => {
          if (section) setSections((prev) => [...prev, section]);
        }}
      >
        Undo
      </ToastAction>
```

```
    ),
  });
};

const deleteColumn = (sectionId: string, columnId: string) => {
  const section = sections.find((s) => s.id === sectionId);
  const column = section?.columns.find((c) => c.id === columnId);
  setSections(
    sections.map((section) =>
      section.id === sectionId
        ? {
            ...section,
            columns: section.columns.filter((col) => col.id !== columnId),
          }
        : section
    )
  );
  toast({
    title: "Column deleted",
    description: `${column?.title || "Column"} has been deleted.`,
    action: (
      <ToastAction
        altText="Undo"
        onClick={() => {
          if (column) {
            setSections((prev) =>
              prev.map((s) =>
                s.id === sectionId
                  ? { ...s, columns: [...s.columns, column] }
                  : s
              )
            );
          }
        }}
      >
        Undo
      </ToastAction>
    ),
  });
};
```

```
const deleteNote = (sectionId: string, columnId: string, noteId: string) => {
  const section = sections.find((s) => s.id === sectionId);
  const note = section?.columns
    .find((c) => c.id === columnId)
    ?.notes.find((n) => n.id === noteId);
  setSections(
    sections.map((section) =>
      section.id === sectionId
        ? {
            ...section,
            columns: section.columns.map((col) =>
              col.id === columnId
                ? {
                    ...col,
                    notes: col.notes.filter((note) => note.id !== noteId),
                  }
                : col
            ),
          }
        : section
    )
  );
  toast({
    title: "Note deleted",
    description: `${note?.content || "Note"} has been deleted.`,
    action: (
      <ToastAction
        altText="Undo"
        onClick={() => {
          if (note) {
            setSections((prev) =>
              prev.map((s) =>
                s.id === sectionId
                  ? {
                      ...s,
                      columns: s.columns.map((c) =>
                        c.id === columnId
                          ? { ...c, notes: [...c.notes, note] }
                          : c
```

```
                    ),
                  }
                : s
            )
          );
        }
      }}
    >
      Undo
    </ToastAction>
  ),
 });
};

const deleteSubtask = (
  sectionId: string,
  columnId: string,
  parentNoteId: string,
  subtaskId: string
) => {
  let deletedSubtask: Note | undefined;
  setSections((prevSections) =>
    prevSections.map((section) => {
      if (section.id !== sectionId) return section;
      return {
        ...section,
        columns: section.columns.map((column) => {
          if (column.id !== columnId) return column;
          return {
            ...column,
            notes: column.notes.map((note) => {
              if (note.id !== parentNoteId) return note;
              const newSubtasks = note.subtasks?.filter((subtask) => {
                if (subtask.id === subtaskId) {
                  deletedSubtask = subtask;
                  return false;
                }
                return true;
              });
              return {
```

```
            ...note,
            subtasks: newSubtasks,
          };
        }),
      };
    }),
  };
 })
);

toast({
  title: "Subtask deleted",
  description: `${deletedSubtask?.content || "Subtask"} has been deleted.`,
  action: (
    <ToastAction
      altText="Undo"
      onClick={() => {
        if (deletedSubtask) {
          setSections((prev) =>
            prev.map((s) =>
              s.id === sectionId
                ? {
                    ...s,
                    columns: s.columns.map((c) =>
                      c.id === columnId
                        ? {
                            ...c,
                            notes: c.notes.map((n) =>
                              n.id === parentNoteId
                                ? {
                                    ...n,
                                    subtasks: [
                                      ...(n.subtasks || []),
                                      deletedSubtask!,
                                    ],
                                  }
                                : n
                            ),
                          }
                        : c
```

```
                    ),
                  }
                : s
            )
          );
        }
      }}
    >
      Undo
    </ToastAction>
  ),
 });
};

const toggleNoteChecked = (
  sectionId: string,
  columnId: string,
  noteId: string
) => {
  setSections(
    sections.map((section) =>
      section.id === sectionId
        ? {
            ...section,
            columns: section.columns.map((col) =>
              col.id === columnId
                ? {
                    ...col,
                    notes: col.notes.map((note) =>
                      note.id === noteId
                        ? { ...note, checked: !note.checked }
                        : note
                    ),
                  }
                : col
            ),
          }
        : section
    )
  );
```

```
};

const toggleSubtaskChecked = (
  sectionId: string,
  columnId: string,
  parentNoteId: string,
  subtaskId: string
) => {
  setSections((prevSections) =>
    prevSections.map((section) => {
      if (section.id !== sectionId) return section;
      return {
        ...section,
        columns: section.columns.map((col) => {
          if (col.id !== columnId) return col;
          return {
            ...col,
            notes: col.notes.map((note) => {
              if (note.id !== parentNoteId) return note;
              return {
                ...note,
                subtasks: note.subtasks?.map((subtask) =>
                  subtask.id === subtaskId
                    ? { ...subtask, checked: !subtask.checked }
                    : subtask
                ),
              };
            }),
          };
        }),
      };
    })
  );
};

const updateSubtaskContent = (
  sectionId: string,
  columnId: string,
  parentNoteId: string,
  subtaskId: string,
```

```
  newContent: string
) => {
  setSections((prevSections) =>
    prevSections.map((section) => {
      if (section.id !== sectionId) return section;
      return {
        ...section,
        columns: section.columns.map((col) => {
          if (col.id !== columnId) return col;
          return {
            ...col,
            notes: col.notes.map((note) => {
              if (note.id !== parentNoteId) return note;
              return {
                ...note,
                subtasks: note.subtasks?.map((subtask) =>
                  subtask.id === subtaskId
                    ? { ...subtask, content: newContent }
                    : subtask
                ),
              };
            }),
          };
        }),
      };
    })
  );
};

// DRAG & DROP HANDLERS

// Drag and drop event handlers
const handleDragStart = (event: DragStartEvent) => {
  // Sets up drag overlay and tracks dragged item type
  const { active } = event;
  const activeData = active.data.current as DragData;
  if (activeData.type === "note") {
    const noteId = active.id as string;
    const draggedNote = sections
      .flatMap((section) =>
```

```
        section.columns.flatMap((col) =>
          col.notes.find((note) => note.id === noteId)
        )
      )
      .find(Boolean);
    if (draggedNote) setActiveDragItem({ type: "note", item: draggedNote });
  } else if (activeData.type === "subtask") {
    const subtaskId = activeData.subtaskId;
    let draggedSubtask: Note | undefined;
    sections.forEach((section) => {
      section.columns.forEach((col) => {
        col.notes.forEach((note) => {
          if (note.subtasks) {
            const found = note.subtasks.find((sub) => sub.id === subtaskId);
            if (found) draggedSubtask = found;
          }
        });
      });
    });
    if (draggedSubtask)
      setActiveDragItem({
        type: "subtask",
        item: draggedSubtask,
        extraData: { parentNoteId: activeData.parentNoteId },
      });
  }
};

const handleDragEnd = (event: DragEndEvent) => {
  // Handles different types of drag operations:
  // - Column reordering
  // - Note moving between columns
  // - Subtask reordering and reparenting
  const { active, over } = event;
  setActiveDragItem(null);
  if (!over) return;
  const activeData = active.data.current as DragData;
  const overData = over.data.current as any;

  // ── COLUMN DRAGGING ──
```

```
if (activeData.type === "column" && overData.type === "column") {
  const sectionId = activeData.sectionId;
  const section = sections.find((s) => s.id === sectionId);
  if (!section) return;
  const fromIndex = section.columns.findIndex(
    (c) => c.id === activeData.columnId
  );
  const toIndex = section.columns.findIndex(
    (c) => c.id === overData.columnId
  );
  if (fromIndex !== toIndex) moveColumn(sectionId, fromIndex, toIndex);
  return;
}

// ── NOTE DRAGGING ──
if (activeData.type === "note") {
  if (!overData || !overData.columnId) return;
  let fromSectionId = "",
    fromColumnId = "";
  sections.some((section) => {
    return section.columns.some((col) => {
      const found = col.notes.some((note) => note.id === active.id);
      if (found) {
        fromSectionId = section.id;
        fromColumnId = col.id;
        return true;
      }
      return false;
    });
  });
  const toSectionId = overData.sectionId;
  const toColumnId = overData.columnId;
  if (fromSectionId && fromColumnId && toSectionId && toColumnId) {
    moveNote(
      fromSectionId,
      fromColumnId,
      toSectionId,
      toColumnId,
      active.id as string
    );
```

```
  }
}

// ── SUBTASK DRAGGING ──
if (activeData.type === "subtask") {
  if (!overData || !activeData.columnId) return;
  if (overData.type === "subtask-container") {
    // Retrieve the target parent note and target column from the droppable.
    const toParentNoteId = overData.parentNoteId;
    const toColumnId = overData.columnId;
    const { sectionId, columnId, parentNoteId, subtaskId } = activeData;
    if (
      !sectionId ||
      !columnId ||
      !parentNoteId ||
      !subtaskId ||
      !toColumnId
    )
      return;

    if (parentNoteId === toParentNoteId && columnId === toColumnId) {
      // Reordering within the same parent note.
      let fromIndex = -1,
        toIndex = -1;
      sections.forEach((section) => {
        if (section.id === sectionId) {
          section.columns.forEach((col) => {
            if (col.id === columnId) {
              col.notes.forEach((note) => {
                if (note.id === parentNoteId && note.subtasks) {
                  fromIndex = note.subtasks.findIndex(
                    (s) => s.id === subtaskId
                  );
                  if (over.id.toString().startsWith("subtask-")) {
                    const hoveredSubtaskId = over.id
                      .toString()
                      .replace("subtask-", "");
                    toIndex = note.subtasks.findIndex(
                      (s) => s.id === hoveredSubtaskId
                    );
```

```
                } else {
                  toIndex = note.subtasks.length;
                }
              }
            });
          }
        });
      }
    });
    if (fromIndex !== -1 && toIndex !== -1 && fromIndex !== toIndex) {
      moveSubtaskWithinParent(
        sectionId,
        columnId,
        parentNoteId,
        fromIndex,
        toIndex
      );
    }
  } else {
    moveSubtaskToAnotherParent(
      sectionId,
      columnId,
      parentNoteId,
      subtaskId,
      toParentNoteId,
      toColumnId
    );
  }
 }
};

// RENDERING

return (
  <div className="flex flex-col h-full">
    {" "}
    {/* Changed from h-screen to h-full */}
    <header className="flex items-center justify-between p-4 border-b">
      {!hideBackButton && <BackButton />}
```

```
  <Sheet>
    <SheetTrigger asChild>
      <Button variant="outline" size="icon">
        <Archive className="h-4 w-4" />
      </Button>
    </SheetTrigger>
    <SheetContent>
      <SheetHeader>
        <SheetTitle>Archived Sections</SheetTitle>
      </SheetHeader>
      {/* (Optional archived sections UI) */}
    </SheetContent>
  </Sheet>
</header>
<DndContext
  sensors={sensors}
  collisionDetection={closestCorners}
  onDragStart={handleDragStart}
  onDragEnd={handleDragEnd}
>
  <div
    className={`max-w-none bg-white flex-1 relative ${
      isDashboard ? "pb-24" : "pb-20"
    }`}
  >
    {sections.map((section) => (
      <div
        key={section.id}
        className="border-b border-gray-100 mb-8 overflow-x-auto"
      >
        <div className="flex justify-between items-center px-4 py-2 bg-white group">
          <h2
            className="text-lg font-medium outline-none focus:outline-none"
            contentEditable
            suppressContentEditableWarning
            onBlur={(e) => {
              setSections(
                sections.map((s) =>
                  s.id === section.id
                    ? {
```

```
                    ...s,
                    title: e.currentTarget.textContent || s.title,
                  }
                : s
            )
          );
        }}
      >
        {section.title}
      </h2>
      <div className="flex items-center space-x-2 opacity-0
group-hover:opacity-100 transition-opacity">
        <Button
          variant="ghost"
          size="sm"
          onClick={() => archiveSection(section.id)}
        >
          Archive
        </Button>
        <Button
          variant="ghost"
          size="sm"
          onClick={() => deleteSection(section.id)}
        >
          <X className="h-4 w-4" />
        </Button>
      </div>
    </div>
    <div className="px-4 py-2">
      <div className="flex space-x-4 min-h-[200px] pb-4">
        {section.columns.map((column) => (
          <DraggableColumn
            key={column.id}
            column={column}
            section={section}
          >
            <DroppableColumn
              key={column.id}
              column={column}
              section={section}
```

```
                >
                  <div className="flex justify-between items-center mb-4 group">
                    <h3
                      className="font-medium text-gray-700 outline-none
focus:outline-none"
                      contentEditable
                      suppressContentEditableWarning
                      onBlur={(e) => {
                        setSections(
                          sections.map((s) =>
                            s.id === section.id
                              ? {
                                  ...s,
                                  columns: s.columns.map((c) =>
                                    c.id === column.id
                                      ? {
                                          ...c,
                                          title:
                                            e.currentTarget.textContent ||
                                            c.title,
                                        }
                                      : c
                                  ),
                                }
                              : s
                          )
                        );
                      }}
                    >
                      {column.title}
                    </h3>
                    <div className="flex items-center space-x-1 opacity-0
group-hover:opacity-100 transition-opacity">
                      <Button
                        variant="ghost"
                        size="sm"
                        onClick={() =>
                          deleteColumn(section.id, column.id)
                        }
                      >
```

```
                    <X className="h-4 w-4" />
                  </Button>
                </div>
              </div>
              <div className="space-y-4">
                {column.notes.map((note) => (
                  <DraggableNote key={note.id} note={note}>
                    <div className="flex flex-col">
                      <div className="flex justify-between items-start">
                        <div className="flex items-center gap-2 flex-1">
                          <input
                            type="checkbox"
                            checked={note.checked}
                            onChange={() =>
                              toggleNoteChecked(
                                section.id,
                                column.id,
                                note.id
                              )
                            }
                            className="mt-1 h-4 w-4"
                          />
                          <p
                            contentEditable
                            suppressContentEditableWarning
                            className={`text-gray-800 outline-none focus:outline-none
flex-1 ${
                              note.checked
                                ? "line-through opacity-75"
                                : ""
                            }`}
                            onBlur={(e) => {
                              setSections(
                                sections.map((s) =>
                                  s.id === section.id
                                    ? {
                                        ...s,
                                        columns: s.columns.map((c) =>
                                          c.id === column.id
                                            ? {
```

```
                                    ...c,
                                    notes: c.notes.map(
                                      (n) =>
                                        n.id === note.id
                                          ? {
                                              ...n,
                                              content:
                                                e
                                                  .currentTarget
                                                  .textContent ||
                                                "",
                                            }
                                          : n
                                    ),
                                  }
                                : c
                            ),
                          }
                        : s
                    )
                  );
                }}
              >
                {note.content}
              </p>
            </div>
            <div className="flex items-center space-x-1 opacity-0
group-hover:opacity-100 transition-opacity">
              <Button
                variant="ghost"
                size="sm"
                onClick={() =>
                  deleteNote(
                    section.id,
                    column.id,
                    note.id
                  )
                }
              >
                <X className="h-4 w-4" />
```

```
                    </Button>
                  </div>
                </div>
                {/* ——— SUBTASKS RENDERING ——— */}
                <DroppableSubtaskContainer
                  parentNoteId={note.id}
                  columnId={column.id}
                >
                  {note.subtasks &&
                    note.subtasks.map((subtask) => (
                      <DraggableSubtask
                        key={subtask.id}
                        subtask={subtask}
                        sectionId={section.id}
                        columnId={column.id}
                        parentNoteId={note.id}
                      >
                        <div className="flex justify-between items-start">
                          <div className="flex items-center gap-2 flex-1">
                            <input
                              type="checkbox"
                              checked={subtask.checked}
                              onChange={() =>
                                toggleSubtaskChecked(
                                  section.id,
                                  column.id,
                                  note.id,
                                  subtask.id
                                )
                              }
                              className="mt-1 h-4 w-4"
                            />
                            <p
                              contentEditable
                              suppressContentEditableWarning
                              className={`text-gray-800 outline-none
focus:outline-none flex-1 ${
                                subtask.checked
                                  ? "line-through opacity-75"
                                  : ""
```

```
            }`}
            onBlur={(e) => {
              updateSubtaskContent(
                section.id,
                column.id,
                note.id,
                subtask.id,
                e.currentTarget.textContent ||
                  ""
              );
            }}
          >
            {subtask.content}
          </p>
        </div>
        <Button
          variant="ghost"
          size="sm"
          onClick={() =>
            deleteSubtask(
              section.id,
              column.id,
              note.id,
              subtask.id
            )
          }
        >
          <X className="h-4 w-4" />
        </Button>
      </div>
    </DraggableSubtask>
  ))}
<Button
  variant="ghost"
  size="sm"
  className="mt-2"
  onClick={() =>
    addSubtask(section.id, column.id, note.id)
  }
>
```

```
                    <Plus className="h-4 w-4 mr-1" />
                    Add Subtask
                  </Button>
                </DroppableSubtaskContainer>
              </div>
            </DraggableNote>
          ))}
        </div>
        <Button
          variant="ghost"
          size="sm"
          className="w-full mt-4"
          onClick={() => addNote(section.id, column.id)}
        >
          <Plus className="h-4 w-4 mr-2" />
          Add Task
        </Button>
      </DroppableColumn>
    </DraggableColumn>
  ))}
  <Button
    variant="outline"
    className="flex-none w-80 h-12"
    onClick={() => addColumn(section.id)}
  >
    <Plus className="h-4 w-4 mr-2" />
    Add Column
  </Button>
 </div>
 </div>
</div>
))}
<DragOverlay>
 {activeDragItem ? (
  <div className="bg-[#fff9e6] rounded-lg p-4 w-72 shadow-lg">
   <div className="flex justify-between items-start">
    <p className="text-gray-800">
     {"content" in activeDragItem.item
      ? activeDragItem.item.content
      : ""}
```

```
              </p>
            </div>
          </div>
        ) : null}
      </DragOverlay>

      <Button
        variant="outline"
        size="lg"
        className={`${
          isDashboard
            ? "absolute bottom-4 left-1/2 transform -translate-x-1/2"
            : "fixed bottom-8 left-1/2 transform -translate-x-1/2"
        }`}
        onClick={addSection}
      >
        <Plus className="h-4 w-4 mr-2" />
        Add Section
      </Button>
    </div>
  </DndContext>
 </div>
 );
};

export default ToDoList;


==========
Directory: src\app\ToDoList
==========

File: src\app\ToDoList\page.tsx
----------

"use client";

import React from "react";
import ToDoList from "@/components/ToDoList";
```

```
const ToDoListPage: React.FC = () => {
  return (
    <div className="flex flex-col h-screen">
      <main className="flex-1 overflow-auto">
        <ToDoList />
      </main>
    </div>
  );
};

export default ToDoListPage;
```

Continuous Information Space (Notebook) Module

```
==========
Directory: src\lib
==========

File: src\lib\continuous-info-space-doc-man-actions.ts
----------

"use server";

import { z } from "zod";
import { db } from "@/lib/firebase";
import { collection, doc, getDocs, getDoc, setDoc, deleteDoc, serverTimestamp } from
"firebase/firestore";
import { auth } from "@clerk/nextjs/server";

/**
 * Zod schema to validate the structure of a Notebook object.
 * Defines the expected data types and constraints for each field in a notebook
document.
 * This schema is used to ensure data integrity when reading and writing notebook data.
 */
const NotebookSchema = z.object({
  id: z.string(),
  title: z.string().min(1, "Title is required"),
  description: z.string().optional(),
  createdAt: z.date(),
  updatedAt: z.date(),
  userId: z.string(),
  sections: z.array(z.any()).optional()
});

export type notebook = z.infer<typeof NotebookSchema>;

/**
 * Helper function to get a Firestore CollectionReference for notebooks.
 * It encapsulates the logic to retrieve the user ID and then construct the path to the
'notebooks' collection
 * under the user's document in Firestore.
```

```
 * @returns {Promise<CollectionReference>} - Firestore CollectionReference for notebooks.
 * @throws {Error} - If the user is not authenticated.
 */
async function getNotebooksCollectionRef() {
  const { userId } = await auth();
  if (!userId) throw new Error("User not authenticated");
  return collection(db, "users", userId, "notebooks");
}

/**
 * Reads all notebooks for the currently authenticated user from Firestore.
 * Fetches all documents from the 'notebooks' collection for the user and parses them using the NotebookSchema.
 * It also includes error handling and filtering out invalid notebooks.
 * @returns {Promise<notebook[]>} - An array of notebook objects, or an empty array if there are errors or no notebooks.
 */
export async function readNotebooks(): Promise<notebook[]> {
  try {
    const notebooksCol = await getNotebooksCollectionRef();
    const snapshot = await getDocs(notebooksCol);

    return snapshot.docs.map(doc => {
      const data = doc.data();
      // Add null checks and default values to handle potentially missing fields in Firestore documents.
      if (!data.title || !data.createdAt || !data.userId) {
        console.warn(`Notebook ${doc.id} is missing required fields:`, data);
        return null;
      }

      return NotebookSchema.parse({
        id: doc.id,
        title: data.title,
        description: data.description || '',  // Provide default empty string if description is missing
        createdAt: data.createdAt?.toDate() || new Date(),  // Provide current date as fallback if createdAt is missing or invalid
```

```
        updatedAt: data.updatedAt?.toDate() || new Date(),  // Provide current date as fallback if updatedAt is missing or invalid
        userId: data.userId,
        sections: data.sections || []
      });
    }).filter(notebook => notebook !== null) as notebook[]; // Filter out invalid notebooks that failed schema parsing
  } catch (error) {
    console.error("Error reading notebooks:", error);
    return [];
  }
}

/**
 * Retrieves a single notebook document from Firestore by its ID.
 * @param {string} id - The ID of the notebook to retrieve.
 * @returns {Promise<notebook | null>} - The notebook object if found and valid, otherwise null.
 */
export async function getNotebook(id: string): Promise<notebook | null> {
  try {
    const notebooksCol = await getNotebooksCollectionRef();
    const notebookDoc = await getDoc(doc(notebooksCol, id));

    if (!notebookDoc.exists()) return null;

    const data = notebookDoc.data();
    return NotebookSchema.parse({
      id: notebookDoc.id,
      title: data.title,
      description: data.description,
      createdAt: data.createdAt?.toDate(),
      updatedAt: data.updatedAt?.toDate(),
      userId: data.userId,
      sections: data.sections || []
    });
  } catch (error) {
    console.error("Error getting notebook:", error);
    return null;
  }
```

```
}

/**
 * Adds a new notebook to Firestore.
 * It automatically sets the userId, createdAt, and updatedAt fields, and initializes sections as an empty array.
 * @param {Omit<notebook, "id" | "createdAt" | "updatedAt" | "userId">} notebook - Notebook data excluding ID and timestamps.
 * @returns {Promise<notebook>} - The newly created notebook object, including the generated ID and timestamps.
 * @throws {Error} - If there's an error during the process of adding the notebook.
 */
export async function addNotebook(notebook: Omit<notebook, "id" | "createdAt" | "updatedAt" | "userId">) {
  try {
    const notebooksCol = await getNotebooksCollectionRef();
    const { userId } = await auth();

    const newNotebookRef = doc(notebooksCol); // Create a new document reference with an auto-generated ID
    const now = serverTimestamp(); // Get the Firestore server timestamp

    await setDoc(newNotebookRef, {
      ...notebook,
      userId,
      sections: [], // Initialize sections as an empty array for new notebooks
      createdAt: now,
      updatedAt: now
    });

    // Return the newly created notebook object with generated ID and timestamps, converting serverTimestamp to Date
    return {
      id: newNotebookRef.id,
      ...notebook,
      sections: [],
      createdAt: new Date(),
      updatedAt: new Date(),
      userId
    };
```

```
  } catch (error) {
    console.error("Error adding notebook:", error);
    throw error; // Re-throw the error to be handled by the caller
  }
}

/**
 * Updates the 'sections' field of an existing notebook in Firestore.
 * Allows for partial updates, specifically for the sections array, and updates the
'updatedAt' timestamp.
 * @param {string} id - The ID of the notebook to update.
 * @param {any[]} sections - The new sections array to be set for the notebook.
 * @throws {Error} - If there's an error during the update process.
 */
export async function updateNotebook(id: string, sections: any[]) {
  try {
    const notebooksCol = await getNotebooksCollectionRef();
    await setDoc(doc(notebooksCol, id), {
      sections,
      updatedAt: serverTimestamp() // Update the 'updatedAt' timestamp to the server's
current time
    }, { merge: true }); // Use merge: true to avoid overwriting other fields in the notebook
document
  } catch (error) {
    console.error("Error updating notebook:", error);
    throw error; // Re-throw the error to be handled by the caller
  }
}

/**
 * Deletes a notebook document from Firestore.
 * @param {string} id - The ID of the notebook to delete.
 * @throws {Error} - If there's an error during deletion.
 */
export async function deleteNotebook(id: string) {
  try {
    const notebooksCol = await getNotebooksCollectionRef();
    await deleteDoc(doc(notebooksCol, id));
  } catch (error) {
    console.error("Error deleting notebook:", error);
```

```
    throw error; // Re-throw the error to be handled by the caller
  }
}

==========
Directory: src\app\ContinuousInfoSpaceDocMan
==========

File: src\app\ContinuousInfoSpaceDocMan\page.tsx
----------

"use client";

import React from "react";
import DocumentJar from
"@/components/ContinuousInfoSpace/ContinuousInfoSpaceDocMan";
import { BackButton } from "@/components/ui/custom-ui/back-button";

const DocumentJarPage: React.FC = () => {
  return (
    <div className="flex flex-col h-screen">
      <header className="flex items-center justify-between p-4 border-b">
        <BackButton />
      </header>
      <main className="flex-1 overflow-auto">
        <DocumentJar />
      </main>
    </div>
  );
};

export default DocumentJarPage;


==========
Directory: src\components\ContinuousInfoSpace
==========

File: src\components\ContinuousInfoSpace\ContinuousInfoSpaceDocMan.tsx
----------
```

```
"use client";

import React, { useState, useEffect } from "react";
import { Plus, Trash2, Notebook, Eye } from "lucide-react";
import {
  DndContext,
  closestCenter,
  KeyboardSensor,
  PointerSensor,
  useSensor,
  useSensors,
} from "@dnd-kit/core";
import { SortableContext, rectSortingStrategy } from "@dnd-kit/sortable";
import { Button } from "@/components/ui/button";
import {
  Dialog,
  DialogContent,
  DialogHeader,
  DialogTitle,
} from "@/components/ui/dialog";
import { useRouter } from "next/navigation";
import {
  readNotebooks,
  deleteNotebook,
  addNotebook,
  getNotebook,
  notebook,
} from "@/lib/continuous-info-space-doc-man-actions";

/**
 * Component for displaying a single Notebook as a card.
 * It handles rendering notebook details and provides interactive buttons for actions like
previewing, opening, and deleting the notebook.
 */
function NotebookCard({
  notebook,
  onDelete,
  onOpen,
  onPreview,
```

```
}: {
  notebook: notebook;
  onDelete: (id: string) => void;
  onOpen: (id: string) => void;
  onPreview: (notebook: notebook) => void;
}) {
  return (
    <div className="bg-white p-4 rounded-lg shadow-md relative group min-h-[150px]
flex flex-col border border-gray-200 hover:shadow-lg transition-shadow">
      <div className="flex-1">
        <div className="flex items-center gap-2 mb-2">
          <Notebook className="h-5 w-5 text-blue-600" />
          <h3 className="text-gray-800 font-semibold text-lg">
            {notebook.title}
          </h3>
        </div>
        <p className="text-gray-600 text-sm line-clamp-3 mb-4">
          {notebook.description || "No description"}
        </p>
        <div className="text-xs text-gray-500 mt-auto">
          Last modified: {new Date(notebook.updatedAt).toLocaleDateString()}
        </div>
      </div>

      {/* Action buttons for the notebook, visible on hover */}
      <div className="absolute top-2 right-2 flex gap-2 opacity-0
group-hover:opacity-100 transition-opacity">
        <Button
          variant="ghost"
          size="sm"
          onClick={() => onPreview(notebook)}
          className="text-gray-600 hover:text-gray-800"
        >
          <Eye className="h-4 w-4" />
        </Button>
        <Button
          variant="ghost"
          size="sm"
          onClick={() => onOpen(notebook.id)}
          className="text-blue-600 hover:text-blue-800"
```

```
        >
          Open
        </Button>
        <Button
          variant="ghost"
          size="sm"
          onClick={() => onDelete(notebook.id)}
          className="text-red-600 hover:text-red-800"
        >
          <Trash2 className="h-4 w-4" />
        </Button>
      </div>
    </div>
  );
}

/**
 * Main component for managing and displaying notebooks.
 * It handles fetching, creating, deleting, and previewing notebooks.
 * Uses drag and drop context for potential future sortable notebook lists, although
sorting is not implemented in this version.
 */
export default function NotebookManager() {
  const router = useRouter();
  // State to hold the list of notebooks fetched from the database
  const [notebooks, setNotebooks] = useState<notebook[]>([]);
  // State to control the visibility of the modal for creating a new notebook
  const [isModalOpen, setIsModalOpen] = useState(false);
  // State to control the visibility of the notebook preview dialog
  const [isPreviewOpen, setIsPreviewOpen] = useState(false);
  // State to hold the notebook data that is currently being previewed
  const [selectedNotebook, setSelectedNotebook] = useState<notebook | null>(
    null
  );
  // State to manage the input values for creating a new notebook (title and description)
  const [newNotebook, setNewNotebook] = useState({
    title: "",
    description: "",
  });
```

```
  // Sensors for drag and drop functionality (pointer and keyboard sensors)
  const sensors = useSensors(
    useSensor(PointerSensor),
    useSensor(KeyboardSensor)
  );

  /**
   * Handles opening the preview dialog for a given notebook.
   * Sets the selected notebook to the notebook to be previewed and opens the preview
modal.
   * @param {notebook} notebook - The notebook object to preview.
   */
  const handlePreview = (notebook: notebook) => {
    setSelectedNotebook(notebook);
    setIsPreviewOpen(true);
  };

  // useEffect hook to load notebooks when the component mounts
  useEffect(() => {
    loadNotebooks();
  }, []);

  /**
   * Loads notebooks from the database and updates the component state.
   * Fetches notebooks using `readNotebooks` action and sets the `notebooks` state.
   */
  const loadNotebooks = async () => {
    const fetchedNotebooks = await readNotebooks();
    setNotebooks(fetchedNotebooks);
  };

  /**
   * Handles the submission of the new notebook form.
   * Prevents default form submission, validates input, and calls `addNotebook` action to
create a new notebook.
   * After successful creation, it closes the modal, resets the new notebook input state,
and navigates to the newly created notebook's page.
   * @param {React.FormEvent} e - The form submit event.
   */
  const handleCreateNotebook = async (e: React.FormEvent) => {
```

```
    e.preventDefault();
    if (!newNotebook.title) return;

    try {
      const createdNotebook = await addNotebook({
        title: newNotebook.title,
        description: newNotebook.description,
      });

      setIsModalOpen(false);
      setNewNotebook({ title: "", description: "" });
      router.push(
        `/ContinuousInfoSpaceDocMan/ContinuousInfoSpace/${createdNotebook.id}`
      );
    } catch (error) {
      console.error("Failed to create notebook:", error);
    }
  };

  /**
   * Handles deleting a notebook.
   * Calls the `deleteNotebook` action to remove a notebook from the database and then
updates the local state by filtering out the deleted notebook.
   * @param {string} id - The ID of the notebook to delete.
   */
  const handleDeleteNotebook = async (id: string) => {
    try {
      await deleteNotebook(id);
      setNotebooks((prev) => prev.filter((n) => n.id !== id));
    } catch (error) {
      console.error("Failed to delete notebook:", error);
    }
  };

  return (
    <div className="p-6">
      {/* Header section with title and button to add a new notebook */}
      <div className="flex justify-between items-center mb-6">
        <h1 className="text-2xl font-bold text-gray-800">My Notebooks</h1>
        <Button onClick={() => setIsModalOpen(true)}>
```

```
          <Plus className="h-4 w-4 mr-2" />
          New Notebook
        </Button>
      </div>

      {/* Drag and Drop Context for potential future sortable notebook list */}
      <DndContext sensors={sensors} collisionDetection={closestCenter}>
        <SortableContext items={notebooks} strategy={rectSortingStrategy}>
          <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 xl:grid-cols-4
gap-4">
            {/* Map through notebooks and render each as a NotebookCard */}
            {notebooks.map((notebook) => (
              <NotebookCard
                key={notebook.id}
                notebook={notebook}
                onDelete={handleDeleteNotebook}
                onOpen={(id) =>
                  router.push(
                    `/ContinuousInfoSpaceDocMan/ContinuousInfoSpace/${id}`
                  )
                }
                onPreview={handlePreview}
              />
            ))}
          </div>
        </SortableContext>
      </DndContext>

      {/* Modal for creating a new notebook */}
      {isModalOpen && (
        <div className="fixed inset-0 bg-black/50 flex items-center justify-center">
          <div className="bg-white p-6 rounded-lg w-full max-w-md">
            <h2 className="text-xl font-semibold mb-4">Create New Notebook</h2>
            <form onSubmit={handleCreateNotebook}>
              <div className="mb-4">
                <label className="block text-sm font-medium mb-2">Title</label>
                <input
                  required
                  type="text"
                  value={newNotebook.title}
```

```
              onChange={(e) =>
                setNewNotebook({ ...newNotebook, title: e.target.value })
              }
              className="w-full px-4 py-2 rounded-md border focus:ring-2
focus:ring-blue-500"
            />
          </div>
          <div className="mb-4">
            <label className="block text-sm font-medium mb-2">
              Description
            </label>
            <textarea
              value={newNotebook.description}
              onChange={(e) =>
                setNewNotebook({
                  ...newNotebook,
                  description: e.target.value,
                })
              }
              className="w-full px-4 py-2 rounded-md border focus:ring-2
focus:ring-blue-500"
              rows={3}
            />
          </div>
          <div className="flex justify-end gap-4">
            <Button
              type="button"
              variant="secondary"
              onClick={() => setIsModalOpen(false)}
            >
              Cancel
            </Button>
            <Button type="submit">Create Notebook</Button>
          </div>
        </form>
      </div>
    </div>
  )}

  {/* Dialog for previewing a notebook */}
```

```
<Dialog open={isPreviewOpen} onOpenChange={setIsPreviewOpen}>
  <DialogContent className="max-w-4xl max-h-[80vh] flex flex-col">
    <DialogHeader className="flex-none border-b pb-4">
      <DialogTitle>{selectedNotebook?.title}</DialogTitle>
    </DialogHeader>
    <div className="flex-1 overflow-auto">
      <div className="space-y-4 p-6">
        {/* Render sections and columns of the selected notebook for preview */}
        {selectedNotebook?.sections?.map(
          (section: any, index: number) => (
            <div
              key={section.id || index}
              className="border rounded-lg p-4"
            >
              <h3 className="font-medium text-lg mb-3">
                {section.title}
              </h3>
              <div className="flex gap-4">
                {section.columns?.map((column: any, colIndex: number) => (
                  <div
                    key={column.id || colIndex}
                    className="flex-none w-72 bg-gray-50 rounded-lg p-3"
                  >
                    <h4 className="font-medium mb-2">{column.title}</h4>
                    <div className="space-y-2">
                      {column.notes?.map(
                        (note: any, noteIndex: number) => (
                          <div
                            key={note.id || noteIndex}
                            className="bg-white p-2 rounded border"
                          >
                            {note.content}
                          </div>
                        )
                      )}
                    </div>
                  </div>
                ))}
              </div>
            </div>
```

```
              )
            )}
            {/* Display message if the notebook has no sections */}
            {(!selectedNotebook?.sections ||
              selectedNotebook.sections.length === 0) && (
              <p className="text-gray-500 text-center py-8">
                This notebook is empty
              </p>
            )}
          </div>
        </div>
      </DialogContent>
    </Dialog>
  </div>
 );
}


===========
Directory: src\app\ContinuousInfoSpaceDocMan\ContinuousInfoSpace
===========

File: src\app\ContinuousInfoSpaceDocMan\ContinuousInfoSpace\page.tsx
----------

"use client";

import { use404Redirect } from "@/hooks/use-404-redirect";

export default function ContinuousInfoSpacePage() {
 use404Redirect();

 return (
  <div className="flex items-center justify-center min-h-screen">
   <p>Redirecting...</p>
  </div>
 );
}
```

==========
Directory: src\app\ContinuousInfoSpaceDocMan\ContinuousInfoSpace\[notebookId]
==========

File: src\app\ContinuousInfoSpaceDocMan\ContinuousInfoSpace\[notebookId]\page.tsx
---

```
"use client";

import React, { useEffect, useState } from "react";
import { useRouter, useParams } from "next/navigation";
import { BackButton } from "@/components/ui/custom-ui/back-button";
import {
  getNotebook,
  updateNotebook,
} from "@/lib/continuous-info-space-doc-man-actions";
import { Archive, Plus, X } from "lucide-react";
import { Button } from "@/components/ui/button";
import { useUser } from "@clerk/nextjs";
import {
  Sheet,
  SheetContent,
  SheetHeader,
  SheetTitle,
  SheetTrigger,
} from "@/components/ui/sheet";
import { Card, CardContent, CardHeader, CardTitle } from "@/components/ui/card";
import {
  DndContext,
  DragOverlay,
  useSensors,
  useSensor,
  PointerSensor,
  closestCorners,
  DragStartEvent,
  DragEndEvent,
  useDroppable,
  useDraggable,
} from "@dnd-kit/core";
import { useToast } from "@/hooks/use-toast";
```

```
import { ToastAction } from "@/components/ui/toast";

interface Note {
  id: string;
  content: string;
}

interface Column {
  id: string;
  title: string;
  notes: Note[];
  isEditing: boolean;
}

interface Section {
  id: string;
  title: string;
  columns: Column[];
  isEditing: boolean;
  isArchived?: boolean;
  isPreviewingArchived?: boolean;
}

interface DragData {
  type: "note" | "column";
  sectionId: string;
  columnId?: string;
  noteId?: string;
}

const DraggableColumn = ({
  column,
  section,
  children,
}: {
  column: Column;
  section: Section;
  children: React.ReactNode;
}) => {
  const { attributes, listeners, setNodeRef, transform } = useDraggable({
```

```
    id: `column-${column.id}`,
    data: {
      type: "column",
      sectionId: section.id,
      columnId: column.id,
    },
  });

  const style = transform
    ? {
        transform: `translate3d(${transform.x}px, ${transform.y}px, 0)`,
      }
    : undefined;

  return (
    <div
      ref={setNodeRef}
      style={style}
      {...listeners}
      {...attributes}
      className="flex-none w-80"
    >
      {children}
    </div>
  );
};

const DroppableColumn = ({
  column,
  section,
  children,
}: {
  column: Column;
  section: Section;
  children: React.ReactNode;
}) => {
  const { setNodeRef, isOver } = useDroppable({
    id: `droppable-${column.id}`,
    data: {
      type: "column",
```

```
      sectionId: section.id,
      columnId: column.id,
    },
  });

  return (
    <div
      ref={setNodeRef}
      className={`rounded-lg p-4 ${isOver ? "bg-gray-100" : "bg-gray-50"}`}
    >
      {children}
    </div>
  );
};

const DraggableNote = ({
  note,
  children,
}: {
  note: Note;
  children: React.ReactNode;
}) => {
  const { attributes, listeners, setNodeRef, transform } = useDraggable({
    id: note.id,
    data: {
      type: "note",
      noteId: note.id,
    },
  });

  const style = transform
    ? {
        transform: `translate3d(${transform.x}px, ${transform.y}px, 0)`,
      }
    : undefined;

  return (
    <div
      ref={setNodeRef}
      style={style}
```

```
      {...listeners}
      {...attributes}
      className="bg-[#fff9e6] rounded-lg p-4 cursor-move group"
    >
      {children}
    </div>
  );
};

const NotebooksInterface = () => {
  // User authentication state
  const { user } = useUser();
  const { toast } = useToast();
  const router = useRouter();
  const params = useParams();
  const notebookId = params.notebookId as string;
  const [activeNote, setActiveNote] = useState<Note | null>(null);
  const [sections, setSections] = useState<Section[]>([]);
  const [archivedSections, setArchivedSections] = useState<Section[]>([]);
  const [previewSection, setPreviewSection] = useState<Section | null>(null);
  const sensors = useSensors(
    useSensor(PointerSensor, {
      activationConstraint: {
        distance: 8,
      },
    })
  );

  // Fetch initial data
  useEffect(() => {
    if (!user || !notebookId) return;

    const fetchNotebookData = async () => {
      try {
        const notebook = await getNotebook(notebookId);
        if (notebook) {
          setSections(notebook.sections || []);
        }
      } catch (error) {
        console.error("Error fetching notebook data:", error);
```

```
    }
  };

  fetchNotebookData();
}, [user, notebookId]);

// Sync changes to Firestore with debounce
useEffect(() => {
  if (!user || !notebookId) return;

  // Add debounce to prevent too many writes
  const timeoutId = setTimeout(async () => {
    try {
      await updateNotebook(notebookId, sections);
    } catch (error) {
      console.error("Error updating notebook:", error);
    }
  }, 1000); // Wait 1 second after last change

  return () => clearTimeout(timeoutId);
}, [sections, user, notebookId]);

const archiveSection = (sectionId: string) => {
  const sectionToArchive = sections.find((s) => s.id === sectionId);
  if (sectionToArchive) {
    setSections(sections.filter((s) => s.id !== sectionId));
    setArchivedSections([
      ...archivedSections,
      { ...sectionToArchive, isArchived: true },
    ]);
  }
};

const unarchiveSection = (sectionId: string) => {
  const sectionToUnarchive = archivedSections.find((s) => s.id === sectionId);
  if (sectionToUnarchive) {
    // Close preview if the unarchived section is currently being previewed
    if (previewSection && previewSection.id === sectionId) {
      setPreviewSection(null);
    }
```

```
    setArchivedSections(archivedSections.filter((s) => s.id !== sectionId));
    setSections([...sections, { ...sectionToUnarchive, isArchived: false }]);
  }
};

const previewArchivedSection = (section: Section) => {
  setPreviewSection({ ...section, isPreviewingArchived: true });
};

const closePreview = () => {
  setPreviewSection(null);
};

const addSection = () => {
  const newSection: Section = {
    id: Date.now().toString(),
    title: "New Section",
    isEditing: true,
    columns: [
      {
        id: `col-${Date.now()}`,
        title: "New Column",
        isEditing: false,
        notes: [],
      },
    ],
  };
  setSections([...sections, newSection]);
};

const addColumn = (sectionId: string) => {
  setSections(
    sections.map((section) => {
      if (section.id === sectionId) {
        return {
          ...section,
          columns: [
            ...section.columns,
            {
```

```
            id: `col-${Date.now()}`,
            title: "New Column",
            isEditing: true,
            notes: [],
          },
        ],
      };
    }
    return section;
  })
);
};

const addNote = (sectionId: string, columnId: string) => {
  setSections(
    sections.map((section) => {
      if (section.id === sectionId) {
        return {
          ...section,
          columns: section.columns.map((column) => {
            if (column.id === columnId) {
              return {
                ...column,
                notes: [
                  ...column.notes,
                  {
                    id: `note-${Date.now()}`,
                    content: "New note",
                    isEditing: true,
                  },
                ],
              };
            }
            return column;
          }),
        };
      }
      return section;
    })
  );
```

```
};

const moveNote = (
  fromSectionId: string,
  fromColumnId: string,
  toSectionId: string,
  toColumnId: string,
  noteId: string
) => {
  // First, find the note to move
  const noteToMove = sections
    .find((s) => s.id === fromSectionId)
    ?.columns.find((c) => c.id === fromColumnId)
    ?.notes.find((n) => n.id === noteId);

  if (!noteToMove) return; // Exit if note not found

  setSections(
    sections.map((section) => {
      // If this is not the source or target section, return unchanged
      if (section.id !== fromSectionId && section.id !== toSectionId) {
        return section;
      }

      // Handle source section
      if (section.id === fromSectionId) {
        return {
          ...section,
          columns: section.columns.map((column) => {
            // Remove note from source column
            if (column.id === fromColumnId) {
              return {
                ...column,
                notes: column.notes.filter((note) => note.id !== noteId),
              };
            }
            // Add note to destination column if same section
            if (column.id === toColumnId && fromSectionId === toSectionId) {
              return {
                ...column,
```

```
              notes: [...column.notes, noteToMove],
            };
          }
          return column;
        }),
      };
    }

    // Handle target section (when different from source)
    if (section.id === toSectionId) {
      return {
        ...section,
        columns: section.columns.map((column) => {
          if (column.id === toColumnId) {
            return {
              ...column,
              notes: [...column.notes, noteToMove],
            };
          }
          return column;
        }),
      };
    }

    return section;
  })
 );
};

const moveColumn = (
  sectionId: string,
  fromIndex: number,
  toIndex: number
) => {
  setSections(
    sections.map((section) => {
      if (section.id === sectionId) {
        const newColumns = [...section.columns];
        const [movedColumn] = newColumns.splice(fromIndex, 1);
        newColumns.splice(toIndex, 0, movedColumn);
```

```
        return { ...section, columns: newColumns };
      }
      return section;
    })
  );
};

const handleDragStart = (event: DragStartEvent) => {
  const { active } = event;
  const noteId = active.id as string;

  // Find the dragged note
  const draggedNote = sections
    .flatMap((section) =>
      section.columns.flatMap((column) =>
        column.notes.find((note) => note.id === noteId)
      )
    )
    .find(Boolean);

  if (draggedNote) {
    setActiveNote(draggedNote);
  }
};

const handleDragEnd = (event: DragEndEvent) => {
  const { active, over } = event;

  // Reset activeNote regardless of outcome
  setActiveNote(null);

  // If there's no over target, return early - note will stay in original position
  if (!over) {
    return;
  }

  const activeId = active.id as string;
  const activeData = active.data.current as DragData;
  const overData = over.data.current as DragData;
```

```
// Handle column dragging
if (activeData.type === "column" && overData.type === "column") {
  const sectionId = activeData.sectionId;
  const section = sections.find((s) => s.id === sectionId);
  if (!section) return;

  const fromIndex = section.columns.findIndex(
    (c) => c.id === activeData.columnId
  );
  const toIndex = section.columns.findIndex(
    (c) => c.id === overData.columnId
  );

  if (fromIndex !== toIndex) {
    moveColumn(sectionId, fromIndex, toIndex);
  }
  return;
}

// Handle note dragging
if (activeData.type === "note" || !activeData.type) {
  // Only proceed if we're dropping onto a column
  if (!overData?.type || !overData.columnId) {
    return; // Note will stay in original position
  }

  // Find the source section and column
  let fromSectionId, fromColumnId;
  sections.some((section) => {
    return section.columns.some((column) => {
      const found = column.notes.some((note) => note.id === activeId);
      if (found) {
        fromSectionId = section.id;
        fromColumnId = column.id;
        return true;
      }
      return false;
    });
  });
```

```
    // Get destination details from the over data
    const toSectionId = overData.sectionId;
    const toColumnId = overData.columnId;

    // Only move the note if we have valid source and destination
    if (fromSectionId && fromColumnId && toSectionId && toColumnId) {
      moveNote(
        fromSectionId,
        fromColumnId,
        toSectionId,
        toColumnId,
        activeId
      );
    }
  }
};

const deleteSection = (sectionId: string) => {
  const section = sections.find((s) => s.id === sectionId);
  if (!section) return;

  toast({
    title: "Delete Section?",
    description: `Are you sure you want to delete "${section.title}"?`,
    action: (
      <ToastAction
        altText="Delete"
        onClick={() => {
          setSections(sections.filter((section) => section.id !== sectionId));
          toast({
            title: "Section deleted",
            description: "The section has been deleted successfully.",
          });
        }}
      >
        Delete
      </ToastAction>
    ),
  });
};
```

```
const deleteColumn = (sectionId: string, columnId: string) => {
  const column = sections
    .find((s) => s.id === sectionId)
    ?.columns.find((c) => c.id === columnId);
  if (!column) return;

  toast({
    title: "Delete Column?",
    description: `Are you sure you want to delete "${column.title}"?`,
    action: (
      <ToastAction
        altText="Delete"
        onClick={() => {
          setSections(
            sections.map((section) => {
              if (section.id === sectionId) {
                return {
                  ...section,
                  columns: section.columns.filter(
                    (column) => column.id !== columnId
                  ),
                };
              }
              return section;
            })
          );
          toast({
            title: "Column deleted",
            description: "The column has been deleted successfully.",
          });
        }}
      >
        Delete
      </ToastAction>
    ),
  });
};

const deleteNote = (sectionId: string, columnId: string, noteId: string) => {
```

```
const note = sections
  .find((s) => s.id === sectionId)
  ?.columns.find((c) => c.id === columnId)
  ?.notes.find((n) => n.id === noteId);
if (!note) return;

toast({
  title: "Delete Note?",
  description: `Are you sure you want to delete this note?`,
  action: (
    <ToastAction
      altText="Delete"
      onClick={() => {
        setSections(
          sections.map((section) => {
            if (section.id === sectionId) {
              return {
                ...section,
                columns: section.columns.map((column) => {
                  if (column.id === columnId) {
                    return {
                      ...column,
                      notes: column.notes.filter(
                        (note) => note.id !== noteId
                      ),
                    };
                  }
                  return column;
                }),
              };
            }
            return section;
          })
        );
        toast({
          title: "Note deleted",
          description: "The note has been deleted successfully.",
        });
      }}
    >
```

```
        Delete
      </ToastAction>
    ),
  });
};

return (
  <div className="flex flex-col h-screen">
    <header className="flex-none flex items-center justify-between p-4 border-b
bg-white">
      <BackButton />

      {/* Archive Button */}
      <Sheet>
        <SheetTrigger asChild>
          <Button variant="outline" size="icon">
            <Archive className="h-4 w-4" />
          </Button>
        </SheetTrigger>
        <SheetContent>
          <SheetHeader>
            <SheetTitle>Archived Sections</SheetTitle>
          </SheetHeader>
          <div className="mt-4 space-y-4">
            {archivedSections.map((section) => (
              <Card key={section.id}>
                <CardHeader className="flex flex-row items-center justify-between
space-y-0 pb-2">
                  <CardTitle className="text-sm font-medium">
                    {section.title}
                  </CardTitle>
                  <Button
                    variant="ghost"
                    size="sm"
                    onClick={() => unarchiveSection(section.id)}
                  >
                    Unarchive
                  </Button>
                </CardHeader>
                <CardContent>
```

```
            <Button
              variant="secondary"
              size="sm"
              className="w-full"
              onClick={() => previewArchivedSection(section)}
            >
              Preview
            </Button>
          </CardContent>
        </Card>
      ))}
    </div>
  </SheetContent>
 </Sheet>
</header>
<DndContext
 sensors={sensors}
 collisionDetection={closestCorners}
 onDragStart={handleDragStart}
 onDragEnd={handleDragEnd}
>
 <div className="max-w-none max-h-none bg-white min-h-screen pb-20">
  {/* Archived Sections */}

  {/* Preview Section (if active) */}
  {previewSection && (
   <div className="border-2 border-gray-200 rounded-lg mb-8">
    <div className="flex justify-between items-center px-4 py-2 bg-white">
     <h2 className="text-lg font-medium">
      {previewSection.title}{" "}
      <span className="text-gray-500 text-sm">(Preview)</span>
     </h2>
     <Button variant="ghost" size="sm" onClick={closePreview}>
      <X className="h-4 w-4" />
     </Button>
    </div>
    <div className="px-4 py-2 overflow-x-auto">
     <div className="flex flex-col space-y-4">
      <div className="flex space-x-4 min-h-[200px] pb-4">
       {previewSection.columns.map((column) => (
```

```
            <div
              key={column.id}
              className="flex-none w-80 rounded-lg p-4 bg-gray-50"
            >
              <h3 className="font-medium text-gray-700 mb-4">
                {column.title}
              </h3>
              <div className="space-y-4">
                {column.notes.map((note) => (
                  <div
                    key={note.id}
                    className="bg-[#fff9e6] rounded-lg p-4"
                  >
                    <p className="text-gray-800">{note.content}</p>
                  </div>
                ))}
              </div>
            </div>
          ))}
        </div>
        <div className="flex justify-end px-4">
          <Button
            variant="secondary"
            size="sm"
            onClick={closePreview}
          >
            Close Preview
          </Button>
        </div>
      </div>
    </div>
  </div>
)}

{sections.map((section) => (
  <div
    key={section.id}
    className="border-b border-gray-100 mb-8 overflow-x-auto"
  >
    {/* Section Header */}
```

```
            <div className="flex justify-between items-center px-4 py-2 bg-white group">
              <div className="flex items-center space-x-2">
                <h2
                  className="text-lg font-medium outline-none focus:outline-none"
                  contentEditable
                  suppressContentEditableWarning
                  onBlur={(e) => {
                    setSections(
                      sections.map((s) =>
                        s.id === section.id
                          ? {
                              ...s,
                              title: e.currentTarget.textContent || s.title,
                            }
                          : s
                      )
                    );
                  }}
                >
                  {section.title}
                </h2>
              </div>
              <div className="flex items-center space-x-2 opacity-0
group-hover:opacity-100 transition-opacity">
                <Button
                  variant="ghost"
                  size="sm"
                  className="text-gray-500"
                  onClick={() => archiveSection(section.id)}
                >
                  Archive
                </Button>
                <Button
                  variant="ghost"
                  size="sm"
                  className="text-red-500"
                  onClick={() => deleteSection(section.id)}
                >
                  <X className="h-4 w-4" />
                </Button>
```

```
          </div>
        </div>

        {/* Notebooks Board */}
        <div className="px-4 py-2">
          <div className="flex space-x-4 min-h-[200px] pb-4">
            {section.columns.map((column) => (
              <DraggableColumn
                key={column.id}
                column={column}
                section={section}
              >
                <DroppableColumn
                  key={column.id}
                  column={column}
                  section={section}
                >
                  {/* Column Header */}
                  <div className="flex justify-between items-center mb-4 group">
                    <div className="flex items-center justify-between w-full">
                      <h3
                        className="font-medium text-gray-700 outline-none
focus:outline-none"
                        contentEditable
                        suppressContentEditableWarning
                        onBlur={(e) => {
                          setSections(
                            sections.map((s) =>
                              s.id === section.id
                                ? {
                                    ...s,
                                    columns: s.columns.map((c) =>
                                      c.id === column.id
                                        ? {
                                            ...c,
                                            title:
                                              e.currentTarget
                                                .textContent || c.title,
                                          }
                                        : c
```

```
                    ),
                  }
                : s
            )
          );
        }}
      >
        {column.title}
      </h3>
      <div className="flex items-center space-x-1 opacity-0
group-hover:opacity-100 transition-opacity">
        <Button
          variant="ghost"
          size="sm"
          className="text-red-500"
          onClick={() =>
            deleteColumn(section.id, column.id)
          }
        >
          <X className="h-4 w-4" />
        </Button>
      </div>
    </div>
  </div>

  {/* Notes */}
  <div className="space-y-4">
    {column.notes.map((note) => (
      <DraggableNote key={note.id} note={note}>
        <div className="flex justify-between items-start">
          <p
            contentEditable
            suppressContentEditableWarning
            className="text-gray-800 outline-none focus:outline-none"
            onBlur={(e) => {
              setSections(
                sections.map((s) =>
                  s.id === section.id
                    ? {
                        ...s,
```

```
                                columns: s.columns.map((c) =>
                                  c.id === column.id
                                    ? {
                                        ...c,
                                        notes: c.notes.map((n) =>
                                          n.id === note.id
                                            ? {
                                                ...n,
                                                content:
                                                  e.currentTarget
                                                    .textContent ||
                                                  "",
                                              }
                                            : n
                                        ),
                                      }
                                    : c
                                ),
                              }
                            : s
                        )
                      );
                    }}
                  >
                    {note.content}
                  </p>
                  <div className="flex items-center space-x-1 opacity-0
group-hover:opacity-100 transition-opacity">
                    <Button
                      variant="ghost"
                      size="sm"
                      className="text-red-500 h-4 w-4 p-0"
                      onClick={() =>
                        deleteNote(section.id, column.id, note.id)
                      }
                    >
                      <X className="h-4 w-4" />
                    </Button>
                  </div>
                </div>
```

```
                </DraggableNote>
              ))}
            </div>

            {/* Add Note Button */}
            <Button
              variant="ghost"
              size="sm"
              className="w-full mt-4"
              onClick={() => addNote(section.id, column.id)}
            >
              <Plus className="h-4 w-4 mr-2" />
              Add Note
            </Button>
          </DroppableColumn>
        </DraggableColumn>
      ))}

      {/* Add Column Button */}
      <Button
        variant="outline"
        className="flex-none w-80 h-12"
        onClick={() => addColumn(section.id)}
      >
        <Plus className="h-4 w-4 mr-2" />
        Add Column
      </Button>
    </div>
  </div>
</div>
))}

{/* Drag Overlay */}
<DragOverlay>
  {activeNote ? (
    <div className="bg-[#fff9e6] rounded-lg p-4 w-72 shadow-lg">
      <div className="flex justify-between items-start">
        <p className="text-gray-800">{activeNote.content}</p>
      </div>
    </div>
```

```
            ) : null}
          </DragOverlay>

          {/* Add Section Button */}
          <Button
            variant="outline"
            size="lg"
            className="fixed bottom-8 left-1/2 transform -translate-x-1/2"
            onClick={addSection}
          >
            <Plus className="h-4 w-4 mr-2" />
            Add Section
          </Button>
        </div>
      </DndContext>
    </div>
  );
};

export default NotebooksInterface;
```

Stage Manager Module

```
==========
Directory: src\app\stage-manager
==========

File: src\app\stage-manager\page.tsx
--------------------

"use client";

import { Manager, Space, BasicWindow } from "@/components/stage-manager";

export default function StageDemo() {
  return (
    <div style={{ padding: "2rem", background: "#f0f0f0", minHeight: "100vh" }}>
      <h1 style={{ marginBottom: "2rem" }}>Stage Manager Demo</h1>

      <Manager
        size={[1280, 720]}
        style={{
          margin: "0 auto",
          border: "2px solid #333",
          borderRadius: "8px",
        }}
      >
        <Space>
          <BasicWindow
            title="Document 1"
            initialPosition={[100, 100]}
            initialSize={[400, 300]}
            style={{ background: "#fff" }}
          >
            <div style={{ padding: "1rem" }}>
              <h2>Document Content</h2>
              <p>Drag the title bar to move the window</p>
              <p>Drag edges to resize</p>
            </div>
          </BasicWindow>
```

```
        <BasicWindow
          title="Browser"
          initialPosition={[300, 200]}
          initialSize={[500, 400]}
          style={{ background: "#fff" }}
        >
          <div style={{ padding: "1rem" }}>
            <h3>Web Content</h3>
            <p>
              Try dragging windows close to each other to see snapping
              behavior
            </p>
          </div>
        </BasicWindow>

        <BasicWindow
          title="Settings"
          initialPosition={[700, 150]}
          initialSize={[300, 250]}
          style={{ background: "#fff" }}
        >
          <div style={{ padding: "1rem" }}>
            <p>System Settings</p>
            <ul>
              <li>Display</li>
              <li>Network</li>
              <li>Storage</li>
            </ul>
          </div>
        </BasicWindow>
      </Space>
    </Manager>
  </div>
 );
}


===========
Directory: src\components\stage-manager
===========
```

File: src\components\stage-manager\contexts.ts

```
export { ManagerContext } from "./components/Manager";
export type { ManagerContextProps } from "./components/Manager/library";
export { SpaceContext } from "./components/Space";
export type { SpaceContextProps } from "./components/Space/library";
export { WindowContext } from "./components/Window";
export type { WindowContextProps } from "./components/Window";
```

File: src\components\stage-manager\hooks.ts

```
import { useState, useContext } from "react";

import { ManagerContext, SpaceContext } from "./contexts";

/**
 * Hook to generate and manage a unique ID for a space.
 *
 * @returns {[string, React.Dispatch<React.SetStateAction<string>>]} An array
containing:
 *   - The unique space ID (string).
 *   - A function to set or update the space ID.
 */
export function useSpaceId(): [
  string,
  React.Dispatch<React.SetStateAction<string>>
] {
  const uuid =
    Math.random().toString(36).substring(2, 15) +
    Math.random().toString(36).substring(2, 15);
  const [state, setState] = useState<string>(uuid);
  return [state, setState] as const;
}

/**
 * Hook to manage the position of a window.
```

```
 *
 * It leverages ManagerContext to get the manager size and SpaceContext
 * to access the last window position for 'auto' placement.
 *
 * @param {[number, number] | "random" | "auto"} initial_position - Initial position of the
window.
 *   Can be a fixed [x, y] coordinate, "random" for a random position within the manager,
 *   or "auto" to position the window slightly offset from the last window position.
 *
 * @returns {[[number, number], React.Dispatch<React.SetStateAction<[number,
number]>>]} An array containing:
 *   - The current window position [x, y] (number array).
 *   - A function to set or update the window position.
 */
export function usePosition(
  initial_position: [number, number] | "random" | "auto"
): [[number, number], React.Dispatch<React.SetStateAction<[number, number]>>] {
  const { size } = useContext(ManagerContext);
  const { lastWindowPosition } = useContext(SpaceContext);

  let position: [number, number];

  if (initial_position === "random") {
    position = [
      Math.floor(Math.random() * size[0]),
      Math.floor(Math.random() * size[1]),
    ];
  } else if (initial_position === "auto") {
    position = [lastWindowPosition[0] + 20, lastWindowPosition[1] + 20];
  } else {
    position = initial_position;
  }

  const [state, setState] = useState<[number, number]>(position);

  return [state, setState] as const;
}

/**
 * Hook to manage the size of a window.
```

```
 *
 * @param {[number, number]} initial_size - Initial width and height of the window [width,
height].
 *
 * @returns {[[number, number], React.Dispatch<React.SetStateAction<[number,
number]>>]} An array containing:
 *   - The current window size [width, height] (number array).
 *   - A function to set or update the window size.
 */
export function useSize(
  initial_size: [number, number]
): [[number, number], React.Dispatch<React.SetStateAction<[number, number]>>] {
  const [state, setState] = useState<[number, number]>(initial_size);

  return [state, setState] as const;
}
```

File: src\components\stage-manager\index.ts

```
export * from "./space";
export * from "./components";
export * from "./contexts";
export * from "./hooks";
```

File: src\components\stage-manager\space.ts

```
// src/components/stage-manager/space.ts
/* eslint-disable @typescript-eslint/no-explicit-any */
/**
 * @fileoverview This file defines utility functions and classes for the stage manager
space,
 * including version information, device detection, ID hashing, position validation, and
 * an event dispatcher class.
 */

/**
```

```
 * @const {string} version - Current version of the stage manager space module.
 * Useful for tracking updates and ensuring compatibility.
 */
export const version = "0.4.3";

/**
 * @function isMobileDevice
 * @returns {boolean} - True if the user agent indicates a mobile device, false otherwise.
 * @description Detects if the current user agent is likely a mobile device.
 * This is used to conditionally apply mobile-specific behaviors or UI adjustments within
the stage manager.
 * It's important to check `typeof window !== 'undefined'` to ensure this code runs in a
browser environment
 * and not on the server, where `window` and `navigator` are not available.
 */
export function isMobileDevice() {
  if (typeof window === "undefined") return false; // Server-side guard
  return /Mobi|Android|iPhone|iPad|iPod|BlackBerry|IEMobile|Opera Mini/i.test(
    navigator.userAgent
  );
}

/**
 * @function hashSpaceIds
 * @param {string[]} ids - An array of space IDs (typically window IDs).
 * @returns {string} - A deterministic hash string generated from the sorted space IDs.
 * @description Generates a consistent hash from an array of space IDs.
 * This is useful for uniquely identifying a set of spaces, regardless of their order in the
array.
 * Sorting ensures that the same set of IDs always produces the same hash, which is
critical
 * for features like snapping or grouping windows based on their IDs.
 */
export function hashSpaceIds(ids: string[]): string {
  return ids.toSorted((a, b) => a.localeCompare(b)).join("");
}

/**
 * @function nonZeroPosition
```

```
 * @param {[number, number]} position - A position array representing [x, y]
coordinates.
 * @returns {[number, number]} - A position array where both x and y are guaranteed to
be non-negative.
 * @description Ensures that a given position is within the bounds of the visible
workspace by making sure
 * both x and y coordinates are not negative. This is a utility function to prevent windows
from being placed
 * off-screen to the left or top. It effectively clamps the position to the origin (0, 0) if it's
negative.
 */
export function nonZeroPosition(position: [number, number]): [number, number] {
  const compensated: [number, number] = [
    position[0] < 0 ? 0 : position[0],
    position[1] < 0 ? 0 : position[1],
  ];
  return compensated;
}

/**
 * @class EventDispatcher
 * @template T - Type of event names (e.g., string literals for specific event types).
 * @template E - Type of event objects passed to listeners.
 * @description A generic event dispatcher class for managing and dispatching custom
events.
 * It allows adding, setting, dispatching, and removing event listeners for different event
types.
 * This pattern is useful for decoupling components and enabling them to communicate
through events
 * without direct dependencies.
 */
export class EventDispatcher<T, E> {
  /**
   * @private
   * @member {Map<T, ((event: E) => void)[]>} listeners - A map storing event listeners
for each event type.
   * The keys are event types (T), and the values are arrays of listener functions.
   */
  private listeners: Map<T, ((event: E) => void)[]> = new Map();
```

```
  /**
   * @method addListener
   * @param {T} type - The event type to listen for.
   * @param {(event: C) => void} listener - The listener function to be called when the
event is dispatched.
   * @template C - Type constraint for the event object, ensuring type safety.
   * @description Adds a new listener function for a specific event type.
   * If listeners already exist for this type, the new listener is appended to the list.
   */
  addListener<C>(type: T, listener: (event: C) => void) {
    const listeners = this.listeners.get(type);
    if (!listeners) {
      this.listeners.set(type, [listener as any]);
      return;
    }
    listeners.push(listener as any);
  }

  /**
   * @method setListener
   * @param {T} type - The event type to set the listener for.
   * @param {(event: C) => void} listener - The listener function to be set.
   * @template C - Type constraint for the event object.
   * @description Sets or updates a listener function for a specific event type.
   * If listeners already exist, it replaces the first occurrence of the given listener.
   * If the listener is not found, it is added as a new listener.
   */
  setListener<C>(type: T, listener: (event: C) => void) {
    const listeners = this.listeners.get(type);
    if (!listeners) {
      this.listeners.set(type, [listener as any]);
      return;
    }
    const index = listeners.indexOf(listener as any);
    if (index !== -1) listeners[index] = listener as any;
    else listeners.push(listener as any);
  }

  /**
   * @method dispatch
```

```
   * @param {T} name - The event type to dispatch.
   * @param {E} event - The event object to pass to listeners.
   * @description Dispatches an event of a specific type, calling all registered listeners
with the event object.
   * If no listeners are registered for the event type, this method does nothing.
   */
  dispatch(name: T, event: E) {
    const listeners = this.listeners.get(name as any);
    if (!listeners) return;
    listeners.forEach((listener) => listener(event));
  }

  /**
   * @method removeListener
   * @param {T} type - The event type from which to remove the listener.
   * @param {(event: C) => void} listener - The listener function to remove.
   * @template C - Type constraint for the event object.
   * @description Removes a specific listener function for a given event type.
   * If the listener is found, it is removed from the list of listeners for that type.
   */
  removeListener<C>(type: T, listener: (event: C) => void) {
    const listeners = this.listeners.get(type as any);
    if (!listeners) return;
    const index = listeners.indexOf(listener as any);
    if (index !== -1) listeners.splice(index, 1);
  }

  /**
   * @method removeListeners
   * @param {T} type - The event type for which to remove all listeners.
   * @description Removes all listener functions for a specific event type.
   * After calling this, no listeners will be registered for the given event type.
   */
  removeListeners(type: T) {
    this.listeners.delete(type);
  }

  /**
   * @method removeAllListeners
   * @description Removes all listeners for all event types managed by this dispatcher.
```

```
   * This effectively resets the event dispatcher to its initial state with no active listeners.
   */
  removeAllListeners() {
    this.listeners.clear();
  }
}


===========
Directory: src\components\stage-manager\components
===========

File: src\components\stage-manager\components\index.ts
----------

export * from "./Manager";
export * from "./Space";
export * from "./Window";


===========
Directory: src\components\stage-manager\components\Manager
===========

File: src\components\stage-manager\components\Manager\Manager.tsx
----------

import React, {
  useState,
  useMemo,
  useCallback,
  HTMLAttributes,
  useEffect,
  useRef,
} from "react";
import { ManagerContext } from "./library";
import clsx from "clsx";

const DEFAULT_SIZE: [number, number] = [800, 600];
const DEFAULT_SCALE: [number, number] = [1, 1];
```

```
interface ManagerProps extends React.HTMLAttributes<HTMLDivElement> {
  children?: React.ReactNode;
  size?: [number, number];
  scale?: [number, number];
}

/**
 * Manager Component: Provides the context for managing windows within a space.
 * It handles scaling, positioning, and interaction events for all child components
(Spaces and Windows).
 */
export function Manager({
  children = null,
  size = DEFAULT_SIZE,
  scale = DEFAULT_SCALE,
  ...attrs
}: ManagerProps) {
  // State to track the manager's position in the viewport.
  const [position, setPosition] = useState<[number, number]>([0, 0]);
  // State to track the pointer's position relative to the manager.
  const [pointer, setPointer] = useState<[number, number]>([0, 0]);
  // State to track if the left mouse button is pressed.
  const [lmb, setLmb] = useState<boolean>(false);
  // State to indicate if the wheel is currently busy (e.g., scrolling within a staged
window).
  const [wheelBusy, setWheelBusy] = useState<boolean>(false);
  // useRef to hold the IntersectionObserver for detecting when the component's position
changes in the viewport.
  const intersectionObserverRef = useRef<IntersectionObserver | null>(null);
  // useRef to store the previous DOMRect of the component, used to detect position
changes.
  const prevRect = useRef<DOMRect | null>(null);

  /**
   * useCallback hook for scaling X coordinates based on the manager's scale.
   * @param x - The x-coordinate to scale.
   * @returns The scaled x-coordinate.
   */
  const scaleX = useCallback((x: number) => x * scale[0], [scale]);
```

```
/**
 * useCallback hook for scaling Y coordinates based on the manager's scale.
 * @param y - The y-coordinate to scale.
 * @returns The scaled y-coordinate.
 */
const scaleY = useCallback((y: number) => y * scale[1], [scale]);

/**
 * useCallback hook for reverting scaled X coordinates back to original coordinates.
 * @param x - The scaled x-coordinate.
 * @returns The original x-coordinate.
 */
const revertScaleX = useCallback((x: number) => x / scale[0], [scale]);
/**
 * useCallback hook for reverting scaled Y coordinates back to original coordinates.
 * @param y - The scaled y-coordinate.
 * @returns The original y-coordinate.
 */
const revertScaleY = useCallback((y: number) => y / scale[1], [scale]);

// Memoized context value to be provided to all children components.
const contextProps = useMemo(
  () => ({
    position,
    pointer,
    setPointer,
    lmb,
    size,
    wheelBusy,
    setWheelBusy,
    scale,
    scaleX,
    scaleY,
    revertScaleX,
    revertScaleY,
  }),
  [
    position,
    pointer,
    lmb,
```

```
      size,
      wheelBusy,
      scale,
      scaleX,
      scaleY,
      revertScaleX,
      revertScaleY,
    ]
  );

  // useEffect hook to reset wheelBusy state when the left mouse button is released.
  useEffect(() => {
    !lmb && setWheelBusy(false);
  }, [lmb]);

  /**
   * useCallback hook for handling mouse move events within the manager.
   * Updates the pointer position relative to the manager's bounding rectangle.
   * @param event - The mouse move event.
   */
  const onMouseMove = useCallback((event: React.MouseEvent<HTMLDivElement>)
=> {
    const rect = event.currentTarget.getBoundingClientRect();
    const new_pointer: [number, number] = [
      event.clientX - rect.x,
      event.clientY - rect.y,
    ];
    setPointer(new_pointer);
  }, []);

  /**
   * useCallback hook for handling touch move events within the manager.
   * Updates the pointer position based on touch coordinates, similar to mouse move.
   * @param event - The touch move event.
   */
  const onTouchMove = useCallback((event: React.TouchEvent<HTMLDivElement>) =>
{
    if (event.touches.length !== 1) return;
    const touch = event.touches[0];
    const rect = event.currentTarget.getBoundingClientRect();
```

```
  const new_pointer: [number, number] = [
    touch.clientX - rect.x,
    touch.clientY - rect.y,
  ];
  setPointer(new_pointer);
}, []);

return (
  // ManagerContext.Provider to make contextProps available to child components.
  <ManagerContext.Provider value={contextProps}>
    <div
      {...(attrs as HTMLAttributes<HTMLDivElement>)}
      ref={(ref) => {
        // Disconnect any existing IntersectionObserver to prevent memory leaks.
        if (intersectionObserverRef.current)
          intersectionObserverRef.current.disconnect();
        if (!ref) return;
        // Create a new IntersectionObserver to watch for intersection changes.
        const observer = new IntersectionObserver(
          (entries) => {
            if (entries.length === 0) return;
            const entry = entries[0];
            // If the component is not intersecting, no position update needed.
            if (!entry.isIntersecting) return;

            const rect = ref.getBoundingClientRect();

            // Prevent position update if rect is same as previous rect to avoid infinite loops.
            if (
              !prevRect.current ||
              (prevRect.current.x == rect.x && prevRect.current.y == rect.y)
            )
              return;

            // Update the manager's position based on the new bounding rectangle.
            setPosition([rect.x, rect.y]);
            prevRect.current = rect;
          },
          { threshold: 0 }
        );
```

```
        // Start observing the current element.
        observer.observe(ref);
        intersectionObserverRef.current = observer;
      }}
      className={clsx("relative overflow-hidden touch-none", attrs.className)}
      style={{
        width: size[0],
        height: size[1],
        transform: `scale(${scale[0]}, ${scale[1]})`,
        ...attrs.style,
      }}
      onMouseMove={onMouseMove}
      onTouchMove={onTouchMove}
      // Event handlers to update lmb state on mouse and touch interactions.
      onMouseDown={() => setLmb(true)}
      onMouseUp={() => setLmb(false)}
      onTouchStart={() => setLmb(true)}
      onTouchEnd={() => setLmb(false)}
    >
      {children}
    </div>
  </ManagerContext.Provider>
 );
}
```

File: src\components\stage-manager\components\Manager\index.ts
---

```
export { Manager } from "./Manager";
export { ManagerContext } from "./library";
```

File: src\components\stage-manager\components\Manager\library.ts
---

```
import { createContext } from "react";

export interface ManagerContextProps {
  position: [number, number];
```

```
  size: [number, number];
  lmb: boolean;
  pointer: [number, number];
  setPointer: React.Dispatch<React.SetStateAction<[number, number]>>;
  wheelBusy: boolean;
  setWheelBusy: React.Dispatch<React.SetStateAction<boolean>>;
  scale: [number, number];
  scaleX: (x: number) => number;
  scaleY: (y: number) => number;
  revertScaleX: (x: number) => number;
  revertScaleY: (y: number) => number;
}

export const ManagerContext = createContext<ManagerContextProps>({
  position: [0, 0],
  size: [0, 0],
  lmb: false,
  pointer: [0, 0],
  setPointer: () => {},
  wheelBusy: false,
  setWheelBusy: () => {},
  scale: [1, 1],
  scaleX: (x: number) => x,
  scaleY: (y: number) => y,
  revertScaleX: (x: number) => x,
  revertScaleY: (y: number) => y,
});


==========
Directory: src\components\stage-manager\components\Space
==========

File: src\components\stage-manager\components\Space\Overlay.module.css
----------

/* Style for the overlay component that covers the entire viewport. */
.overlay {
  position: fixed;
  inset: 0;
```

```
  background: rgba(0, 0, 0, 0.8);
  display: flex;
  align-items: center;
  justify-content: center;
  z-index: 1000;
}

/* Style for the horizontal row that contains the space thumbnails. */
.horizontalRow {
  display: flex;
  gap: 16px;
}

/* Style for each space thumbnail. */
.thumbnail {
  width: 120px;
  height: 90px;
  background: #fff;
  display: flex;
  align-items: center;
  justify-content: center;
  border-radius: 8px;
  transition: transform 0.3s ease, opacity 0.3s ease;
  opacity: 0.8;
}

/* Style for the active state of a space thumbnail. */
.thumbnail.active {
  transform: scale(1.1);
  border: 2px solid #007aff;
  opacity: 1;
}

/* Style for the label within each thumbnail. */
.label {
  font-size: 14px;
  font-weight: bold;
  color: #333;
}
```

File: src\components\stage-manager\components\Space\Overlay.tsx
---

```
import React from "react";
import classNames from "classnames";
import styles from "./Overlay.module.css";

/**
 * Component for displaying an overlay with space thumbnails.
 * This overlay is typically used to switch between different workspaces or spaces within the application.
 */
export function Overlay({
  spaces, // Number of spaces to display thumbnails for
  activeSpace, // Index of the currently active space
  onSpaceHover, // Callback function to handle space thumbnail hover, typically to preview a space
}: {
  spaces: number;
  activeSpace: number;
  onSpaceHover: (space: number) => void;
}) {
  return (
    <div className={styles.overlay}>
      <div className={styles.horizontalRow}>
        {/* Map over the number of spaces to create thumbnail elements */}
        {Array.from({ length: spaces }).map((_, index) => (
          <div
            key={index}
            className={classNames(styles.thumbnail, {
              [styles.active]: index === activeSpace, // Apply 'active' style if this thumbnail represents the active space
            })}
            onMouseEnter={() => onSpaceHover(index)} // Call onSpaceHover callback when mouse enters a thumbnail, passing the space index
          >
            <div className={styles.label}>Space {index + 1}</div>
          </div>
        ))}
      </div>
```

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

File: src\components\stage-manager\components\Space\Space.module.css
---

```
/* Style for the Space container component */
.Space {
  position: relative; /* Position context for absolute positioning of child elements */
  perspective: 800px; /* Perspective for 3D effects on staged windows */
}

/* Style for the container holding window components within the Space, using display:
contents to avoid extra DOM element */
.Space_windows {
  display: contents; /* Allows children to behave as if they are direct children of the
Space component */
}

/* Style for the staged windows area on the side of the Space */
.Space_stageds {
  position: absolute; /* Positioned absolutely relative to the Space container */
  top: 0px;
  left: 0px;
  display: flex; /* Use flexbox for layout */
  flex-direction: column; /* Stack staged windows vertically */
  flex-wrap: nowrap; /* Prevent wrapping of staged windows */
  justify-content: flex-start; /* Align items to the start of the container */
  align-items: center; /* Center items horizontally */
  width: 150px; /* Fixed width for the staged windows area */
  height: 100%; /* Full height of the Space container */
  box-sizing: border-box; /* Include padding and border in element's total width and
height */
  user-select: none; /* Prevent text selection in the staged area */
  -webkit-user-select: none;
  -moz-user-select: none;
  perspective: 20rem; /* Perspective for staged windows 3D effect */
  transition: all 0.5s; /* Smooth transition for opacity and pointer-events */
```

```
  overflow-y: auto; /* Enable vertical scrolling if content overflows */
  padding: 50px 10px; /* Padding around staged windows */
  -ms-overflow-style: none; /* Hide scrollbar for IE and Edge */
  scrollbar-width: none; /* Hide scrollbar for Firefox */
}
/* Hide scrollbar for Chrome, Safari and Opera */
.Space_stageds::-webkit-scrollbar {
  display: none;
}
/* Pseudo-elements to ensure full height for flexbox layout in staged area */
.Space_stageds::before,
.Space_stageds::after {
  content: "";
  flex: 1 0 auto;
  min-height: 0;
}

/* Style to hide stageds area when autoHideStageds is enabled */
.Space.Space__autoHideStageds .Space_stageds {
  pointer-events: none; /* Disable interaction with staged windows when hidden */
  opacity: 0; /* Make staged windows area transparent */
}
/* Style to show stageds area when Space__showStageds is enabled, overriding
autoHide if both classes are present */
.Space.Space__showStageds .Space_stageds,
.Space.Space__autoHideStageds.Space__showStageds .Space_stageds {
  pointer-events: all; /* Enable interaction with staged windows */
  opacity: 1; /* Make staged windows area opaque */
}

/* Style for the snapping indicator in the Space */
.Space_snap {
  display: flex; /* Use flexbox for layout */
  position: absolute; /* Positioned absolutely within the Space */
  align-items: center; /* Center items vertically */
  justify-content: center; /* Center items horizontally */
  flex-direction: column; /* Arrange items in a column */
  gap: 5px; /* Spacing between snap elements */
}
/* Animation for the snapping indicator to appear */
```

```
.Space_snap.Space_snap__show {
  animation: SnappingHorizontalAppear 0.25s
    cubic-bezier(0.68, -0.55, 0.265, 1.75) forwards;
}
/* Animation for the snapping indicator when snapped */
.Space_snap.Space_snap__snapped {
  animation: SnappingHorizontalAppear 0.25s
    cubic-bezier(0.68, -0.55, 0.265, 1.75) forwards;
}

/* Style for the mover element of the snapping indicator */
.Space_snap_mover {
  position: relative; /* Position context for absolute positioning */
  width: 10px;
  height: 10px;
  border-radius: 100%; /* Make it circular */
  background: white;
  border-radius: 20px;
  transition: all 0.5s; /* Smooth transition for changes */
  cursor: move; /* Indicate draggable element */
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.25); /* Shadow for depth */
  animation: SnappingHorizontalBottomToTop 0.75s
    cubic-bezier(0.68, -0.55, 0.265, 1.75) forwards; /* Entrance animation */
}

/* Style for the resizer element of the snapping indicator */
.Space_snap_resizer {
  position: relative; /* Position context for absolute positioning */
  width: 50%; /* Relative width to parent */
  max-width: 10px; /* Maximum width to limit size */
  height: 50px;
  background: white;
  border-radius: 20px;
  transition: all 0.5s; /* Smooth transition for changes */
  cursor: ew-resize; /* Indicate horizontal resize cursor */
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.25); /* Shadow for depth */
  animation: SnappingHorizontalTopToBottom 0.33s
    cubic-bezier(0.68, -0.55, 0.265, 1.75); /* Entrance animation */
}
```

```
/* Set transform origin for scaling and rotation animations */
.Space_snap_resizer,
.Space_snap_mover {
  transform-origin: center;
}

/* Keyframe animation for snapping indicator appearance */
@keyframes SnappingHorizontalAppear {
  0% {
    scale: 0; /* Start with no scale */
  }
  100% {
    scale: 1; /* End at normal scale */
  }
}

/* Keyframe animation for mover element entrance */
@keyframes SnappingHorizontalBottomToTop {
  0% {
    transform: translateY(50px); /* Start below normal position */
  }
  100% {
    transform: translateY(0%); /* End at normal position */
  }
}

/* Keyframe animation for resizer element entrance */
@keyframes SnappingHorizontalTopToBottom {
  0% {
    transform: translateY(-50px); /* Start above normal position */
  }
  100% {
    transform: translateY(0px); /* End at normal position */
  }
}

/* Style for the SnapMover component, covering the mover area for interaction */
.SnapMover,
.SnapResizer {
  position: absolute; /* Cover the entire area of parent */
```

```
  left: 0px;
  top: 0px;
  width: 100%;
  height: 100%;
}

File: src\components\stage-manager\components\Space\Space.tsx
--------------------

"use client";

import React, {
  HTMLAttributes,
  useCallback,
  useContext,
  useEffect,
  useMemo,
  useRef,
  useState,
} from "react";
import classNames from "classnames";

import { ManagerContext } from "../../contexts";
import { hashSpaceIds } from "../../space";
import {
  SpaceContext,
  SpaceEventDispatcher,
  SpaceWindow,
  SpaceWindows,
  ToSnap,
  WindowEvent,
  MoveEvent,
  ResizeEvent,
} from "./library";

import styles from "./Space.module.css";

const DEFAULT_AUTO_HIDE_STAGEDS: boolean = false;
const DEFAULT_STAGEDS_WIDTH: number = 150;
const DEFAULT_SNAP: boolean = true;
```

```
const DEFAULT_SNAP_MARGIN: number = 20;
const DEFAULT_SNAP_THRESHOLD: number = 50;
const DEFAULT_SNAP_WITH = "all";

type SnappingOrientation = "horizontal" | "vertical";

// Represents a window snapping connection between two windows
// and maintains the relationship between snapped windows
class Snapping {
  orientation: SnappingOrientation;
  interactedWindow: SpaceWindow;
  relatedWindows: SpaceWindows = new SpaceWindows();
  leftWindow: SpaceWindow;
  rightWindow: SpaceWindow;

  constructor(
    orientation: SnappingOrientation,
    interactedWindow: SpaceWindow,
    relatedWindows: SpaceWindows,
    leftWindow: SpaceWindow,
    rightWindow: SpaceWindow
  ) {
    this.orientation = orientation;
    this.interactedWindow = interactedWindow;
    this.leftWindow = leftWindow;
    this.rightWindow = rightWindow;
    this.relatedWindows.merge(relatedWindows);
  }

  // Checks if two snapping instances represent the same window connection
  equals(other: Snapping | null) {
    return (
      other &&
      this.orientation === other.orientation &&
      this.leftWindow.equals(other.leftWindow) &&
      this.rightWindow.equals(other.rightWindow)
    );
  }

  // Generates a unique hash for the related windows to track snapping state
```

```
  getRelatedWindowsHash(prefix: string = ""): string {
    return (
      prefix +
      (prefix ? "-" : "") +
      hashSpaceIds(Array.from(this.relatedWindows.all.keys()))
    );
  }
}

// Extends Snapping to track active snapping states and z-index for rendering
class Snap extends Snapping {
  snapMoving: boolean = false;
  snapResizing: boolean = false;
  zIndex: number = 0;
}

interface SpaceProps extends React.HTMLAttributes<HTMLDivElement> {
  children?: React.ReactNode;
  autoHideStageds?: boolean;
  stagedsWidth?: number;
  snap?: boolean;
  snapMargin?: number;
  snapThreshold?: number;
  snapResizer?: boolean;
  snapWith?: "all" | "move" | "resize";
}

// Main Space component that manages window layout, snapping, and interactions
export function Space({
  children = null,
  autoHideStageds = DEFAULT_AUTO_HIDE_STAGEDS,
  stagedsWidth = DEFAULT_STAGEDS_WIDTH,
  snap = DEFAULT_SNAP,
  snapMargin = DEFAULT_SNAP_MARGIN,
  snapThreshold = DEFAULT_SNAP_THRESHOLD,
  snapWith = DEFAULT_SNAP_WITH,
  ...attrs
}: SpaceProps) {
  const { size, setWheelBusy, scaleX, scaleY, revertScaleX, revertScaleY } =
    useContext(ManagerContext);
```

```
const [lmb, setLmb] = useState<boolean>(false);
const [pointer, setPointer] = useState<[number, number]>([0, 0]);
const windowsContainerRef = useRef<HTMLDivElement>(null);
const stagedsRef = useRef<HTMLDivElement>(null);
const [focusedWindow, setFocusedWindow] = useState<string | null>(null);
const [lastWindowPosition, setLastWindowPosition] = React.useState<
  [number, number]
>([0, 0]);
const [windowZIndexCounter, setWindowZIndexCounter] =
  React.useState<number>(1000);
const [showStageds, setShowStageds] = useState<boolean>(!autoHideStageds);
const [toSnap, setToSnap] = useState<ToSnap | null>(null);
const [snapping, setSnapping] = useState<Snapping | null>(null);
const [snaps, setSnaps] = useState<Snap[]>([]);
const [eventDispatcher] = useState(new SpaceEventDispatcher());
const [unmountedWindows, setUnmountedWindows] = useState<string[]>([]);
const snapMovingRef = useRef(false);
const snapResizingRef = useRef(false);

useEffect(() => setShowStageds(pointer[0] <= 150), [pointer]);

const snapsRef = useRef<Snap[]>(snaps);
useEffect(() => {
  snapsRef.current = snaps;
}, [snaps]);
const windowsRef = useRef<SpaceWindows>(new SpaceWindows());

// Updates snaps when windows are unmounted to prevent orphaned connections
useEffect(() => {
  if (unmountedWindows.length === 0) return;

  setSnaps(
    snapsRef.current.filter(
      (snap) => !unmountedWindows.some((id) => snap.relatedWindows.has(id))
    )
  );
  windowsRef.current = windowsRef.current.filter(
    (window) => !unmountedWindows.includes(window.id)
  );
```

```
  setUnmountedWindows([]);
}, [unmountedWindows]);

// Handles window position and size changes, managing snap connections
const onWindowBoundsChange = useCallback(
  (
    id: string,
    position: [number, number],
    size: [number, number],
    moving: boolean,
    resizing: boolean,
    staged: boolean
  ) => {
    // Skip snapping logic if snapping is disabled or window is staged
    if (!snap) {
      windowsRef.current.set(
        new SpaceWindow(id, position, size, moving, resizing, staged)
      );

      setToSnap(null);
      setSnapping(null);
      snapsRef.current = [];
      setSnaps([]);

      return;
    }

    if (!snapsRef.current || staged) {
      setToSnap(null);
      setSnapping(null);
      return;
    }

    windowsRef.current.set(
      new SpaceWindow(id, position, size, moving, resizing, staged)
    );

    setSnaps([...snapsRef.current]);

    // Find nearby windows within snapping threshold distance
```

```
const nearbyWindows = windowsRef.current.map((other) => {
  if (other.id == id) return;
  if (other.staged) return;

  const left_right_distance = Math.abs(
    other.position[0] + scaleX(other.size[0]) - position[0]
  );
  const top_distance = Math.abs(other.position[1] - position[1]);
  const bottom_distance = Math.abs(
    other.position[1] +
      scaleY(other.size[1]) -
      (position[1] + scaleY(size[1]))
  );
  if (
    left_right_distance <= snapThreshold &&
    top_distance <= snapThreshold &&
    bottom_distance <= snapThreshold
  ) {
    other.clockPosition = 0;
    return other;
  }

  const right_left_distance = Math.abs(
    other.position[0] - (position[0] + scaleX(size[0]))
  );
  if (
    right_left_distance <= snapThreshold &&
    top_distance <= snapThreshold &&
    bottom_distance <= snapThreshold
  ) {
    other.clockPosition = 1;
    return other;
  }
});

// Remove existing snaps if windows are no longer nearby
const existingSnaps = snapsRef.current.filter((snap) =>
  snap.relatedWindows.has(id)
);
for (const existingSnap of existingSnaps) {
```

```
  if (
    !nearbyWindows.has(existingSnap.leftWindow.id) &&
    !nearbyWindows.has(existingSnap.rightWindow.id)
  ) {
    snapsRef.current = snapsRef.current.filter(
      (snap) => !snap.equals(existingSnap)
    );
    setSnaps([...snapsRef.current]);
  }
}

// Apply snapping based on window positions and snap configuration
if (nearbyWindows.size == 0 || nearbyWindows.size > 2) {
  setToSnap(null);
  setSnapping(null);
  snapsRef.current = snapsRef.current.filter(
    (snap) => !snap.relatedWindows.has(id)
  );
  setSnaps([...snapsRef.current]);

  for (const nearbyWindow of Array.from(nearbyWindows.values())) {
    const existingSnap = snapsRef.current.find(
      (snap) =>
        snap.relatedWindows.has(id) &&
        snap.relatedWindows.has(nearbyWindow.id)
    );
    if (existingSnap) {
      snapsRef.current = snapsRef.current.filter(
        (snap) => !snap.equals(existingSnap)
      );
      setSnaps([...snapsRef.current]);
    }
  }

  return;
}

if (
  !(
    (snapWith == "all" && (moving || resizing)) ||
```

```
    (snapWith == "resize" && resizing) ||
    (snapWith == "move" && moving)
  )
)
  return;

for (const nearbyWindow of Array.from(nearbyWindows.values())) {
  const interactedWindow = windowsRef.current.get(id);
  if (!interactedWindow) {
    setToSnap(null);
    setSnapping(null);
    snapsRef.current = snapsRef.current.filter(
      (snap) => !snap.relatedWindows.has(id)
    );
    setSnaps([...snapsRef.current]);
    return;
  }

  const existingSnap = snapsRef.current.find(
    (snap) =>
      snap.relatedWindows.has(id) &&
      snap.relatedWindows.has(nearbyWindow.id)
  );
  if (
    existingSnap &&
    !existingSnap.snapMoving &&
    !existingSnap.snapResizing
  ) {
    snapsRef.current = snapsRef.current.filter(
      (snap) => !snap.equals(existingSnap)
    );
    setSnaps([...snapsRef.current]);
  }

  if (nearbyWindow.clockPosition == 0) {
    const relatedWindows = new SpaceWindows(
      new Map<string, SpaceWindow>([
        [nearbyWindow.id, nearbyWindow],
        [id, interactedWindow],
      ])
```

```
        );
        const newSnapping = new Snapping(
          "horizontal",
          interactedWindow,
          relatedWindows,
          nearbyWindow,
          windowsRef.current.get(id)!
        );
        if (!newSnapping.equals(snapping)) setSnapping(newSnapping);
      } else if (nearbyWindow.clockPosition == 1) {
        const relatedWindows = new SpaceWindows(
          new Map<string, SpaceWindow>([
            [nearbyWindow.id, nearbyWindow],
            [id, interactedWindow],
          ])
        );
        const newSnapping = new Snapping(
          "horizontal",
          interactedWindow,
          relatedWindows,
          windowsRef.current.get(id)!,
          nearbyWindow
        );
        if (!newSnapping.equals(snapping)) setSnapping(newSnapping);
      }
    }
  },
  [snap, snapThreshold, snapWith, snapping, scaleX, scaleY]
);

const onWindowMoveStart = useCallback(() => {}, []);
const onWindowMoveEnd = useCallback(() => {}, []);

// Finalizes window snapping when user finishes moving/resizing
const onUserBoundsChangeEnd = useCallback(
  (
    id: string,
    _position: [number, number],
    size: [number, number],
    moving: boolean,
```

```
  resizing: boolean,
  staged: boolean
) => {
  setSnapping(null);

  if (
    !(
      snapping &&
      ((snapWith == "all" && !resizing && !moving) ||
        (snapWith == "resize" && !resizing) ||
        (snapWith == "move" && !moving))
    )
  ) {
    if (staged) {
      snapsRef.current = snapsRef.current.filter(
        (snap) => !snap.relatedWindows.has(id)
      );
      setSnaps([...snapsRef.current]);
    }
    return;
  }

  const interactedWindow = windowsRef.current.get(id);
  if (!interactedWindow) return;

  const other = snapping.leftWindow.equals(interactedWindow)
    ? snapping.rightWindow
    : snapping.leftWindow;

  // Calculate new position and size for snapped windows
  if (snapping.orientation == "horizontal") {
    snapsRef.current = snapsRef.current.filter(
      (snap) => !snap.relatedWindows.has(id)
    );
    setSnaps([...snapsRef.current]);

    if (staged || other.staged) return;

    let new_position: [number, number];
    let new_size: [number, number];
```

```
if (other.clockPosition == 0) {
  new_position = [
    other.position[0] + scaleX(other.size[0]) + snapMargin,
    other.position[1],
  ];
  new_size = [scaleX(size[0]), scaleY(other.size[1])];
} else if (other.clockPosition == 1) {
  new_position = [
    other.position[0] - scaleX(size[0]) - snapMargin,
    other.position[1],
  ];
  new_size = [scaleX(size[0]), scaleY(other.size[1])];
} else {
  snapsRef.current = snapsRef.current.filter(
    (snap) =>
      !snap.relatedWindows.has(id) && !snap.relatedWindows.has(other.id)
  );
  setSnaps([...snapsRef.current]);
  return;
}

setToSnap(
  new ToSnap(
    snapping.interactedWindow,
    [snapping.leftWindow, snapping.rightWindow],
    new_position,
    [revertScaleX(new_size[0]), revertScaleY(new_size[1])]
  )
);

const snap = new Snap(
  "horizontal",
  snapping.interactedWindow,
  snapping.relatedWindows,
  snapping.leftWindow,
  snapping.rightWindow
);
snapsRef.current = [...snapsRef.current, snap];
setSnaps([...snapsRef.current]);
```

```
    }
  },
  [snapWith, snapping, snapMargin, scaleX, scaleY, revertScaleX, revertScaleY]
);

// Updates z-index for all windows in a snap group when focus changes
useEffect(() => {
  if (!focusedWindow) return;
  const snap = snapsRef.current.find((snap) =>
    snap.relatedWindows.has(focusedWindow)
  );
  if (!snap) return;
  snap.zIndex = windowZIndexCounter;
  snapsRef.current = [
    ...snapsRef.current.filter((otherSnap) => !snap.equals(otherSnap)),
    snap,
  ];
  setSnaps([...snapsRef.current]);
}, [focusedWindow, windowZIndexCounter]);

const contextProps = useMemo(
  () => ({
    lmb,
    pointer,
    setPointer,
    windowsRef: windowsContainerRef,
    stagedsRef,
    focusedWindow,
    setFocusedWindow,
    lastWindowPosition,
    setLastWindowPosition,
    windowZIndexCounter,
    setWindowZIndexCounter,
    stagedsWidth,
    snap,
    snapMargin,
    snapThreshold,
    toSnap,
    onWindowMoveStart,
    onWindowMoveEnd,
```

```
      onWindowBoundsChanged: onWindowBoundsChange,
      onUserBoundsChangeEnd,
      eventDispatcher,
      unmountedWindows,
      setUnmountedWindows,
    }),
    [
      lmb,
      pointer,
      setPointer,
      windowsContainerRef,
      stagedsRef,
      focusedWindow,
      setFocusedWindow,
      lastWindowPosition,
      setLastWindowPosition,
      windowZIndexCounter,
      setWindowZIndexCounter,
      stagedsWidth,
      snap,
      snapMargin,
      snapThreshold,
      toSnap,
      onWindowMoveStart,
      onWindowMoveEnd,
      onWindowBoundsChange,
      onUserBoundsChangeEnd,
      eventDispatcher,
      unmountedWindows,
      setUnmountedWindows,
    ]
  );

  const onMouseMove = useCallback((event: React.MouseEvent<HTMLDivElement>)
=> {
    const rect = event.currentTarget.getBoundingClientRect();
    const new_pointer: [number, number] = [
      event.clientX - rect.x,
      event.clientY - rect.y,
    ];
```

```
    setPointer(new_pointer);
  }, []);

  const onTouchMove = useCallback((event: React.TouchEvent<HTMLDivElement>) =>
{
    if (event.touches.length !== 1) return;
    const touch = event.touches[0];
    const rect = event.currentTarget.getBoundingClientRect();
    const new_pointer: [number, number] = [
      touch.clientX - rect.x,
      touch.clientY - rect.y,
    ];
    setPointer(new_pointer);
  }, []);

  return (
    <SpaceContext.Provider value={contextProps}>
      <div
        {...(attrs as HTMLAttributes<HTMLDivElement>)}
        className={`${classNames([
          styles.Space,
          { [styles.Space__autoHideStageds]: autoHideStageds },
          { [styles.Space__showStageds]: showStageds },
        ])} ${typeof attrs.className !== "undefined" ? attrs.className : ""}`}
        style={{
          minWidth: size[0],
          height: size[1],
          ...attrs.style,
        }}
        onMouseMove={onMouseMove}
        onTouchMove={onTouchMove}
        onMouseDown={() => setLmb(true)}
        onMouseUp={() => setLmb(false)}
        onTouchStart={() => setLmb(true)}
        onTouchEnd={() => setLmb(false)}
      >
        {children}
        <div
          ref={stagedsRef}
          className={classNames([styles.Space_stageds])}
```

```
  style={{ width: stagedsWidth }}
  onMouseOver={() => setWheelBusy(true)}
  onMouseLeave={() => setWheelBusy(false)}
  onWheel={(event) => event.stopPropagation()}
></div>
<div
  ref={windowsContainerRef}
  className={classNames([styles.Space_windows])}
></div>
<div
  className={classNames([
    styles.Space_snap,
    { [styles.Space_snap__show]: snapping },
  ])}
  style={{
    ...(() =>
      !snapping
        ? {}
        : {
            left:
              snapping.interactedWindow.id == snapping.rightWindow.id
                ? undefined
                : revertScaleX(
                    snapping.leftWindow.position[0] +
                      scaleX(snapping.leftWindow.size[0])
                  ),
            right:
              snapping.interactedWindow.id == snapping.leftWindow.id
                ? undefined
                : revertScaleY(
                    scaleX(size[0]) - snapping.rightWindow.position[0]
                  ),
            top: revertScaleY(snapping.leftWindow.position[1]),
            height: snapping.leftWindow.size[1],
            zIndex: windowZIndexCounter + 1,
            width: revertScaleX(
              (() => {
                const leftWindowRight =
                  snapping.leftWindow.position[0] +
                  scaleX(snapping.leftWindow.size[0]);
```

```
              const rightWindowLeft =
                snapping.rightWindow.position[0];
              return Math.abs(leftWindowRight - rightWindowLeft);
            })()
          ),
        })(),
  }}
>
  <div className={styles.Space_snap_resizer}></div>
</div>
{snaps.map((snap, _index) => (
  <div
    key={snap.getRelatedWindowsHash()}
    className={classNames([
      styles.Space_snap,
      styles.Space_snap__snapped,
    ])}
    style={{
      ...(() => ({
        left: revertScaleX(
          snap.leftWindow.position[0] + scaleX(snap.leftWindow.size[0])
        ),
        top: revertScaleY(snap.leftWindow.position[1]),
        height: snap.leftWindow.size[1],
        width: revertScaleX(
          (() => {
            const leftWindowRight =
              snap.leftWindow.position[0] +
              scaleX(snap.leftWindow.size[0]);
            const rightWindowLeft = snap.rightWindow.position[0];
            return Math.abs(leftWindowRight - rightWindowLeft);
          })()
        ),
        zIndex: snap.zIndex,
      }))(),
    }}
  >
    <div
      className={styles.Space_snap_mover}
      key={snap.getRelatedWindowsHash("mover")}
```

```
>
  <SnapMover
    key={snap.getRelatedWindowsHash()}
    onMove={(positionDelta) => {
      snap.relatedWindows.all.forEach((window) =>
        eventDispatcher.dispatch("move", {
          target: window,
          positionDelta,
        } as MoveEvent)
      );
    }}
    onMoveStart={() => {
      snapMovingRef.current = true;
      snap.snapMoving = true;
      snap.relatedWindows.all.forEach((window) => {
        eventDispatcher.dispatch("move-start", {
          target: window,
        } as WindowEvent);
        window.moving = true;
      });
    }}
    onMoveEnd={() => {
      snapMovingRef.current = false;
      snap.snapMoving = false;
      snap.relatedWindows.all.forEach((window) => {
        eventDispatcher.dispatch("move-end", {
          target: window,
        } as WindowEvent);
        window.moving = false;
      });
    }}
  />
</div>
<div
  className={styles.Space_snap_resizer}
  key={snap.getRelatedWindowsHash("resizer")}
>
  <SnapResizer
    key={snap.getRelatedWindowsHash()}
    onResize={(sizeDelta) => {
```

```
                eventDispatcher.dispatch("resize", {
                  target: snap.leftWindow,
                  sizeDelta: [revertScaleX(sizeDelta[0]), 0],
                } as ResizeEvent);
                eventDispatcher.dispatch("resize", {
                  target: snap.rightWindow,
                  sizeDelta: [-revertScaleX(sizeDelta[0]), 0],
                } as ResizeEvent);
                eventDispatcher.dispatch("move", {
                  target: snap.rightWindow,
                  positionDelta: [sizeDelta[0], 0],
                } as MoveEvent);
              }}
              onResizeStart={() => {
                snapResizingRef.current = true;
                snap.snapResizing = true;
                snap.relatedWindows.all.forEach((window) => {
                  window.resizing = true;
                });
              }}
              onResizeEnd={() => {
                snapResizingRef.current = false;
                snap.snapResizing = false;
                snap.relatedWindows.all.forEach((window) => {
                  window.resizing = false;
                });
              }}
            />
          </div>
        </div>
      ))}
    </div>
  </SpaceContext.Provider>
 );
}

// Handles dragging of snapped window groups together
interface SnapMoverProps {
 onMove: (positionDelta: [number, number]) => void;
 onMoveStart?: () => void;
```

```
  onMoveEnd?: () => void;
}

function SnapMover({
  onMove = () => {},
  onMoveStart = () => {},
  onMoveEnd = () => {},
}: SnapMoverProps) {
  const { pointer } = useContext(ManagerContext);
  const { lmb } = useContext(SpaceContext);
  const [dragging, setDragging] = useState<boolean>(false);
  const onMoveRef = useRef(onMove);
  useEffect(() => {
    onMoveRef.current = onMove;
  }, [onMove]);
  const prevDragPositionRef = useRef<[number, number]>([0, 0]);

  useEffect(() => {
    if (!lmb) setDragging(false);
  }, [lmb]);

  // Updates window positions while dragging snap group
  useEffect(() => {
    if (!dragging) return;
    const positionDelta: [number, number] = [
      pointer[0] - prevDragPositionRef.current[0],
      pointer[1] - prevDragPositionRef.current[1],
    ];
    onMoveRef.current && onMoveRef.current(positionDelta);
    prevDragPositionRef.current = pointer;
  }, [dragging, pointer]);

  useEffect(
    () => (dragging ? onMoveStart() : onMoveEnd()),
    [dragging, onMoveStart, onMoveEnd]
  );

  const onMouseDown = useCallback(() => {
    setDragging(true);
    prevDragPositionRef.current = [pointer[0], pointer[1]];
```

```
  }, [pointer]);

  const onMouseUp = useCallback(() => {
    setDragging(false);
  }, []);

  return (
    <div
      className={classNames([
        styles.SnapMover,
        { [styles.SnapMover__dragging]: dragging },
      ])}
      onMouseDown={onMouseDown}
      onMouseUp={onMouseUp}
    ></div>
  );
}

// Handles resizing between snapped windows
interface SnapResizerProps {
  onResize: (sizeDelta: [number, number]) => void;
  onResizeStart?: () => void;
  onResizeEnd?: () => void;
}

function SnapResizer({
  onResize,
  onResizeStart = () => {},
  onResizeEnd = () => {},
}: SnapResizerProps) {
  const { pointer } = useContext(ManagerContext);
  const { lmb } = useContext(SpaceContext);

  const [dragging, setDragging] = useState<boolean>(false);

  const onResizeRef = useRef(onResize);
  useEffect(() => {
    onResizeRef.current = onResize;
  }, [onResize]);
  const prevDragPositionRef = useRef<[number, number]>([0, 0]);
```

```
  useEffect(() => {
    if (!lmb) setDragging(false);
  }, [lmb]);

  // Updates window sizes while resizing snap connection
  useEffect(() => {
    if (!dragging) return;
    const positionDelta: [number, number] = [
      pointer[0] - prevDragPositionRef.current[0],
      pointer[1] - prevDragPositionRef.current[1],
    ];
    onResizeRef.current &&
      onResizeRef.current([positionDelta[0], positionDelta[1]]);
    prevDragPositionRef.current = pointer;
  }, [dragging, pointer]);

  useEffect(
    () => (dragging ? onResizeStart() : onResizeEnd()),
    [dragging, onResizeStart, onResizeEnd]
  );

  const onMouseDown = useCallback(() => {
    setDragging(true);
    prevDragPositionRef.current = [pointer[0], pointer[1]];
  }, [pointer]);

  const onMouseUp = useCallback(() => {
    setDragging(false);
  }, []);

  return (
    <div
      className={styles.SnapResizer}
      onMouseDown={onMouseDown}
      onMouseUp={onMouseUp}
    ></div>
  );
}
```

File: src\components\stage-manager\components\Space\Spaces.tsx
---

```
"use client";

import React, { useEffect, useContext, HTMLAttributes, useState } from "react";
import classNames from "classnames";
import { ManagerContext } from "../../contexts";
import { Overlay } from "./Overlay";

interface SpacesProps extends React.HTMLAttributes<HTMLDivElement> {
  children?: React.ReactNode;
  bounceDelay?: number;
  scrollThreshold?: number;
  swipeThreshold?: number;
  space: number;
  onSpaceChange: (space: number) => void;
}

/**
 * @component Spaces
 * @description Manages and renders multiple Space components within the Manager.
 *              Handles space switching, keyboard navigation, and overlay for space
selection.
 */
function Spaces({
  children = null,
  space = 0,
  onSpaceChange = () => {},
  ...attrs
}: SpacesProps) {
  const { size } = useContext(ManagerContext);
  const totalSpaces = React.Children.count(children);
  // State to control the visibility of the space selection overlay
  const [overlayVisible, setOverlayVisible] = useState(false);
  // State to track the currently highlighted space in the overlay
  const [highlightedSpace, setHighlightedSpace] = useState(space);
  // State to track if the 'z' key is pressed for overlay activation
  const [zKeyPressed, setZKeyPressed] = useState(false);
```

```
  /**
   * @useEffect Keyboard Navigation for Space Switching
   * @description Enables Ctrl + ArrowLeft/Right for navigating between spaces.
   */
  useEffect(() => {
    const handleKeyDown = (e: KeyboardEvent) => {
      // Check if Ctrl key is pressed
      if (e.ctrlKey) {
        e.preventDefault(); // Prevent default browser behavior with Ctrl+Arrow
        switch (e.key) {
          case "ArrowLeft":
            // Navigate to the previous space, ensuring it doesn't go below 0
            onSpaceChange(Math.max(0, space - 1));
            break;
          case "ArrowRight":
            // Navigate to the next space, ensuring it doesn't exceed total spaces
            onSpaceChange(Math.min(totalSpaces - 1, space + 1));
            break;
        }
      }
    };

    window.addEventListener("keydown", handleKeyDown);
    return () => window.removeEventListener("keydown", handleKeyDown);
  }, [totalSpaces, space, onSpaceChange]);

  /**
   * @useEffect Overlay and 'z' Key Handling for Space Selection
   * @description Controls the space selection overlay using 'z' key for activation and 'q'
for cycling.
   */
  useEffect(() => {
    const handleKeyDown = (e: KeyboardEvent) => {
      // Activate overlay on 'z' key press (if not already active)
      if (e.key === "z" && !zKeyPressed) {
        setZKeyPressed(true);
        setOverlayVisible(true); // Show overlay
        setHighlightedSpace(space); // Start with current space
      }
```

```
      // Cycle through spaces on 'q' key press (only if overlay is active)
      if (e.key === "q" && zKeyPressed) {
        e.preventDefault(); // Prevent browser's default tab behavior
        setHighlightedSpace((prev) => (prev + 1) % totalSpaces); // Cycle through spaces
      }
    };

    const handleKeyUp = (e: KeyboardEvent) => {
      // Deactivate overlay on 'z' key release
      if (e.key === "z") {
        setZKeyPressed(false);
        setOverlayVisible(false); // Hide overlay
        onSpaceChange(highlightedSpace); // Activate highlighted space
      }
    };

    window.addEventListener("keydown", handleKeyDown);
    window.addEventListener("keyup", handleKeyUp);

    return () => {
      window.removeEventListener("keydown", handleKeyDown);
      window.removeEventListener("keyup", handleKeyUp);
    };
  }, [space, totalSpaces, zKeyPressed, onSpaceChange, highlightedSpace]);

  /**
   * @useEffect Space Range Validation
   * @description Ensures the 'space' prop stays within the valid range of available
spaces.
   */
  useEffect(() => {
    // Keep space index within valid bounds [0, totalSpaces - 1]
    onSpaceChange(Math.max(0, Math.min(space, totalSpaces - 1)));
  }, [space, totalSpaces, onSpaceChange]);

  return (
    <div
      {...(attrs as HTMLAttributes<HTMLDivElement>)}
      className={classNames("overflow-hidden relative", attrs.className)}
```

```
      style={{ width: size[0], height: size[1], ...attrs.style }}
    >
      <>
        {/* Overlay Component */}
        {/* Render the overlay component conditionally based on overlay visibility state */}
        {overlayVisible && (
          <Overlay
            spaces={totalSpaces}
            activeSpace={highlightedSpace}
            onSpaceHover={setHighlightedSpace}
          />
        )}

        {/* Existing Spaces Layout */}
        {/* Container for rendering Space components, handles horizontal scrolling for
space switching */}
        <div
          className="flex flex-nowrap absolute left-0 top-0 h-full"
          style={{
            // Translate the container horizontally to show the currently active space
            transform: `translateX(${space * size[0] * -1}px)`,
            // Add a smooth transition for space switching animation
            transition: "transform 100ms ease-out",
          }}
        >
          {children}
        </div>
      </>
    </div>
  );
}

export { Spaces };


File: src\components\stage-manager\components\Space\index.ts
--------------------

export { Spaces } from "./Spaces";
export { Space } from "./Space";
```

```
export { SpaceContext } from "./library";


File: src\components\stage-manager\components\Space\library.ts
----------

import { createContext } from "react";

import { EventDispatcher } from "../../space";

export class SpaceWindow {
  id: string;
  position: [number, number];
  size: [number, number];
  moving?: boolean;
  resizing?: boolean;
  staged?: boolean;
  clockPosition?: number;

  constructor(
    id: string,
    position: [number, number],
    size: [number, number],
    moving?: boolean,
    resizing?: boolean,
    staged?: boolean,
    clockPosition?: number
  ) {
    this.id = id;
    this.position = position;
    this.size = size;
    this.moving = moving;
    this.resizing = resizing;
    this.staged = staged;
    this.clockPosition = clockPosition;
  }

  equals(other: SpaceWindow | null) {
    return (
      other &&
```

```
      this.id === other.id &&
      this.position[0] === other.position[0] &&
      this.position[1] === other.position[1] &&
      this.size[0] === other.size[0] &&
      this.size[1] === other.size[1] &&
      this.moving === other.moving &&
      this.resizing === other.resizing
    );
  }
}

export class SpaceWindows {
  private windows: Map<string, SpaceWindow> = new Map();

  constructor(windows?: Map<string, SpaceWindow>) {
    if (windows) windows.forEach((value, key) => this.windows.set(key, value));
  }

  get all() {
    return this.windows;
  }
  copy() {
    return new SpaceWindows(this.windows);
  }

  set(window: SpaceWindow) {
    const existing = this.windows.get(window.id);
    if (!existing) {
      this.windows.set(window.id, window);
      return this;
    }
    existing.position = window.position;
    existing.size = window.size;
    existing.clockPosition = window.clockPosition;
    return this;
  }

  remove(id: string) {
    this.windows.delete(id);
    return this;
```

```
  }

  get(id: string): SpaceWindow | undefined {
    return this.windows.get(id);
  }

  merge(other: SpaceWindows) {
    other.windows.forEach((value, _key) => this.set(value));
  }

  has(id: string): boolean {
    return this.windows.has(id);
  }

  exclude(windows: SpaceWindows) {
    return this.filter((window) => !windows.windows.has(window.id));
  }

  map(
    fn: (window: SpaceWindow) => SpaceWindow | undefined
  ): Map<string, SpaceWindow> {
    const result = new Map<string, SpaceWindow>();

    this.windows.forEach((value, key) => {
      const opt = fn(value);
      if (opt !== undefined) result.set(key, opt);
    });

    return result;
  }

  filter(fn: (window: SpaceWindow) => boolean): SpaceWindows {
    const result = new SpaceWindows();
    this.windows.forEach((value, _key) => {
      if (fn(value)) result.set(value);
    });
    return result;
  }
}
```

```
export type ToSnapWindows =
  | null
  | [SpaceWindow, SpaceWindow]
  | [SpaceWindow, SpaceWindow, SpaceWindow]
  | [SpaceWindow, SpaceWindow, SpaceWindow, SpaceWindow];

export class ToSnap {
  target: SpaceWindow;
  windows: ToSnapWindows;
  newPosition?: [number, number];
  newSize?: [number, number];

  constructor(
    target: SpaceWindow,
    windows: ToSnapWindows,
    newPosition?: [number, number],
    newSize?: [number, number]
  ) {
    this.target = target;
    this.windows = windows;
    this.newPosition = newPosition;
    this.newSize = newSize;
  }

  getCurrentAndOthers(
    spaceId: string
  ): [SpaceWindow, SpaceWindow[]] | undefined {
    if (this.windows === null) return undefined;

    const current = this.windows.find((window) => window.id === spaceId);
    if (current === undefined) return undefined;

    const others = this.windows.filter((window) => window.id !== spaceId);
    return [current, others];
  }
}

export interface WindowEvent {
  target: SpaceWindow;
}
```

```
export interface MoveEvent extends WindowEvent {
  positionDelta: [number, number];
}

export interface ResizeEvent extends WindowEvent {
  sizeDelta: [number, number];
}

export type SpaceWindowBoundsUpdateEventCallback = (
  id: string,
  position: [number, number],
  size: [number, number],
  moving: boolean,
  resizing: boolean,
  staged: boolean
) => void;

export interface SpaceContextProps {
  lmb: boolean;
  pointer: [number, number];
  setPointer: React.Dispatch<React.SetStateAction<[number, number]>>;
  windowsRef: React.RefObject<HTMLDivElement>;
  stagedsWidth: number;
  focusedWindow: string | null;
  setFocusedWindow: React.Dispatch<React.SetStateAction<string | null>>;
  windowZIndexCounter: number;
  setWindowZIndexCounter: React.Dispatch<React.SetStateAction<number>>;
  stagedsRef: React.RefObject<HTMLDivElement>;
  lastWindowPosition: [number, number];
  setLastWindowPosition: React.Dispatch<React.SetStateAction<[number, number]>>;
  snap: boolean;
  snapMargin: number;
  snapThreshold: number;
  toSnap: ToSnap | null;
  eventDispatcher: SpaceEventDispatcher | null;
  unmountedWindows: string[];
  setUnmountedWindows: React.Dispatch<React.SetStateAction<string[]>>;
  onWindowMoveStart: SpaceWindowBoundsUpdateEventCallback;
  onWindowMoveEnd: SpaceWindowBoundsUpdateEventCallback;
```

```
  onWindowBoundsChanged: SpaceWindowBoundsUpdateEventCallback;
  onUserBoundsChangeEnd: SpaceWindowBoundsUpdateEventCallback;
}

export const SpaceContext = createContext<SpaceContextProps>({
  lmb: false,
  pointer: [0, 0],
  setPointer: () => {},
  windowsRef: { current: null },
  stagedsWidth: 150,
  focusedWindow: null,
  setFocusedWindow: () => {},
  windowZIndexCounter: 0,
  setWindowZIndexCounter: () => {},
  stagedsRef: { current: null },
  lastWindowPosition: [0, 0],
  setLastWindowPosition: () => {},
  snap: false,
  snapMargin: 0,
  snapThreshold: 0,
  toSnap: null,
  eventDispatcher: null,
  unmountedWindows: [],
  setUnmountedWindows: () => {},
  onWindowMoveStart: () => {},
  onWindowMoveEnd: () => {},
  onWindowBoundsChanged: () => {},
  onUserBoundsChangeEnd: () => {},
});

export type SpaceEventName = "move-start" | "move-end" | "move" | "resize";
export type SpaceEventType = WindowEvent | MoveEvent | ResizeEvent;
export type SpaceEvent<T = SpaceEventType> = T;
export class SpaceEventDispatcher extends EventDispatcher<
  SpaceEventName,
  SpaceEvent<SpaceEventType>
> {}


==========
```

Directory: src\components\stage-manager\components\Window
==========

File: src\components\stage-manager\components\Window\Window.module.css
---

```
/* Base window container with smooth animations and modern styling */
.Window {
  position: absolute;
  left: 0px;
  top: 0px;
  width: 500px;
  height: 400px;
  background-color: rgb(255, 255, 255);
  border-radius: 8px;
  box-shadow: 0 1px 3px 0 rgba(0, 0, 0, 0.1), 0 1px 2px -1px rgba(0, 0, 0, 0.1);
  border: 1px solid #e5e7eb;
  display: flex;
  flex-direction: column;
  animation: AppearWindow 0.3s cubic-bezier(0.4, 0, 0.2, 1);
  box-sizing: border-box;
  transition: transform 0.2s ease, opacity 0.2s ease;
}

/* Special states for window interactions */
.Window.Window__moving {
}

/* Transition state while window is being staged */
.Window.Window__staging {
  transition: transform 0.25s, width 0.5s, height 0.5s, opacity 0.75s;
}

/* Enhanced visual feedback for the currently focused window */
.Window.Window__focused {
  box-shadow: 0px 0px 12px 0px rgba(0, 0, 0, 0.2),
    0px 0px 0px 1px rgba(0, 0, 0, 0.2);
}

/* Staged window appearance with 3D transform effects */
```

```
.Window.Window__staged {
  animation: AppearWindow 0.5s forwards;
  position: static;
  transition: all 0.5s !important;
  z-index: 100;
}
.Window.Window__snapMoving {
}

/* Overlay layer for staged windows to prevent unwanted interactions */
.Window_stagedLayer {
  position: absolute;
  left: 0px;
  top: 0px;
  width: 100%;
  height: 100%;
  z-index: 100000;
  content: " ";
  display: none;
  user-select: none;
  -webkit-user-select: none;
  -moz-user-select: none;
}
.Window.Window__staged .Window_stagedLayer {
  display: flex;
}

/* Interactive 3D effects for staged windows with hover states */
.Window_stagedWindow {
  position: relative;
  transition: all 0.5s;
  transform: rotateY(20deg);
  margin-bottom: 25px;
  z-index: 100;
  overflow: visible;
}

/* Enhanced hover effect with zoom and perspective adjustment */
.Window_stagedWindow:hover {
  transform: rotateY(0) translateX(10px);
```

```
  scale: 1.2;
  z-index: 200;
}

/* Animation keyframes for window transitions */
@keyframes AppearWindow {
  0% {
    opacity: 0;
    scale: 0.25;
  }
  100% {
    opacity: 1;
    scale: 1;
  }
}

@keyframes DisappearWindow {
  0% {
    opacity: 1;
    scale: 1;
  }
  100% {
    opacity: 0;
    scale: 0.25;
  }
}

/* Title bar styling with controls and draggable area */
.TitleBar {
  width: 100%;
  height: 30px;
  background: whitesmoke;
  border-top-left-radius: 5px;
  border-top-right-radius: 5px;
  display: flex;
  justify-content: space-between;
  align-items: center;
  text-align: center;
  border-bottom: rgb(204, 204, 204) 1px solid;
  user-select: none;
```

```
  box-sizing: border-box;
  position: relative;
}

/* Prevent text selection during window interactions */
.TitleBar,
.TitleBar * {
  cursor: default;
}
.TitleBar__dragging {
}

.TitleBar_stagingLayer {
  position: absolute;
  left: 0px;
  top: 0px;
  width: 100%;
  height: 100%;
  z-index: 100000;
  content: " ";
  display: none;
  user-select: none;
  -webkit-user-select: none;
  -moz-user-select: none;
}
.Window.Window.Window__staging .TitleBar_stagingLayer {
  display: flex;
}

/* Window control buttons (close, minimize, maximize) */
.TitleBar .Buttons {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-right: 5px;
}

/* Individual button styling with hover animations */
.TitleBar .Buttons .Button {
  width: 15px;
```

```
  height: 15px;
  border-radius: 100%;
  margin-left: 5px;
  font-size: 9px;
  color: white;
  line-height: 15px;
  text-align: center;
  transition: all 0.25s;
  border: 1px solid rgba(0, 0, 0, 0.125);
}

.TitleBar .Buttons .Button:hover {
  animation: Bubble 0.25s forwards;
  box-shadow: 0px 0px 0px 1px rgba(255, 255, 255, 0.5);
}

.Title {
  font-size: 12px;
  font-weight: 700;
  color: rgba(0, 0, 0, 0.5);
  padding: 10px 0px;
  padding-right: 10px;
  box-sizing: border-box;
  flex-grow: 1;
  white-space: nowrap;
  overflow: hidden;
  margin-right: 5px;
}

.CloseButton {
  background-color: #ef4444;
  color: white !important;
}

.StageButton {
  background-color: #ffc260;
}

/* Content area styling with placeholder support */
.Content {
```

```
  flex-grow: 1;
  position: relative;
  display: flex;
  align-items: center;
  justify-content: center;
}

/* Add this near the Content styles */
.Content [contenteditable].empty-content:before {
  content: attr(data-placeholder);
  color: rgba(0, 0, 0, 0.3);
  font-style: italic;
  cursor: text;
  position: absolute;
}

/* Fluid content layout for scrollable areas */
.Content_fluid {
  position: absolute;
  left: 0px;
  top: 0px;
  width: 100%;
  height: 100%;
  overflow: auto;
  display: flex;
  flex-direction: row;
  align-items: top;
  justify-content: start;
}

.Content_unfocusedLayer {
  position: absolute;
  left: 0px;
  top: 0px;
  width: 100%;
  height: 100%;
  z-index: 100000;
  content: " ";
  user-select: none;
  -webkit-user-select: none;
```

```
  -moz-user-select: none;
  overflow: hidden;
  box-sizing: border-box;
}
.Window.Window__focused .Content_unfocusedLayer {
  display: none;
}

.Resizers {
}

/* Window resize handles positioning and styling */
.Resizer {
  position: absolute;
  width: 10px;
  height: 10px;
  background-color: rgba(0, 0, 0, 0.1);
  border-radius: 50%;
  pointer-events: none;
  opacity: 0;
  animation: Disappear 0.5s forwards;
}

/* Directional resize handles with appropriate cursors */
.Resizer__bottom {
  bottom: -5px;
  left: 50%;
  transform: translateX(-50%);
  cursor: ns-resize;
}

.Resizer__top {
  top: -5px;
  left: 50%;
  transform: translateX(-50%);
  cursor: ns-resize;
}

.Resizer__left {
  top: 50%;
```

```
  left: -5px;
  transform: translateY(-50%);
  cursor: ew-resize;
}

.Resizer__right {
  top: 50%;
  right: -5px;
  transform: translateY(-50%);
  cursor: ew-resize;
}

.Resizer__bottomLeft {
  bottom: -5px;
  left: -5px;
  cursor: nesw-resize;
}

.Resizer__bottomRight {
  bottom: -5px;
  right: -5px;
  cursor: nwse-resize;
}

.Resizer__topLeft {
  top: -5px;
  left: -5px;
  cursor: nwse-resize;
}

.Resizer__topRight {
  top: -5px;
  right: -5px;
  cursor: nesw-resize;
}

/* Animation for resize handles appearance/disappearance */
.Window.Window__mayResize .Resizer,
.Window.Window__showResizers .Resizer {
  animation: AppearResizer 0.5s forwards;
```

```
  pointer-events: all;
  transition: scale 0.25s;
  transition-timing-function: cubic-bezier(0.075, 0.82, 0.165, 1);
}
.Window.Window__mayResize .Resizer:hover,
.Window.Window__showResizers .Resizer:hover {
  scale: 1.25 !important;
}

.Window.Window__mayResize,
.Window.Window__mayResize * {
  user-select: none;
}

/* Interactive animations */
@keyframes Bubble {
  0% {
    scale: 1;
  }
  50% {
    scale: 1.25;
  }
  100% {
    scale: 1;
  }
}

@keyframes AppearResizer {
  0% {
    opacity: 0;
    scale: 0;
  }
  50% {
    opacity: 1;
    scale: 1.25;
  }
  100% {
    opacity: 1;
    scale: 1;
  }
```

```
}

@keyframes Disappear {
  0% {
    opacity: 1;
    scale: 1;
  }
  50% {
    opacity: 1;
    scale: 1.25;
  }
  100% {
    opacity: 0;
    scale: 0;
  }
}
```

File: src\components\stage-manager\components\Window\Window.tsx

---

```
"use client";

import React, {
  useEffect,
  useState,
  useCallback,
  useContext,
  useRef,
  useMemo,
  HTMLAttributes,
} from "react";
import { createPortal } from "react-dom";
import classNames from "classnames";
import { isMobileDevice, nonZeroPosition } from "../../space";
import { usePosition, useSpaceId } from "../../hooks";
import { ManagerContext, SpaceContext } from "../../contexts";
import {
  WindowEvent,
  MoveEvent,
```

```
  ResizeEvent,
  SpaceEvent,
} from "../Space/library";
import { ALWAYS_ON_TOP_Z_INDEX, WindowContext } from "./library";

import styles from "./Window.module.css";

type ResizerDirection =
  | "top"
  | "right"
  | "bottom"
  | "left"
  | "top-right"
  | "top-left"
  | "bottom-right"
  | "bottom-left";
type OnMayResize = (may_resize: boolean) => void;
type BoundsChangeReason = "user" | "system";

const DEFAULT_MIN_SIZE: [number, number] = [100, 100];
const DEFAULT_MAX_SIZE = null;
const DEFAULT_RESIZABLE: boolean = true;
const DEFAULT_RESIZER_THRESHOLD: number = 25;
const DEFAUULT_STAGED: boolean = false;
const DEFAULT_STAGING_DISTANCE: number = 150;
const DEFAULT_STAGED_SIZE: [number, number] = [100, 120];
const DEFAULT_ALLOW_OUTSIDE: boolean = true;
const DEFAULT_COMPENSATE_POSITION_ON_VIEWPORT_RESIZE: boolean =
true;
const DEFAULT_CALLBACK = () => {};

interface WindowProps extends React.HTMLAttributes<HTMLDivElement> {
  children?: React.ReactNode;
  spaceId: string;
  size: [number, number];
  position: [number, number];
  minSize?: [number, number] | null;
  maxSize?: [number, number] | null;
  staged?: boolean;
  resizable?: boolean;
```

```
  resizerThreshold?: number;
  alwaysOnTop?: boolean;
  stagingDistance?: number;
  stagedSize?: [number, number] | [number, null] | [null, number];
  allowOutside?: boolean;
  compensatePositionOnViewportResize?: boolean;
  onStagedChange: (staged: boolean) => void;
  onSizeChange: (size: [number, number], reason: BoundsChangeReason) => void;
  onPositionChange: (
    position: [number, number],
    reason: BoundsChangeReason
  ) => void;
  onMoveStart?: () => void;
  onMoveEnd?: () => void;
  onFocus?: () => void;
  onBlur?: () => void;
}

// Base component that provides window functionality with dragging, resizing, and
staging capabilities
function Window({
  children = null,
  spaceId,
  size,
  position,
  minSize = DEFAULT_MIN_SIZE,
  maxSize = DEFAULT_MAX_SIZE,
  resizable = DEFAULT_RESIZABLE,
  resizerThreshold = DEFAULT_RESIZER_THRESHOLD,
  alwaysOnTop = false,
  staged = DEFAUULT_STAGED,
  stagingDistance = DEFAULT_STAGING_DISTANCE,
  stagedSize = DEFAULT_STAGED_SIZE,
  allowOutside = DEFAULT_ALLOW_OUTSIDE,
  compensatePositionOnViewportResize =
DEFAULT_COMPENSATE_POSITION_ON_VIEWPORT_RESIZE,
  onSizeChange = DEFAULT_CALLBACK,
  onStagedChange = DEFAULT_CALLBACK,
  onPositionChange = DEFAULT_CALLBACK,
  onMoveStart = DEFAULT_CALLBACK,
```

```
  onMoveEnd = DEFAULT_CALLBACK,
  onFocus = DEFAULT_CALLBACK,
  onBlur = DEFAULT_CALLBACK,
  ...attrs
}: WindowProps) {
  const {
    size: managerSize,
    setWheelBusy,
    scaleX,
    scaleY,
    revertScaleX,
    revertScaleY,
  } = useContext(ManagerContext);
  const {
    lmb,
    pointer,
    windowsRef,
    stagedsRef,
    focusedWindow,
    setFocusedWindow,
    setLastWindowPosition: setSpaceLastWindowPosition,
    windowZIndexCounter,
    setWindowZIndexCounter,
    stagedsWidth,
    onWindowMoveStart,
    onWindowMoveEnd,
    onUserBoundsChangeEnd,
    onWindowBoundsChanged,
    snapMargin,
    toSnap,
    eventDispatcher: spaceEventDispatcher,
    unmountedWindows: spaceUnmountedWindows,
    setUnmountedWindows: setSpaceUnmountedWindows,
  } = useContext(SpaceContext);
  const [showResizers, setShowResizers] = useState(false);
  const resizerMouseMoveTimeoutRef = useRef<ReturnType<typeof setTimeout>>();
  const [mayResize, setMayResize] = useState(false);
  const [moving, setMoving] = useState(false);
  const [resizing, setResizing] = useState(false);
  const [prevMoving, setPrevMoving] = useState(false);
```

```
const [zIndex, setZIndex] = useState(0);
const [focused, setFocused] = useState(true);
const [rotation, setRotation] = useState(0);
const [scale, setScale] = useState(1);
const [staging, setStaging] = useState(false);
const [stagingXCompenstation, setStagingXCompenstation] = useState(0);
const [stagingYCompenstation, setStagingYCompenstation] = useState(0);
const [stagedScale, setStagedScale] = useState([1, 1]);
const [scaledStagedSize, setScaledStagedSize] = useState([0, 0]);
const [movingStartPosition, setMovingStartPosition] =
  useState<[number, number]>(position);
const [prevManagerSize, setPrevManagerSize] = useState(managerSize);
const prevAlwaysOnTopRef = useRef<boolean>(false);
const hideResizersTimeoutRef = useRef<ReturnType<typeof setTimeout>>();
const prevResizingRef = useRef(resizing);
const [lastWindowPosition, setLastWindowPosition] =
  useState<[number, number]>(position);
const [stagedBy, setStagedBy] = useState<"instant" | "move">("instant");
const [snapMoving, setSnapMoving] = useState(false);

const spaceIdRef = useRef(spaceId);
useEffect(() => {
  spaceIdRef.current = spaceId;
}, [spaceId]);
const spaceUnmountedWindowsRef = useRef(spaceUnmountedWindows);
const setSpaceUnmountedWindowsRef = useRef(setSpaceUnmountedWindows);
useEffect(() => {
  setSpaceUnmountedWindowsRef.current = setSpaceUnmountedWindows;
}, [setSpaceUnmountedWindows]);

const prevPositionRef = useRef<[number, number]>(position);
const prevSizeRef = useRef<[number, number]>(size);

const onMount = useCallback(() => {}, []);

const onUnmount = useCallback(() => {
  setSpaceUnmountedWindowsRef.current([
    ...spaceUnmountedWindowsRef.current,
    spaceIdRef.current,
  ]);
```

```
}, []);

const onMountRef = useRef(onMount);
const onUnmountRef = useRef(onUnmount);

useEffect(() => {
  onMountRef.current();
  return onUnmountRef.current;
}, []);

useEffect(() => {
  focused && setFocusedWindow(spaceId);
}, [spaceId, focused, setFocusedWindow]);
useEffect(() => {
  focusedWindow === spaceId ? setFocused(true) : setFocused(false);
}, [focusedWindow, spaceId]);
useEffect(() => {
  focused ? onFocus() : onBlur();
}, [focused, onFocus, onBlur]);
useEffect(
  () => setStaging(moving && pointer[0] < scaleX(stagingDistance)),
  [moving, pointer, stagingDistance, scaleX]
);
useEffect(() => {
  !staging && setSpaceLastWindowPosition(nonZeroPosition(position));
}, [staging, position, setSpaceLastWindowPosition, managerSize]);
useEffect(() => {
  setWheelBusy(moving);
}, [moving, setWheelBusy]);
useEffect(() => setLastWindowPosition(nonZeroPosition(position)), [position]);

useEffect(() => {
  if (moving && !prevMoving) {
    setMovingStartPosition(position);
  }
  setPrevMoving(moving);
}, [moving, prevMoving, position]);

useEffect(() => {
  if (!staged) return;
```

```
if (stagedSize[0] && !stagedSize[1]) {
  const width_scale =
    size[1] > size[0] ? stagedSize[0] / size[1] : stagedSize[0] / size[0];
  const scaled_width = size[0] * width_scale;
  const scaled_height = size[1] * width_scale;
  const height_scale = scaled_height / size[1];
  const scale_to_staged = [width_scale, height_scale];

  setStagedScale(scale_to_staged);
  setScaledStagedSize([scaled_width, scaled_height]);
} else if (!stagedSize[0] && stagedSize[1]) {
  const width_scale =
    size[0] > size[1] ? stagedsWidth / size[0] : stagedSize[1] / size[1];
  const scaled_width = size[0] * width_scale;
  const scaled_height = size[1] * width_scale;
  const height_scale = scaled_height / size[1];
  const scale_to_staged = [width_scale, height_scale];

  setStagedScale(scale_to_staged);
  setScaledStagedSize([scaled_width * 0.8, scaled_height * 0.8]);
} else if (stagedSize[0] && stagedSize[1]) {
  if (size[0] > size[1]) {
    const width_scale = stagedSize[0] / size[0];
    const scaled_width = size[0] * width_scale;
    const scaled_height = size[1] * width_scale;
    const height_scale = scaled_height / size[1];
    const scale_to_staged = [width_scale, height_scale];

    setStagedScale(scale_to_staged);
    setScaledStagedSize([scaled_width, scaled_height]);
  } else {
    const height_scale = stagedSize[1] / size[1];
    const scaled_width = size[0] * height_scale;
    const scaled_height = size[1] * height_scale;
    const width_scale = scaled_width / size[0];
    const scale_to_staged = [width_scale, height_scale];

    setStagedScale(scale_to_staged);
    setScaledStagedSize([scaled_width, scaled_height]);
```

```
    }
  }
}, [staged, size, stagedSize, stagedsWidth, lastWindowPosition]);

useEffect(() => {
  if (!staged) setStagedBy("instant");
}, [staged]);

useEffect(() => {
  onWindowBoundsChanged(spaceId, position, size, true, resizing, staged);
  onUserBoundsChangeEnd(spaceId, position, size, false, resizing, staged);
}, [staged]);

useEffect(() => {
  if (!staging) return;
  if (lmb) return;
  if (staging) setStagedBy("move");
  setStaging(false);
  onStagedChange(true);
}, [staging, lmb, onStagedChange]);

useEffect(() => {
  if (alwaysOnTop == prevAlwaysOnTopRef.current) {
    if (!focused) return;
    if (focusedWindow !== spaceId) return;
    if (zIndex === windowZIndexCounter) return;
    if (zIndex === windowZIndexCounter + ALWAYS_ON_TOP_Z_INDEX) return;
  }
  prevAlwaysOnTopRef.current = alwaysOnTop;
  const newCounter = windowZIndexCounter + 1;
  setZIndex(newCounter + (alwaysOnTop ? ALWAYS_ON_TOP_Z_INDEX : 0));
  setWindowZIndexCounter(newCounter);
}, [
  focused,
  focusedWindow,
  windowZIndexCounter,
  setWindowZIndexCounter,
  spaceId,
  zIndex,
  alwaysOnTop,
```

```
]);

useEffect(() => {
  if (!moving) return;
  const distance = pointer[0];
  const scaled_distance = scaleX(stagingDistance);
  if (distance > scaled_distance) {
    setRotation(0);
    setScale(1);
    setStagingXCompenstation(0);
    setStagingYCompenstation(0);
  } else if (stagedSize[0] && !stagedSize[1]) {
    const scale_to_staged =
      size[1] > size[0] ? stagedSize[0] / size[1] : stagedSize[0] / size[0];
    const x_compensation = pointer[0] - position[0];
    const y_compensation = pointer[1] - position[1];

    setRotation(90 * (1 - distance / scaled_distance));
    setScale(scale_to_staged);
    setStagingXCompenstation(x_compensation);
    setStagingYCompenstation(y_compensation);
  } else if (!stagedSize[0] && stagedSize[1]) {
    const scale_to_staged =
      size[0] > size[1] ? stagedsWidth / size[0] : stagedSize[1] / size[1];
    const x_compensation = pointer[0] - position[0];
    const y_compensation = pointer[1] - position[1];

    setRotation(90 * (1 - distance / scaled_distance));
    setScale(scale_to_staged);
    setStagingXCompenstation(x_compensation);
    setStagingYCompenstation(y_compensation);
  } else if (stagedSize[0] && stagedSize[1]) {
    if (size[0] > size[1]) {
      const scale_to_staged = stagedSize[0] / size[0];
      const x_compensation = pointer[0] - position[0];
      const y_compensation = pointer[1] - position[1];

      setRotation(90 * (1 - distance / scaled_distance));
      setScale(scale_to_staged);
      setStagingXCompenstation(x_compensation);
```

```
      setStagingYCompenstation(y_compensation);
    } else {
      const scale_to_staged = stagedSize[1] / size[1];
      const x_compensation = pointer[0] - position[0];
      const y_compensation = pointer[1] - position[1];

      setRotation(90 * (1 - distance / scaled_distance));
      setScale(scale_to_staged);
      setStagingXCompenstation(x_compensation);
      setStagingYCompenstation(y_compensation);
    }
  }
}, [
  pointer,
  moving,
  stagingDistance,
  position,
  size,
  stagedSize,
  scaleX,
  stagedsWidth,
]);

useEffect(
  () => (moving ? onMoveStart() : onMoveEnd()),
  [moving, onMoveStart, onMoveEnd]
);

useEffect(() => {
  if (allowOutside) return;
  if (position[0] + size[0] > managerSize[0])
    onPositionChange([managerSize[0] - size[0], position[1]], "system");
  if (position[1] + size[1] > managerSize[1])
    onPositionChange([position[0], managerSize[1] - size[1]], "system");
  if (position[0] < 0) onPositionChange([0, position[1]], "system");
  if (position[1] < 0) onPositionChange([position[0], 0], "system");
}, [allowOutside, position, size, managerSize, onPositionChange]);

useEffect(() => {
  if (!compensatePositionOnViewportResize) return;
```

```
  if (
    managerSize[0] === prevManagerSize[0] &&
    managerSize[1] === prevManagerSize[1]
  )
    return;

  const [right, bottom] = [position[0] + size[0], position[1] + size[1]];
  let [x, y] = [position[0], position[1]];

  if (right > managerSize[0]) x = managerSize[0] - size[0];
  if (bottom > managerSize[1]) y = managerSize[1] - size[1];

  if (x !== position[0] || y !== position[1])
    onPositionChange([x, y], "system");

  setPrevManagerSize(managerSize);
}, [
  managerSize,
  prevManagerSize,
  position,
  size,
  onPositionChange,
  compensatePositionOnViewportResize,
]);

useEffect(() => {
  if (
    !(
      managerSize[0] !== prevManagerSize[0] ||
      managerSize[1] !== prevManagerSize[1]
    )
  )
    return;

  onPositionChange && onPositionChange(nonZeroPosition(position), "system");
  setSpaceLastWindowPosition(nonZeroPosition(position));
  setPrevManagerSize(managerSize);
}, [
  managerSize,
  prevManagerSize,
```

```
  position,
  onPositionChange,
  setSpaceLastWindowPosition,
]);

useEffect(() => {
  if (!isMobileDevice()) return;
  if (!showResizers) return;
  clearTimeout(hideResizersTimeoutRef.current);
  hideResizersTimeoutRef.current = setTimeout(
    () => setShowResizers(false),
    2500
  );
}, [showResizers]);

useEffect(() => {
  if (
    prevPositionRef.current[0] === position[0] &&
    prevPositionRef.current[1] === position[1] &&
    prevSizeRef.current[0] === size[0] &&
    prevSizeRef.current[1] === size[1]
  )
    return;
  onWindowBoundsChanged(spaceId, position, size, moving, resizing, staged);
}, [
  spaceId,
  position,
  size,
  moving,
  resizing,
  staged,
  onWindowBoundsChanged,
]);

useEffect(() => {
  if (!toSnap?.newPosition || !toSnap?.newSize) return;
  if (toSnap.target.id != spaceId) return;
  const currentAndOthers = toSnap.getCurrentAndOthers(spaceId);
  if (!currentAndOthers) return;
  const [current, others] = currentAndOthers;
```

```
  if (!current || !others) return;

  onPositionChange(toSnap.newPosition, "system");
  onSizeChange(toSnap.newSize, "system");
}, [toSnap, spaceId, onPositionChange, onSizeChange, snapMargin]);

const moveStartEventCallback = useCallback(
  (event: SpaceEvent<WindowEvent>) => {
    if (event.target.id !== spaceId) return;
    setSnapMoving(true);
  },
  [spaceId, setSnapMoving]
);

const moveEndEventCallback = useCallback(
  (event: SpaceEvent<WindowEvent>) => {
    if (event.target.id !== spaceId) return;
    setSnapMoving(false);
  },
  [spaceId, setSnapMoving]
);

const moveEventCallback = useCallback(
  (event: SpaceEvent<MoveEvent>) => {
    if (event.target.id !== spaceId) return;
    const newPosition: [number, number] = [
      position[0] + event.positionDelta[0],
      position[1] + event.positionDelta[1],
    ];
    onPositionChange(newPosition, "user");
    setSpaceLastWindowPosition(newPosition);
  },
  [spaceId, position, onPositionChange, setSpaceLastWindowPosition]
);

const resizeEventCallback = useCallback(
  (event: SpaceEvent<ResizeEvent>) => {
    if (event.target.id !== spaceId) return;
    const newSize: [number, number] = [
      size[0] + event.sizeDelta[0],
```

```
      size[1] + event.sizeDelta[1],
    ];
    if (minSize) {
      if (newSize[0] < minSize[0]) newSize[0] = minSize[0];
      if (newSize[1] < minSize[1]) newSize[1] = minSize[1];
    }
    if (maxSize) {
      if (newSize[0] > maxSize[0]) newSize[0] = maxSize[0];
      if (newSize[1] > maxSize[1]) newSize[1] = maxSize[1];
    }
    onSizeChange([newSize[0], newSize[1]], "user");
  },
  [spaceId, size, onSizeChange, minSize, maxSize]
);

useEffect(() => {
  if (!spaceEventDispatcher) return;

  spaceEventDispatcher.removeListener("move-start", moveStartEventCallback);
  spaceEventDispatcher.removeListener("move-end", moveEndEventCallback);
  spaceEventDispatcher.removeListener("move", moveEventCallback);
  spaceEventDispatcher.removeListener("resize", resizeEventCallback);

  spaceEventDispatcher.setListener("move-start", moveStartEventCallback);
  spaceEventDispatcher.setListener("move-end", moveEndEventCallback);
  spaceEventDispatcher.setListener("move", moveEventCallback);
  spaceEventDispatcher.setListener("resize", resizeEventCallback);
}, [
  spaceId,
  spaceEventDispatcher,
  moveEventCallback,
  resizeEventCallback,
  moveStartEventCallback,
  moveEndEventCallback,
]);

const onResize = useCallback(
  (size: [number, number], delta: [number, number]) => {
    const new_size: [number, number] = [
      size[0] + delta[0],
```

```
      size[1] + delta[1],
    ];

    if (minSize) {
      if (new_size[0] < minSize[0]) new_size[0] = minSize[0];
      if (new_size[1] < minSize[1]) new_size[1] = minSize[1];
    }
    if (maxSize) {
      if (new_size[0] > maxSize[0]) new_size[0] = maxSize[0];
      if (new_size[1] > maxSize[1]) new_size[1] = maxSize[1];
    }

    if (size[0] !== new_size[0] || size[1] !== new_size[1])
      onSizeChange(new_size, "user");
  },
  [onSizeChange, minSize, maxSize]
);

const onMouseMove = useCallback(
  (event: React.MouseEvent<HTMLDivElement>) => {
    if (isMobileDevice()) return;
    clearTimeout(resizerMouseMoveTimeoutRef.current);
    if (!resizable) return;
    const rect = event.currentTarget.getBoundingClientRect();

    resizerMouseMoveTimeoutRef.current = setTimeout(() => {
      const x_threshold = scaleX(resizerThreshold);
      const y_threshold = scaleY(resizerThreshold);

      const x = event.clientX - rect.x;
      const y = event.clientY - rect.y;

      setShowResizers(
        x < x_threshold ||
          y < x_threshold ||
          x > rect.width - x_threshold ||
          y > rect.height - y_threshold
      );
    }, 100);
  },
```

```
  [resizable, resizerThreshold, scaleX, scaleY]
);

const onMouseLeave = useCallback(() => {
  if (isMobileDevice()) return;
  clearTimeout(resizerMouseMoveTimeoutRef.current);
  setShowResizers(false);
}, []);

const onMoveStartCallback = useCallback(() => {
  setMoving(true);
  onWindowMoveStart(spaceId, position, size, true, resizing, staged);
}, [onWindowMoveStart, spaceId, position, size, resizing, staged]);
const onMoveEndCallback = useCallback(() => {
  setMoving(false);
  onWindowMoveEnd(spaceId, position, size, false, resizing, staged);
  onUserBoundsChangeEnd(spaceId, position, size, false, resizing, staged);
}, [
  onWindowMoveEnd,
  onUserBoundsChangeEnd,
  spaceId,
  position,
  size,
  resizing,
  staged,
]);

const onResizeStartCallback = useCallback(() => {}, []);

const onResizeEndCallback = useCallback(() => {
  onUserBoundsChangeEnd(spaceId, position, size, moving, false, staged);
}, [onUserBoundsChangeEnd, spaceId, position, size, moving, staged]);

useEffect(() => {
  if (!prevResizingRef.current && resizing) onResizeStartCallback();
  else if (prevResizingRef.current && !resizing) onResizeEndCallback();
  prevResizingRef.current = resizing;
}, [resizing, onResizeStartCallback, onResizeEndCallback]);

useEffect(() => {
```

```
  prevPositionRef.current = position;
}, [position]);
useEffect(() => {
  prevSizeRef.current = size;
}, [size]);

const contextProps = useMemo(
  () => ({
    size,
    position,
    minSize,
    maxSize,
    moving,
    resizing,
    setResizing,
    focused,
    setFocused,
    staging,
    staged,
    showResizers,
    setShowResizers,
    onResizeStart: onResizeStartCallback,
    onResizeEnd: onResizeEndCallback,
    onMoveStart: onMoveStartCallback,
    onMoveEnd: onMoveEndCallback,
  }),
  [
    size,
    position,
    minSize,
    maxSize,
    moving,
    resizing,
    setResizing,
    focused,
    staging,
    staged,
    showResizers,
    setShowResizers,
    onResizeStartCallback,
```

```
    onResizeEndCallback,
    onMoveStartCallback,
    onMoveEndCallback,
  ]
);

const resizersOnMoveCallback = useCallback(
  (position: [number, number]) => {
    onPositionChange(position, "user");
  },
  [onPositionChange]
);

const window = (
  <div
    {...(attrs as HTMLAttributes<HTMLDivElement>)}
    className={classNames([
      styles.Window,
      { [styles.Window__showResizers]: !moving && !staged && showResizers },
      { [styles.Window__mayResize]: mayResize },
      { [styles.Window__moving]: moving },
      { [styles.Window__staging]: staging },
      { [styles.Window__staged]: staged },
      { [styles.Window__focused]: focused },
      { [styles.Window__snapMoving]: snapMoving },
    ])}
    draggable={false}
    style={{
      width: size[0],
      height: size[1],
      transformOrigin: staged || staging ? "top left" : undefined,
      transform: staged
        ? `
        scale(${stagedScale[0]}, ${stagedScale[1]})
        translate(0, 0)
      `
        : staging
        ? `
        rotate3d(0, 1, 0, ${rotation}deg)
        scale(${scale})
```

```
      `
        : undefined,
      translate: staged
        ? undefined
        : staging
        ? `${
            revertScaleX(position[0]) + revertScaleX(stagingXCompenstation)
          }px ${
            revertScaleY(position[1]) + revertScaleY(stagingYCompenstation)
          }px`
        : `${revertScaleX(position[0])}px ${revertScaleY(position[1])}px`,
      zIndex: zIndex + (snapMoving ? 10000 : 0),
      ...attrs.style,
    }}
    onMouseMove={onMouseMove}
    onMouseLeave={onMouseLeave}
  >
    {children}
    {resizable && (
      <Resizers
        onMayResize={setMayResize}
        onResize={onResize}
        onMove={resizersOnMoveCallback}
      />
    )}
    <div
      className={classNames([styles.Window_stagedLayer])}
      onClick={() => {
        onStagedChange(false);
        if (stagedBy == "move") onPositionChange(movingStartPosition, "user");
        setFocused(true);
      }}
    ></div>
  </div>
);

return (
  <WindowContext.Provider value={contextProps}>
    {stagedsRef.current &&
      staged &&
```

```
        createPortal(
          <div
            className={classNames([styles.Window_stagedWindow])}
            style={{
              width: scaledStagedSize[0],
              height: scaledStagedSize[1],
              transform: staged
                ? `
        `
                : undefined,
            }}
            onMouseOver={() => setWheelBusy(true)}
            onMouseLeave={() => setWheelBusy(false)}
            onTouchStart={() => setWheelBusy(true)}
            onTouchEnd={() => setWheelBusy(false)}
            onClick={() => {
              onStagedChange(false);
              if (stagedBy == "move")
                onPositionChange(movingStartPosition, "user");
              setFocused(true);
            }}
          >
            {window}
          </div>,
          stagedsRef.current
        )}
      {!staged &&
        windowsRef.current &&
        createPortal(window, windowsRef.current)}
    </WindowContext.Provider>
  );
}

interface ResizersProps extends React.PropsWithChildren {
  onMayResize: OnMayResize;
  onResize?: (size: [number, number], delta: [number, number]) => void;
  onMove?: (position: [number, number]) => void;
}

// Manages resize handles for all edges and corners of the window
```

```
function Resizers({
  onMayResize,
  onResize = () => {},
  onMove = () => {},
}: ResizersProps) {
  const onResizeCallback = useCallback(
    (size: [number, number], delta: [number, number]) => {
      onResize(size, delta);
    },
    [onResize]
  );

  return (
    <div className={classNames([styles.Resizers])}>
      {[
        "top",
        "right",
        "bottom",
        "left",
        "top-right",
        "top-left",
        "bottom-right",
        "bottom-left",
      ].map((direction) => (
        <Resizer
          key={direction}
          direction={direction as ResizerDirection}
          onMayResize={onMayResize}
          onResize={onResizeCallback}
          onMove={onMove}
        />
      ))}
    </div>
  );
}

interface ResizerProps extends React.PropsWithChildren {
  direction: ResizerDirection;
  onMayResize: (may_resize: boolean) => void;
  onResize: (size: [number, number], delta: [number, number]) => void;
```

```
  onMove: (position: [number, number]) => void;
}

// Individual resize handle that manages resizing interactions for a specific edge/corner
function Resizer({ direction, onMayResize, onResize, onMove }: ResizerProps) {
  const {
    position: managerPosition,
    scaleX,
    scaleY,
    revertScaleX,
    revertScaleY,
    setWheelBusy,
  } = useContext(ManagerContext);
  const { lmb, pointer, setPointer } = useContext(SpaceContext);
  const { size, position, setResizing } = useContext(WindowContext);
  const [dragging, setDragging] = useState(false);
  const sizeRef = useRef(size);
  const positionRef = useRef(position);
  const prevDragPositionRef = useRef([0, 0]);

  useEffect(() => {
    setResizing(dragging);
  }, [dragging, setResizing]);

  const onResizeCallback = useCallback(
    (size: [number, number], delta: [number, number]) => {
      onResize(size, delta);
    },
    [onResize]
  );

  useEffect(() => {
    if (!dragging) return;

    // Calculate position delta and apply appropriate resize/move operations based on
direction
    const delta = [
      pointer[0] - prevDragPositionRef.current[0],
      pointer[1] - prevDragPositionRef.current[1],
    ];
```

```
  // Handle different resize directions and their corresponding size/position updates
  if (direction === "bottom") {
    onResizeCallback(sizeRef.current, [0, revertScaleX(delta[1])]);
  } else if (direction === "right") {
    onResizeCallback(sizeRef.current, [revertScaleX(delta[0]), 0]);
  } else if (direction === "bottom-right") {
    onResizeCallback(sizeRef.current, [
      revertScaleX(delta[0]),
      revertScaleY(delta[1]),
    ]);
  } else if (direction === "top") {
    onResizeCallback(sizeRef.current, [0, -revertScaleY(delta[1])]);
    onMove([positionRef.current[0], positionRef.current[1] + delta[1]]);
  } else if (direction === "left") {
    onResizeCallback(sizeRef.current, [-revertScaleX(delta[0]), 0]);
    onMove([positionRef.current[0] + delta[0], positionRef.current[1]]);
  } else if (direction === "top-left") {
    onResizeCallback(sizeRef.current, [
      -revertScaleX(delta[0]),
      -revertScaleY(delta[1]),
    ]);
    onMove([
      positionRef.current[0] + delta[0],
      positionRef.current[1] + delta[1],
    ]);
  } else if (direction === "top-right") {
    onResizeCallback(sizeRef.current, [
      revertScaleX(delta[0]),
      -revertScaleY(delta[1]),
    ]);
    onMove([positionRef.current[0], positionRef.current[1] + delta[1]]);
  } else if (direction === "bottom-left") {
    onResizeCallback(sizeRef.current, [
      -revertScaleX(delta[0]),
      revertScaleY(delta[1]),
    ]);
    onMove([positionRef.current[0] + delta[0], positionRef.current[1]]);
  } else throw new Error("Invalid direction");
}, [
```

```
  pointer,
  dragging,
  direction,
  onResizeCallback,
  onMove,
  scaleX,
  scaleY,
  revertScaleX,
  revertScaleY,
]);

useEffect(() => {
  sizeRef.current = size;
}, [size]);
useEffect(() => {
  !lmb && setDragging(false);
}, [lmb]);
useEffect(() => {
  prevDragPositionRef.current = pointer;
}, [pointer]);
useEffect(() => {
  positionRef.current = position;
}, [position]);

const onMouseDown = useCallback(() => {
  onMayResize(true);
  setDragging(true);
}, [onMayResize]);

const onMouseLeave = useCallback(() => {
  onMayResize(false);
}, [onMayResize]);

const onTouchStart = useCallback(
  (event: React.TouchEvent<HTMLDivElement>) => {
    if (event.touches.length !== 1) return;
    const new_pointer: [number, number] = [
      event.touches[0].clientX - managerPosition[0],
      event.touches[0].clientY - managerPosition[1],
    ];
```

```
      setDragging(true);
      onMayResize(true);
      setWheelBusy(true);
      setPointer(new_pointer);
      prevDragPositionRef.current = new_pointer;
    },
    [onMayResize, setWheelBusy, setPointer, managerPosition]
  );

  const onTouchEnd = useCallback(() => {
    setDragging(false);
    onMayResize(false);
    setWheelBusy(false);
  }, [onMayResize, setWheelBusy]);

  return (
    <div
      className={classNames([
        styles.Resizer,
        { [styles.Resizer__top]: direction === "top" },
        { [styles.Resizer__right]: direction === "right" },
        { [styles.Resizer__bottom]: direction === "bottom" },
        { [styles.Resizer__left]: direction === "left" },
        { [styles.Resizer__topRight]: direction === "top-right" },
        { [styles.Resizer__topLeft]: direction === "top-left" },
        { [styles.Resizer__bottomRight]: direction === "bottom-right" },
        { [styles.Resizer__bottomLeft]: direction === "bottom-left" },
      ])}
      draggable={false}
      onMouseDown={onMouseDown}
      onMouseLeave={onMouseLeave}
      onTouchStart={onTouchStart}
      onTouchEnd={onTouchEnd}
    ></div>
  );
}

interface TitleBarProps extends React.HTMLAttributes<HTMLDivElement> {
  children?: React.ReactNode;
  onMove?: ([x, y]: [number, number]) => void;
```

```
}

// Draggable title bar component that enables window movement and contains window
controls
function TitleBar({
  children = null,
  onMove = () => {},
  ...attrs
}: TitleBarProps) {
  const {
    position: managerPosition,
    setWheelBusy,
    setPointer,
  } = useContext(ManagerContext);
  const { lmb, pointer } = useContext(SpaceContext);
  const { position, setFocused, onMoveStart, onMoveEnd, setShowResizers } =
    useContext(WindowContext);
  const [dragging, setDragging] = useState(false);
  const [movingPosition, setMovingPosition] =
    useState<[number, number]>(position);
  const moveStartPointerRef = useRef<[number, number]>([0, 0]);
  const onMoveStartRef = useRef(onMoveStart);
  const onMoveEndRef = useRef(onMoveEnd);

  useEffect(() => {
    onMoveStartRef.current = onMoveStart;
    onMoveEndRef.current = onMoveEnd;
  }, [onMoveStart, onMoveEnd]);

  useEffect(() => {
    if (dragging) {
      onMoveStartRef.current();
    } else {
      onMoveEndRef.current();
    }
  }, [dragging]);
  useEffect(
    () => onMove([movingPosition[0], movingPosition[1]]),
    [movingPosition, onMove]
  );
```

```
useEffect(() => {
  // Update window position during drag operations
  if (!dragging) return;
  setMovingPosition([
    pointer[0] - moveStartPointerRef.current[0],
    pointer[1] - moveStartPointerRef.current[1],
  ]);
}, [dragging, pointer, moveStartPointerRef]);

useEffect(() => {
  if (!dragging) return;
  if (lmb) return;
  setDragging(false);
}, [lmb, dragging]);

const onMouseDown = useCallback(
  (event: React.MouseEvent<HTMLDivElement>) => {
    setFocused(true);
    setDragging(true);
    const rect = event.currentTarget.getBoundingClientRect();
    moveStartPointerRef.current = [
      event.clientX - rect.x,
      event.clientY - rect.y,
    ];
  },
  [setFocused]
);

const onMouseUp = useCallback(() => {
  setDragging(false);
}, []);

const onTouchStart = useCallback(
  (event: React.TouchEvent<HTMLDivElement>) => {
    if (event.touches.length !== 1) return;
    const touch = event.touches[0];
    const rect = event.currentTarget.getBoundingClientRect();
    const new_pointer: [number, number] = [
      touch.clientX - managerPosition[0],
```

```
        touch.clientY - managerPosition[1],
      ];
      moveStartPointerRef.current = [
        touch.clientX - rect.x,
        touch.clientY - rect.y,
      ];
      setPointer(new_pointer);
      setWheelBusy(true);
      setDragging(true);
      setFocused(true);
      setShowResizers(true);
    },
    [setWheelBusy, setFocused, setShowResizers, setPointer, managerPosition]
  );

  const onTouchEnd = useCallback(() => setDragging(false), []);

  return (
    <div
      {...(attrs as React.HTMLAttributes<HTMLDivElement>)}
      className={`${classNames([
        styles.TitleBar,
        dragging && styles.TitleBar__dragging,
      ])} ${typeof attrs.className !== "undefined" ? attrs.className : ""}`}
      draggable={false}
      onMouseDown={onMouseDown}
      onMouseUp={onMouseUp}
      onTouchStart={onTouchStart}
      onTouchEnd={() => onTouchEnd}
    >
      {children}
      <div
        className={classNames([styles.TitleBar_stagingLayer])}
        onMouseUp={onMouseUp}
        onTouchEnd={() => onTouchEnd()}
      ></div>
    </div>
  );
}
```

```
// Container for window title text
function Title({
  children = null,
  ...attrs
}: React.HTMLAttributes<HTMLDivElement>) {
  return (
    <div
      {...(attrs as HTMLAttributes<HTMLDivElement>)}
      className={`${classNames([styles.Title])} ${
        typeof attrs.className !== "undefined" ? attrs.className : ""
      }`}
      draggable={false}
    >
      {children}
    </div>
  );
}

// Container for window control buttons (close, minimize, etc.)
function Buttons({
  children = null,
  ...attrs
}: React.HTMLAttributes<HTMLDivElement>) {
  return (
    <div
      {...(attrs as HTMLAttributes<HTMLDivElement>)}
      className={`${classNames([styles.Buttons])} ${
        typeof attrs.className !== "undefined" ? attrs.className : ""
      }`}
      onClick={(event) => event.stopPropagation()}
      draggable={false}
    >
      {children}
    </div>
  );
}

interface CloseButtonProps extends React.PropsWithChildren {
  children?: React.ReactNode;
  onClick?: (event: React.MouseEvent<HTMLDivElement, MouseEvent>) => void;
```

```
}

/**
 * @description CloseButton component, for the close button of the window.
 */
function CloseButton({
  children = (
    <div
      className={classNames([styles.Button, styles.CloseButton])}
      onClick={(event) => {
        event.stopPropagation();
        onClick(event);
      }}
      onMouseDown={(event) => event.stopPropagation()}
      onMouseUp={(event) => event.stopPropagation()}
      onTouchStart={(event) => event.stopPropagation()}
      onTouchEnd={(event) => event.stopPropagation()}
    >
      ✖
    </div>
  ),
  onClick = () => {},
}: CloseButtonProps) {
  return children;
}

interface StageButtonProps extends React.PropsWithChildren {
  children?: React.ReactNode;
  onClick?: (event: React.MouseEvent<HTMLDivElement, MouseEvent>) => void;
}

/**
 * @description StageButton component, for minimizing the window.
 */
function StageButton({
  children = (
    <div
      className={classNames([styles.Button, styles.StageButton])}
      onClick={(event) => {
        event.stopPropagation();
```

```
        onClick(event);
      }}
      onMouseDown={(event) => event.stopPropagation()}
      onMouseUp={(event) => event.stopPropagation()}
      onTouchStart={(event) => event.stopPropagation()}
      onTouchEnd={(event) => event.stopPropagation()}
    >
      <span style={{ fontSize: 10 }}>—</span>
    </div>
  ),
  onClick = () => {},
}: StageButtonProps) {
  return children;
}

interface ContentProps extends React.HTMLAttributes<HTMLDivElement> {
  children?: React.ReactNode;
  onUnfocusedWheel?: (event: React.WheelEvent<HTMLDivElement>) => boolean;
}

const DEFAULT_ON_UNFOCUSED_WHEEL = (): boolean => true;

// Main content area of the window that handles scrolling and focus management
function Content({
  children = null,
  onUnfocusedWheel = DEFAULT_ON_UNFOCUSED_WHEEL,
  ...attrs
}: ContentProps) {
  const { setWheelBusy } = useContext(ManagerContext);
  const { setFocused } = useContext(WindowContext);
  const fluidRef = useRef<HTMLDivElement>(null);

  return (
    <div
      {...(attrs as HTMLAttributes<HTMLDivElement>)}
      className={`${classNames([styles.Content])} ${
        typeof attrs.className !== "undefined" ? attrs.className : ""
      }`}
      onWheel={(event) => event.stopPropagation()}
    >
```

```
      <div className={classNames([styles.Content_fluid])} ref={fluidRef}>
        {children}
      </div>
      <div
        className={classNames([styles.Content_unfocusedLayer])}
        onClick={() => setFocused(true)}
        onWheel={(event) => {
          if (!onUnfocusedWheel(event)) return;
          fluidRef.current?.scrollBy({ top: event.deltaY, left: 0 });
        }}
        onTouchStart={() => setWheelBusy(true)}
        onTouchEnd={() => setWheelBusy(false)}
      ></div>
    </div>
  );
}

interface BasicWindowProps extends React.HTMLAttributes<HTMLDivElement> {
  children?: React.ReactNode;
  title?: string;
  initialSize?: [number, number];
  initialPosition?: [number, number] | "random" | "auto";
  opened?: boolean;
  onClose?: () => void | undefined;
  onTitleChange?: (newTitle: string) => void;
  content?: string;
  onContentChange?: (content: string) => void;
  onPositionChange?: (position: [number, number]) => void;
  onSizeChange?: (size: [number, number]) => void;
}

// High-level window implementation with standard title bar, content area, and window
controls
function BasicWindow({
  title = "Window",
  content = "", // Changed to empty string by default
  initialSize = [500, 400],
  initialPosition = "auto",
  onTitleChange,
  onContentChange,
```

```
  onClose,
  onPositionChange,
  onSizeChange,
  ...attrs
}: BasicWindowProps) {
  const [position, setPosition] = usePosition(initialPosition);
  const [size, setSize] = useState(initialSize);
  const [spaceId] = useSpaceId();
  const [currentTitle, setCurrentTitle] = useState(title);
  const [currentContent, setCurrentContent] = useState(content);
  const [staged, setStaged] = useState(false);
  const [isEditingContent, setIsEditingContent] = useState(false);
  const [isEmpty, setIsEmpty] = useState(!content);

  // Handle title changes
  const handleTitleChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    const newTitle = e.target.value;
    setCurrentTitle(newTitle);
    onTitleChange?.(newTitle);
  };

  // When the Firestore content changes (and not currently editing), update the state.
  useEffect(() => {
    // Sync content with external changes only when not being edited locally
    if (!isEditingContent) {
      setCurrentContent(content);
      setIsEmpty(!content.trim());
    }
  }, [content, isEditingContent]);

  // Content change handler for the textarea
  const handleContentChange = (e: React.ChangeEvent<HTMLTextAreaElement>) => {
    const newContent = e.target.value;
    setCurrentContent(newContent);
    setIsEmpty(!newContent.trim());
    onContentChange?.(newContent);
  };

  return (
    <Window
```

```
  spaceId={spaceId}
  position={position}
  size={size}
  staged={staged}
  onStagedChange={setStaged}
  onPositionChange={
    onPositionChange
      ? (newPosition: [number, number], reason: any) => {
          setPosition(newPosition);
          onPositionChange(newPosition);
        }
      : (newPosition: [number, number], reason: any) => setPosition(newPosition)
  }
  onSizeChange={
    onSizeChange
      ? (newSize: [number, number], reason: any) => {
          setSize(newSize);
          onSizeChange(newSize);
        }
      : (newSize: [number, number], reason: any) => setSize(newSize)
  }
  style={attrs.style}
>
  <TitleBar onMove={setPosition}>
    <Buttons>
      <CloseButton onClick={onClose} />
      <StageButton onClick={() => setStaged(!staged)} />
    </Buttons>
    <Title>
      <input
        type="text"
        value={currentTitle}
        onChange={handleTitleChange}
        className="outline-none bg-transparent font-bold w-full"
      />
    </Title>
  </TitleBar>
  <Content>
    <textarea
      value={currentContent}
```

```
          onChange={handleContentChange}
          onFocus={() => setIsEditingContent(true)}
          onBlur={() => setIsEditingContent(false)}
          placeholder="Start typing..."
          className={`p-4 min-h-full outline-none w-full h-full resize-none border-none
bg-transparent placeholder-gray-400`}
        />
      </Content>
    </Window>
  );
}


export {
  Window,
  BasicWindow,
  TitleBar,
  Title,
  Buttons,
  CloseButton,
  StageButton,
  Content,
};
```

File: src\components\stage-manager\components\Window\index.tsx
---

```
export {
  Window,
  BasicWindow,
  TitleBar,
  Title,
  Buttons,
  CloseButton,
  StageButton,
  Content,
} from "./Window";
export { WindowContext } from "./library";
export type { WindowContextProps } from "./library";
```

File: src\components\stage-manager\components\Window\library.ts
---

```
import { createContext } from "react";

export const ALWAYS_ON_TOP_Z_INDEX: number = 1000000;

export interface WindowContextProps {
  size: [number, number];
  position: [number, number];
  minSize: [number, number] | null;
  maxSize: [number, number] | null;
  moving: boolean;
  resizing: boolean;
  setResizing: React.Dispatch<React.SetStateAction<boolean>>;
  focused: boolean;
  setFocused: React.Dispatch<React.SetStateAction<boolean>>;
  staging: boolean;
  staged: boolean;
  showResizers: boolean;
  setShowResizers: React.Dispatch<React.SetStateAction<boolean>>;
  onMoveStart: () => void;
  onMoveEnd: () => void;
  onResizeStart: () => void;
  onResizeEnd: () => void;
}

export const WindowContext = createContext<WindowContextProps>({
  size: [0, 0],
  position: [0, 0],
  minSize: null,
  maxSize: null,
  moving: false,
  resizing: false,
  setResizing: () => {},
  focused: false,
  setFocused: () => {},
  staging: false,
```

```
  staged: false,
  showResizers: false,
  setShowResizers: () => {},
  onMoveStart: () => {},
  onMoveEnd: () => {},
  onResizeStart: () => {},
  onResizeEnd: () => {},
});
```

UI Component Library & Custom UI

```
==========
Directory: src\hooks
==========

File: src\hooks\use-404-redirect.ts
----------

import { useEffect } from "react";
import { useRouter, usePathname } from "next/navigation";

/**
 * Hook to redirect users to the parent directory in case of a 404-like situation.
 * This is particularly useful for scenarios where users might navigate to a non-existent child route,
 * and a more user-friendly behavior is to redirect them to the nearest existing parent route.
 */
export const use404Redirect = () => {
  // useRouter hook from next/navigation to programmatically navigate between routes.
  const router = useRouter();
  // usePathname hook to get the current pathname of the URL.
  const pathname = usePathname();

  useEffect(() => {
    // Split the pathname into segments (parts of the path divided by '/') and filter out any empty segments.
    const currentPathSegments = pathname.split("/").filter(Boolean);
    // Create an array of parent path segments by removing the last segment (current page).
    const parentPathSegments = currentPathSegments.slice(0, -1);
    // Join the parent path segments back together with '/' to form the parent path.
    // If there are no parent segments, default to the root path '/'.
    const parentPath = parentPathSegments.length
      ? `/${parentPathSegments.join("/")}`
      : "/";

    // Programmatically redirect the user to the constructed parent path.
```

```
    // This effect runs whenever the pathname changes, ensuring redirection if a 404-like path is accessed.
    router.push(parentPath);
  }, [pathname, router]); // Effect dependencies: pathname and router. The effect re-runs if either of these change.
};
```

File: src\hooks\use-mobile.tsx

```
import * as React from "react";

const MOBILE_BREAKPOINT = 768;

export function useIsMobile() {
  const [isMobile, setIsMobile] = React.useState<boolean | undefined>(
    undefined
  );

  React.useEffect(() => {
    const mql = window.matchMedia(`(max-width: ${MOBILE_BREAKPOINT - 1}px)`);
    const onChange = () => {
      setIsMobile(window.innerWidth < MOBILE_BREAKPOINT);
    };
    mql.addEventListener("change", onChange);
    setIsMobile(window.innerWidth < MOBILE_BREAKPOINT);
    return () => mql.removeEventListener("change", onChange);
  }, []);

  return !!isMobile;
}
```

File: src\hooks\use-toast.ts

```
"use client";

// Inspired by react-hot-toast library
import * as React from "react";
```

```
import type { ToastActionElement, ToastProps } from "@/components/ui/toast";

const TOAST_LIMIT = 1;
const TOAST_REMOVE_DELAY = 1000000;

type ToasterToast = ToastProps & {
  id: string;
  title?: React.ReactNode;
  description?: React.ReactNode;
  action?: ToastActionElement;
};

const actionTypes = {
  ADD_TOAST: "ADD_TOAST",
  UPDATE_TOAST: "UPDATE_TOAST",
  DISMISS_TOAST: "DISMISS_TOAST",
  REMOVE_TOAST: "REMOVE_TOAST",
} as const;

let count = 0;

function genId() {
  count = (count + 1) % Number.MAX_SAFE_INTEGER;
  return count.toString();
}

type ActionType = typeof actionTypes;

type Action =
  | {
      type: ActionType["ADD_TOAST"];
      toast: ToasterToast;
    }
  | {
      type: ActionType["UPDATE_TOAST"];
      toast: Partial<ToasterToast>;
    }
  | {
      type: ActionType["DISMISS_TOAST"];
```

```
      toastId?: ToasterToast["id"];
    }
  | {
      type: ActionType["REMOVE_TOAST"];
      toastId?: ToasterToast["id"];
    };

interface State {
  toasts: ToasterToast[];
}

const toastTimeouts = new Map<string, ReturnType<typeof setTimeout>>();

const addToRemoveQueue = (toastId: string) => {
  if (toastTimeouts.has(toastId)) {
    return;
  }

  const timeout = setTimeout(() => {
    toastTimeouts.delete(toastId);
    dispatch({
      type: "REMOVE_TOAST",
      toastId: toastId,
    });
  }, TOAST_REMOVE_DELAY);

  toastTimeouts.set(toastId, timeout);
};

export const reducer = (state: State, action: Action): State => {
  switch (action.type) {
    case "ADD_TOAST":
      return {
        ...state,
        toasts: [action.toast, ...state.toasts].slice(0, TOAST_LIMIT),
      };

    case "UPDATE_TOAST":
      return {
        ...state,
```

```
    toasts: state.toasts.map((t) =>
      t.id === action.toast.id ? { ...t, ...action.toast } : t
    ),
  };

case "DISMISS_TOAST": {
  const { toastId } = action;

  // ! Side effects ! - This could be extracted into a dismissToast() action,
  // but I'll keep it here for simplicity
  if (toastId) {
    addToRemoveQueue(toastId);
  } else {
    state.toasts.forEach((toast) => {
      addToRemoveQueue(toast.id);
    });
  }

  return {
    ...state,
    toasts: state.toasts.map((t) =>
      t.id === toastId || toastId === undefined
        ? {
            ...t,
            open: false,
          }
        : t
    ),
  };
}
case "REMOVE_TOAST":
  if (action.toastId === undefined) {
    return {
      ...state,
      toasts: [],
    };
  }
  return {
    ...state,
    toasts: state.toasts.filter((t) => t.id !== action.toastId),
```

```
      };
  }
};

const listeners: Array<(state: State) => void> = [];

let memoryState: State = { toasts: [] };

function dispatch(action: Action) {
  memoryState = reducer(memoryState, action);
  listeners.forEach((listener) => {
    listener(memoryState);
  });
}

type Toast = Omit<ToasterToast, "id">;

function toast({ ...props }: Toast) {
  const id = genId();

  const update = (props: ToasterToast) =>
    dispatch({
      type: "UPDATE_TOAST",
      toast: { ...props, id },
    });
  const dismiss = () => dispatch({ type: "DISMISS_TOAST", toastId: id });

  dispatch({
    type: "ADD_TOAST",
    toast: {
      ...props,
      id,
      open: true,
      onOpenChange: (open) => {
        if (!open) dismiss();
      },
    },
  });

  return {
```

```
    id: id,
    dismiss,
    update,
  };
}

function useToast() {
  const [state, setState] = React.useState<State>(memoryState);

  React.useEffect(() => {
    listeners.push(setState);
    return () => {
      const index = listeners.indexOf(setState);
      if (index > -1) {
        listeners.splice(index, 1);
      }
    };
  }, [state]);

  return {
    ...state,
    toast,
    dismiss: (toastId?: string) => dispatch({ type: "DISMISS_TOAST", toastId }),
  };
}

export { useToast, toast };


==========
Directory: src\app\components
==========

File: src\app\components\SpaceDeleteAlertToast.tsx
--------------------

import { Button } from "@/components/ui/button";
import { useToast } from "@/hooks/use-toast";

interface SpaceDeleteButtonProps {
```

```
  spaceId: number;
  onSpaceDelete: (spaceId: number) => void;
}

interface WindowDeleteButtonProps {
  spaceId: number;
  windowId: string;
  windowTitle: string;
  onWindowDelete: (spaceId: number, windowId: string) => void;
}

export function SpaceDeleteToast({
  spaceId,
  onSpaceDelete,
}: SpaceDeleteButtonProps) {
  const { toast } = useToast();

  const handleDeleteClick = () => {
    toast({
      title: "Delete Space",
      description: "Are you sure you want to delete this space?",
      action: (
        <Button
          variant="outline"
          size="sm"
          onClick={() => {
            onSpaceDelete(spaceId);
            toast({
              title: "Space deleted",
              description: `Space ${
                spaceId + 1
              } has been deleted successfully.`,
            });
          }}
        >
          Delete
        </Button>
      ),
    });
  };
```

```
  return (
    <Button variant="destructive" onClick={handleDeleteClick}>
      Delete Space {spaceId + 1}
    </Button>
  );
}

export function WindowDeleteToast({
  spaceId,
  windowId,
  windowTitle,
  onWindowDelete,
}: WindowDeleteButtonProps) {
  const { toast } = useToast();

  const handleDeleteClick = () => {
    toast({
      title: "Delete Window",
      description: `Are you sure you want to delete "${windowTitle}"?`,
      action: (
        <Button
          variant="outline"
          size="sm"
          onClick={() => {
            onWindowDelete(spaceId, windowId);
            toast({
              title: "Window deleted",
              description: `"${windowTitle}" has been deleted successfully.`,
            });
          }}
        >
          Delete
        </Button>
      ),
    });
  };

  return (
    <div onClick={handleDeleteClick} className="cursor-pointer">
```

```
      ✖
    </div>
  );
}


===========
Directory: src\components\ui
===========

File: src\components\ui\button.tsx
-------------------

import * as React from "react"
import { Slot } from "@radix-ui/react-slot"
import { cva, type VariantProps } from "class-variance-authority"

import { cn } from "@/lib/utils"

const buttonVariants = cva(
  "inline-flex items-center justify-center gap-2 whitespace-nowrap rounded-md text-sm
font-medium ring-offset-background transition-colors focus-visible:outline-none
focus-visible:ring-2 focus-visible:ring-ring focus-visible:ring-offset-2
disabled:pointer-events-none disabled:opacity-50 [&_svg]:pointer-events-none
[&_svg]:size-4 [&_svg]:shrink-0",
  {
    variants: {
      variant: {
        default: "bg-primary text-primary-foreground hover:bg-primary/90",
        destructive:
          "bg-destructive text-destructive-foreground hover:bg-destructive/90",
        outline:
          "border border-input bg-background hover:bg-accent
hover:text-accent-foreground",
        secondary:
          "bg-secondary text-secondary-foreground hover:bg-secondary/80",
        ghost: "hover:bg-accent hover:text-accent-foreground",
        link: "text-primary underline-offset-4 hover:underline",
      },
      size: {
```

```
        default: "h-10 px-4 py-2",
        sm: "h-9 rounded-md px-3",
        lg: "h-11 rounded-md px-8",
        icon: "h-10 w-10",
      },
    },
    defaultVariants: {
      variant: "default",
      size: "default",
    },
  }
)

export interface ButtonProps
  extends React.ButtonHTMLAttributes<HTMLButtonElement>,
    VariantProps<typeof buttonVariants> {
  asChild?: boolean
}

const Button = React.forwardRef<HTMLButtonElement, ButtonProps>(
  ({ className, variant, size, asChild = false, ...props }, ref) => {
    const Comp = asChild ? Slot : "button"
    return (
      <Comp
        className={cn(buttonVariants({ variant, size, className }))}
        ref={ref}
        {...props}
      />
    )
  }
)
Button.displayName = "Button"

export { Button, buttonVariants }
```

File: src\components\ui\card.tsx

```
import * as React from "react"
```

```
import { cn } from "@/lib/utils"

const Card = React.forwardRef<
  HTMLDivElement,
  React.HTMLAttributes<HTMLDivElement>
>(({ className, ...props }, ref) => (
  <div
    ref={ref}
    className={cn(
      "rounded-lg border bg-card text-card-foreground shadow-sm",
      className
    )}
    {...props}
  />
))
Card.displayName = "Card"

const CardHeader = React.forwardRef<
  HTMLDivElement,
  React.HTMLAttributes<HTMLDivElement>
>(({ className, ...props }, ref) => (
  <div
    ref={ref}
    className={cn("flex flex-col space-y-1.5 p-6", className)}
    {...props}
  />
))
CardHeader.displayName = "CardHeader"

const CardTitle = React.forwardRef<
  HTMLDivElement,
  React.HTMLAttributes<HTMLDivElement>
>(({ className, ...props }, ref) => (
  <div
    ref={ref}
    className={cn(
      "text-2xl font-semibold leading-none tracking-tight",
      className
    )}
```

```
    {...props}
  />
))
CardTitle.displayName = "CardTitle"

const CardDescription = React.forwardRef<
  HTMLDivElement,
  React.HTMLAttributes<HTMLDivElement>
>(({ className, ...props }, ref) => (
  <div
    ref={ref}
    className={cn("text-sm text-muted-foreground", className)}
    {...props}
  />
))
CardDescription.displayName = "CardDescription"

const CardContent = React.forwardRef<
  HTMLDivElement,
  React.HTMLAttributes<HTMLDivElement>
>(({ className, ...props }, ref) => (
  <div ref={ref} className={cn("p-6 pt-0", className)} {...props} />
))
CardContent.displayName = "CardContent"

const CardFooter = React.forwardRef<
  HTMLDivElement,
  React.HTMLAttributes<HTMLDivElement>
>(({ className, ...props }, ref) => (
  <div
    ref={ref}
    className={cn("flex items-center p-6 pt-0", className)}
    {...props}
  />
))
CardFooter.displayName = "CardFooter"

export { Card, CardHeader, CardFooter, CardTitle, CardDescription, CardContent }
```

File: src\components\ui\checkbox.tsx

```
"use client"

import * as React from "react"
import * as CheckboxPrimitive from "@radix-ui/react-checkbox"
import { Check } from "lucide-react"

import { cn } from "@/lib/utils"

const Checkbox = React.forwardRef<
  React.ElementRef<typeof CheckboxPrimitive.Root>,
  React.ComponentPropsWithoutRef<typeof CheckboxPrimitive.Root>
>(({ className, ...props }, ref) => (
  <CheckboxPrimitive.Root
    ref={ref}
    className={cn(
      "peer h-4 w-4 shrink-0 rounded-sm border border-primary ring-offset-background
focus-visible:outline-none focus-visible:ring-2 focus-visible:ring-ring
focus-visible:ring-offset-2 disabled:cursor-not-allowed disabled:opacity-50
data-[state=checked]:bg-primary data-[state=checked]:text-primary-foreground",
      className
    )}
    {...props}
  >
    <CheckboxPrimitive.Indicator
      className={cn("flex items-center justify-center text-current")}
    >
      <Check className="h-4 w-4" />
    </CheckboxPrimitive.Indicator>
  </CheckboxPrimitive.Root>
))
Checkbox.displayName = CheckboxPrimitive.Root.displayName

export { Checkbox }
```

File: src\components\ui\dialog.tsx

```
"use client"

import * as React from "react"
import * as DialogPrimitive from "@radix-ui/react-dialog"
import { X } from "lucide-react"

import { cn } from "@/lib/utils"

const Dialog = DialogPrimitive.Root

const DialogTrigger = DialogPrimitive.Trigger

const DialogPortal = DialogPrimitive.Portal

const DialogClose = DialogPrimitive.Close

const DialogOverlay = React.forwardRef<
  React.ElementRef<typeof DialogPrimitive.Overlay>,
  React.ComponentPropsWithoutRef<typeof DialogPrimitive.Overlay>
>(({ className, ...props }, ref) => (
  <DialogPrimitive.Overlay
    ref={ref}
    className={cn(
      "fixed inset-0 z-50 bg-black/80  data-[state=open]:animate-in
data-[state=closed]:animate-out data-[state=closed]:fade-out-0
data-[state=open]:fade-in-0",
      className
    )}
    {...props}
  />
))
DialogOverlay.displayName = DialogPrimitive.Overlay.displayName

const DialogContent = React.forwardRef<
  React.ElementRef<typeof DialogPrimitive.Content>,
  React.ComponentPropsWithoutRef<typeof DialogPrimitive.Content>
>(({ className, children, ...props }, ref) => (
  <DialogPortal>
    <DialogOverlay />
```

```
    <DialogPrimitive.Content
      ref={ref}
      className={cn(
        "fixed left-[50%] top-[50%] z-50 grid w-full max-w-lg translate-x-[-50%]
translate-y-[-50%] gap-4 border bg-background p-6 shadow-lg duration-200
data-[state=open]:animate-in data-[state=closed]:animate-out
data-[state=closed]:fade-out-0 data-[state=open]:fade-in-0
data-[state=closed]:zoom-out-95 data-[state=open]:zoom-in-95
data-[state=closed]:slide-out-to-left-1/2 data-[state=closed]:slide-out-to-top-[48%]
data-[state=open]:slide-in-from-left-1/2 data-[state=open]:slide-in-from-top-[48%]
sm:rounded-lg",
        className
      )}
      {...props}
    >
      {children}
      <DialogPrimitive.Close className="absolute right-4 top-4 rounded-sm opacity-70
ring-offset-background transition-opacity hover:opacity-100 focus:outline-none
focus:ring-2 focus:ring-ring focus:ring-offset-2 disabled:pointer-events-none
data-[state=open]:bg-accent data-[state=open]:text-muted-foreground">
        <X className="h-4 w-4" />
        <span className="sr-only">Close</span>
      </DialogPrimitive.Close>
    </DialogPrimitive.Content>
  </DialogPortal>
))
DialogContent.displayName = DialogPrimitive.Content.displayName

const DialogHeader = ({
  className,
  ...props
}: React.HTMLAttributes<HTMLDivElement>) => (
  <div
    className={cn(
      "flex flex-col space-y-1.5 text-center sm:text-left",
      className
    )}
    {...props}
  />
)
```

```
DialogHeader.displayName = "DialogHeader"

const DialogFooter = ({
  className,
  ...props
}: React.HTMLAttributes<HTMLDivElement>) => (
  <div
    className={cn(
      "flex flex-col-reverse sm:flex-row sm:justify-end sm:space-x-2",
      className
    )}
    {...props}
  />
)
DialogFooter.displayName = "DialogFooter"

const DialogTitle = React.forwardRef<
  React.ElementRef<typeof DialogPrimitive.Title>,
  React.ComponentPropsWithoutRef<typeof DialogPrimitive.Title>
>(({ className, ...props }, ref) => (
  <DialogPrimitive.Title
    ref={ref}
    className={cn(
      "text-lg font-semibold leading-none tracking-tight",
      className
    )}
    {...props}
  />
))
DialogTitle.displayName = DialogPrimitive.Title.displayName

const DialogDescription = React.forwardRef<
  React.ElementRef<typeof DialogPrimitive.Description>,
  React.ComponentPropsWithoutRef<typeof DialogPrimitive.Description>
>(({ className, ...props }, ref) => (
  <DialogPrimitive.Description
    ref={ref}
    className={cn("text-sm text-muted-foreground", className)}
    {...props}
  />
```

```
))
DialogDescription.displayName = DialogPrimitive.Description.displayName

export {
  Dialog,
  DialogPortal,
  DialogOverlay,
  DialogClose,
  DialogTrigger,
  DialogContent,
  DialogHeader,
  DialogFooter,
  DialogTitle,
  DialogDescription,
}
```

File: src\components\ui\dropdown-menu.tsx

```
"use client"

import * as React from "react"
import * as DropdownMenuPrimitive from "@radix-ui/react-dropdown-menu"
import { Check, ChevronRight, Circle } from "lucide-react"

import { cn } from "@/lib/utils"

const DropdownMenu = DropdownMenuPrimitive.Root

const DropdownMenuTrigger = DropdownMenuPrimitive.Trigger

const DropdownMenuGroup = DropdownMenuPrimitive.Group

const DropdownMenuPortal = DropdownMenuPrimitive.Portal

const DropdownMenuSub = DropdownMenuPrimitive.Sub

const DropdownMenuRadioGroup = DropdownMenuPrimitive.RadioGroup
```

```
const DropdownMenuSubTrigger = React.forwardRef<
  React.ElementRef<typeof DropdownMenuPrimitive.SubTrigger>,
  React.ComponentPropsWithoutRef<typeof DropdownMenuPrimitive.SubTrigger> & {
    inset?: boolean
  }
>(({ className, inset, children, ...props }, ref) => (
  <DropdownMenuPrimitive.SubTrigger
    ref={ref}
    className={cn(
      "flex cursor-default gap-2 select-none items-center rounded-sm px-2 py-1.5 text-sm
outline-none focus:bg-accent data-[state=open]:bg-accent [&_svg]:pointer-events-none
[&_svg]:size-4 [&_svg]:shrink-0",
      inset && "pl-8",
      className
    )}
    {...props}
  >
    {children}
    <ChevronRight className="ml-auto" />
  </DropdownMenuPrimitive.SubTrigger>
))
DropdownMenuSubTrigger.displayName =
  DropdownMenuPrimitive.SubTrigger.displayName

const DropdownMenuSubContent = React.forwardRef<
  React.ElementRef<typeof DropdownMenuPrimitive.SubContent>,
  React.ComponentPropsWithoutRef<typeof DropdownMenuPrimitive.SubContent>
>(({ className, ...props }, ref) => (
  <DropdownMenuPrimitive.SubContent
    ref={ref}
    className={cn(
      "z-50 min-w-[8rem] overflow-hidden rounded-md border bg-popover p-1
text-popover-foreground shadow-lg data-[state=open]:animate-in
data-[state=closed]:animate-out data-[state=closed]:fade-out-0
data-[state=open]:fade-in-0 data-[state=closed]:zoom-out-95
data-[state=open]:zoom-in-95 data-[side=bottom]:slide-in-from-top-2
data-[side=left]:slide-in-from-right-2 data-[side=right]:slide-in-from-left-2
data-[side=top]:slide-in-from-bottom-2",
      className
    )}
```

```
    {...props}
  />
))
DropdownMenuSubContent.displayName =
  DropdownMenuPrimitive.SubContent.displayName

const DropdownMenuContent = React.forwardRef<
  React.ElementRef<typeof DropdownMenuPrimitive.Content>,
  React.ComponentPropsWithoutRef<typeof DropdownMenuPrimitive.Content>
>(({ className, sideOffset = 4, ...props }, ref) => (
  <DropdownMenuPrimitive.Portal>
    <DropdownMenuPrimitive.Content
      ref={ref}
      sideOffset={sideOffset}
      className={cn(
        "z-50 min-w-[8rem] overflow-hidden rounded-md border bg-popover p-1
text-popover-foreground shadow-md data-[state=open]:animate-in
data-[state=closed]:animate-out data-[state=closed]:fade-out-0
data-[state=open]:fade-in-0 data-[state=closed]:zoom-out-95
data-[state=open]:zoom-in-95 data-[side=bottom]:slide-in-from-top-2
data-[side=left]:slide-in-from-right-2 data-[side=right]:slide-in-from-left-2
data-[side=top]:slide-in-from-bottom-2",
        className
      )}
      {...props}
    />
  </DropdownMenuPrimitive.Portal>
))
DropdownMenuContent.displayName = DropdownMenuPrimitive.Content.displayName

const DropdownMenuItem = React.forwardRef<
  React.ElementRef<typeof DropdownMenuPrimitive.Item>,
  React.ComponentPropsWithoutRef<typeof DropdownMenuPrimitive.Item> & {
    inset?: boolean
  }
>(({ className, inset, ...props }, ref) => (
  <DropdownMenuPrimitive.Item
    ref={ref}
    className={cn(
```

```
      "relative flex cursor-default select-none items-center gap-2 rounded-sm px-2 py-1.5
text-sm outline-none transition-colors focus:bg-accent focus:text-accent-foreground
data-[disabled]:pointer-events-none data-[disabled]:opacity-50
[&_svg]:pointer-events-none [&_svg]:size-4 [&_svg]:shrink-0",
      inset && "pl-8",
      className
    )}
    {...props}
  />
))
DropdownMenuItem.displayName = DropdownMenuPrimitive.Item.displayName

const DropdownMenuCheckboxItem = React.forwardRef<
  React.ElementRef<typeof DropdownMenuPrimitive.CheckboxItem>,
  React.ComponentPropsWithoutRef<typeof DropdownMenuPrimitive.CheckboxItem>
>(({ className, children, checked, ...props }, ref) => (
  <DropdownMenuPrimitive.CheckboxItem
    ref={ref}
    className={cn(
      "relative flex cursor-default select-none items-center rounded-sm py-1.5 pl-8 pr-2
text-sm outline-none transition-colors focus:bg-accent focus:text-accent-foreground
data-[disabled]:pointer-events-none data-[disabled]:opacity-50",
      className
    )}
    checked={checked}
    {...props}
  >
    <span classNAme="absolute left-2 flex h-3.5 w-3.5 items-center justify-center">
      <DropdownMenuPrimitive.ItemIndicator>
        <Check className="h-4 w-4" />
      </DropdownMenuPrimitive.ItemIndicator>
    </span>
    {children}
  </DropdownMenuPrimitive.CheckboxItem>
))
DropdownMenuCheckboxItem.displayName =
  DropdownMenuPrimitive.CheckboxItem.displayName

const DropdownMenuRadioItem = React.forwardRef<
  React.ElementRef<typeof DropdownMenuPrimitive.RadioItem>,
```

```
  React.ComponentPropsWithoutRef<typeof DropdownMenuPrimitive.RadioItem>
>(({ className, children, ...props }, ref) => (
  <DropdownMenuPrimitive.RadioItem
    ref={ref}
    className={cn(
      "relative flex cursor-default select-none items-center rounded-sm py-1.5 pl-8 pr-2
text-sm outline-none transition-colors focus:bg-accent focus:text-accent-foreground
data-[disabled]:pointer-events-none data-[disabled]:opacity-50",
      className
    )}
    {...props}
  >
    <span className="absolute left-2 flex h-3.5 w-3.5 items-center justify-center">
      <DropdownMenuPrimitive.ItemIndicator>
        <Circle className="h-2 w-2 fill-current" />
      </DropdownMenuPrimitive.ItemIndicator>
    </span>
    {children}
  </DropdownMenuPrimitive.RadioItem>
))
DropdownMenuRadioItem.displayName =
DropdownMenuPrimitive.RadioItem.displayName

const DropdownMenuLabel = React.forwardRef<
  React.ElementRef<typeof DropdownMenuPrimitive.Label>,
  React.ComponentPropsWithoutRef<typeof DropdownMenuPrimitive.Label> & {
    inset?: boolean
  }
>(({ className, inset, ...props }, ref) => (
  <DropdownMenuPrimitive.Label
    ref={ref}
    className={cn(
      "px-2 py-1.5 text-sm font-semibold",
      inset && "pl-8",
      className
    )}
    {...props}
  />
))
DropdownMenuLabel.displayName = DropdownMenuPrimitive.Label.displayName
```

```
const DropdownMenuSeparator = React.forwardRef<
  React.ElementRef<typeof DropdownMenuPrimitive.Separator>,
  React.ComponentPropsWithoutRef<typeof DropdownMenuPrimitive.Separator>
>(({ className, ...props }, ref) => (
  <DropdownMenuPrimitive.Separator
    ref={ref}
    className={cn("-mx-1 my-1 h-px bg-muted", className)}
    {...props}
  />
))
DropdownMenuSeparator.displayName =
DropdownMenuPrimitive.Separator.displayName

const DropdownMenuShortcut = ({
  className,
  ...props
}: React.HTMLAttributes<HTMLSpanElement>) => {
  return (
    <span
      className={cn("ml-auto text-xs tracking-widest opacity-60", className)}
      {...props}
    />
  )
}
DropdownMenuShortcut.displayName = "DropdownMenuShortcut"

export {
  DropdownMenu,
  DropdownMenuTrigger,
  DropdownMenuContent,
  DropdownMenuItem,
  DropdownMenuCheckboxItem,
  DropdownMenuRadioItem,
  DropdownMenuLabel,
  DropdownMenuSeparator,
  DropdownMenuShortcut,
  DropdownMenuGroup,
  DropdownMenuPortal,
  DropdownMenuSub,
```

```
  DropdownMenuSubContent,
  DropdownMenuSubTrigger,
  DropdownMenuRadioGroup,
}


File: src\components\ui\input.tsx
---------------------

import * as React from "react"

import { cn } from "@/lib/utils"

const Input = React.forwardRef<HTMLInputElement,
React.ComponentProps<"input">>(
  ({ className, type, ...props }, ref) => {
    return (
      <input
        type={type}
        className={cn(
          "flex h-10 w-full rounded-md border border-input bg-background px-3 py-2
text-base ring-offset-background file:border-0 file:bg-transparent file:text-sm
file:font-medium file:text-foreground placeholder:text-muted-foreground
focus-visible:outline-none focus-visible:ring-2 focus-visible:ring-ring
focus-visible:ring-offset-2 disabled:cursor-not-allowed disabled:opacity-50 md:text-sm",
          className
        )}
        ref={ref}
        {...props}
      />
    )
  }
)
Input.displayName = "Input"

export { Input }


File: src\components\ui\label.tsx
---------------------
```

```
"use client"

import * as React from "react"
import * as LabelPrimitive from "@radix-ui/react-label"
import { cva, type VariantProps } from "class-variance-authority"

import { cn } from "@/lib/utils"

const labelVariants = cva(
  "text-sm font-medium leading-none peer-disabled:cursor-not-allowed
peer-disabled:opacity-70"
)

const Label = React.forwardRef<
  React.ElementRef<typeof LabelPrimitive.Root>,
  React.ComponentPropsWithoutRef<typeof LabelPrimitive.Root> &
    VariantProps<typeof labelVariants>
>(({ className, ...props }, ref) => (
  <LabelPrimitive.Root
    ref={ref}
    className={cn(labelVariants(), className)}
    {...props}
  />
))
Label.displayName = LabelPrimitive.Root.displayName

export { Label }


File: src\components\ui\resizable.tsx
--------------------

"use client"

import { GripVertical } from "lucide-react"
import * as ResizablePrimitive from "react-resizable-panels"

import { cn } from "@/lib/utils"
```

```
const ResizablePanelGroup = ({
  className,
  ...props
}: React.ComponentProps<typeof ResizablePrimitive.PanelGroup>) => (
  <ResizablePrimitive.PanelGroup
    className={cn(
      "flex h-full w-full data-[panel-group-direction=vertical]:flex-col",
      className
    )}
    {...props}
  />
)

const ResizablePanel = ResizablePrimitive.Panel

const ResizableHandle = ({
  withHandle,
  className,
  ...props
}: React.ComponentProps<typeof ResizablePrimitive.PanelResizeHandle> & {
  withHandle?: boolean
}) => (
  <ResizablePrimitive.PanelResizeHandle
    className={cn(
      "relative flex w-px items-center justify-center bg-border after:absolute after:inset-y-0
after:left-1/2 after:w-1 after:-translate-x-1/2 focus-visible:outline-none focus-visible:ring-1
focus-visible:ring-ring focus-visible:ring-offset-1
data-[panel-group-direction=vertical]:h-px data-[panel-group-direction=vertical]:w-full
data-[panel-group-direction=vertical]:after:left-0
data-[panel-group-direction=vertical]:after:h-1
data-[panel-group-direction=vertical]:after:w-full
data-[panel-group-direction=vertical]:after:-translate-y-1/2
data-[panel-group-direction=vertical]:after:translate-x-0
[&[data-panel-group-direction=vertical]>div]:rotate-90",
      className
    )}
    {...props}
  >
    {withHandle && (
```

```
      <div className="z-10 flex h-4 w-3 items-center justify-center rounded-sm border
bg-border">
        <GripVertical className="h-2.5 w-2.5" />
      </div>
    )}
  </ResizablePrimitive.PanelResizeHandle>
)

export { ResizablePanelGroup, ResizablePanel, ResizableHandle }


File: src\components\ui\scroll-area.tsx
--------------------

"use client"

import * as React from "react"
import * as ScrollAreaPrimitive from "@radix-ui/react-scroll-area"

import { cn } from "@/lib/utils"

const ScrollArea = React.forwardRef<
  React.ElementRef<typeof ScrollAreaPrimitive.Root>,
  React.ComponentPropsWithoutRef<typeof ScrollAreaPrimitive.Root>
>(({ className, children, ...props }, ref) => (
  <ScrollAreaPrimitive.Root
    ref={ref}
    className={cn("relative overflow-hidden", className)}
    {...props}
  >
    <ScrollAreaPrimitive.Viewport className="h-full w-full rounded-[inherit]">
      {children}
    </ScrollAreaPrimitive.Viewport>
    <ScrollBar />
    <ScrollAreaPrimitive.Corner />
  </ScrollAreaPrimitive.Root>
))
ScrollArea.displayName = ScrollAreaPrimitive.Root.displayName

const ScrollBar = React.forwardRef<
```

```
  React.ElementRef<typeof ScrollAreaPrimitive.ScrollAreaScrollbar>,
  React.ComponentPropsWithoutRef<typeof ScrollAreaPrimitive.ScrollAreaScrollbar>
>(({ className, orientation = "vertical", ...props }, ref) => (
  <ScrollAreaPrimitive.ScrollAreaScrollbar
    ref={ref}
    orientation={orientation}
    className={cn(
      "flex touch-none select-none transition-colors",
      orientation === "vertical" &&
        "h-full w-2.5 border-l border-l-transparent p-[1px]",
      orientation === "horizontal" &&
        "h-2.5 flex-col border-t border-t-transparent p-[1px]",
      className
    )}
    {...props}
  >
    <ScrollAreaPrimitive.ScrollAreaThumb className="relative flex-1 rounded-full
bg-border" />
  </ScrollAreaPrimitive.ScrollAreaScrollbar>
))
ScrollBar.displayName = ScrollAreaPrimitive.ScrollAreaScrollbar.displayName

export { ScrollArea, ScrollBar }


File: src\components\ui\separator.tsx
--------------------

"use client"

import * as React from "react"
import * as SeparatorPrimitive from "@radix-ui/react-separator"

import { cn } from "@/lib/utils"

const Separator = React.forwardRef<
  React.ElementRef<typeof SeparatorPrimitive.Root>,
  React.ComponentPropsWithoutRef<typeof SeparatorPrimitive.Root>
>(
  (
```

```
    { className, orientation = "horizontal", decorative = true, ...props },
    ref
  ) => (
    <SeparatorPrimitive.Root
      ref={ref}
      decorative={decorative}
      orientation={orientation}
      className={cn(
        "shrink-0 bg-border",
        orientation === "horizontal" ? "h-[1px] w-full" : "h-full w-[1px]",
        className
      )}
      {...props}
    />
  )
)
Separator.displayName = SeparatorPrimitive.Root.displayName

export { Separator }
```

File: src\components\ui\sheet.tsx

```
"use client"

import * as React from "react"
import * as SheetPrimitive from "@radix-ui/react-dialog"
import { cva, type VariantProps } from "class-variance-authority"
import { X } from "lucide-react"

import { cn } from "@/lib/utils"

const Sheet = SheetPrimitive.Root

const SheetTrigger = SheetPrimitive.Trigger

const SheetClose = SheetPrimitive.Close

const SheetPortal = SheetPrimitive.Portal
```

```
const SheetOverlay = React.forwardRef<
  React.ElementRef<typeof SheetPrimitive.Overlay>,
  React.ComponentPropsWithoutRef<typeof SheetPrimitive.Overlay>
>(({ className, ...props }, ref) => (
  <SheetPrimitive.Overlay
    className={cn(
      "fixed inset-0 z-50 bg-black/80  data-[state=open]:animate-in
data-[state=closed]:animate-out data-[state=closed]:fade-out-0
data-[state=open]:fade-in-0",
      className
    )}
    {...props}
    ref={ref}
  />
))
SheetOverlay.displayName = SheetPrimitive.Overlay.displayName

const sheetVariants = cva(
  "fixed z-50 gap-4 bg-background p-6 shadow-lg transition ease-in-out
data-[state=open]:animate-in data-[state=closed]:animate-out
data-[state=closed]:duration-300 data-[state=open]:duration-500",
  {
    variants: {
      side: {
        top: "inset-x-0 top-0 border-b data-[state=closed]:slide-out-to-top
data-[state=open]:slide-in-from-top",
        bottom:
          "inset-x-0 bottom-0 border-t data-[state=closed]:slide-out-to-bottom
data-[state=open]:slide-in-from-bottom",
        left: "inset-y-0 left-0 h-full w-3/4 border-r data-[state=closed]:slide-out-to-left
data-[state=open]:slide-in-from-left sm:max-w-sm",
        right:
          "inset-y-0 right-0 h-full w-3/4  border-l data-[state=closed]:slide-out-to-right
data-[state=open]:slide-in-from-right sm:max-w-sm",
      },
    },
    defaultVariants: {
      side: "right",
    },
```

```
  }
)

interface SheetContentProps
  extends React.ComponentPropsWithoutRef<typeof SheetPrimitive.Content>,
    VariantProps<typeof sheetVariants> {}

const SheetContent = React.forwardRef<
  React.ElementRef<typeof SheetPrimitive.Content>,
  SheetContentProps
>(({ side = "right", className, children, ...props }, ref) => (
  <SheetPortal>
    <SheetOverlay />
    <SheetPrimitive.Content
      ref={ref}
      className={cn(sheetVariants({ side }), className)}
      {...props}
    >
      {children}
      <SheetPrimitive.Close classNameabsolute right-4 top-4 rounded-sm opacity-70
ring-offset-background transition-opacity hover:opacity-100 focus:outline-none
focus:ring-2 focus:ring-ring focus:ring-offset-2 disabled:pointer-events-none
data-[state=open]:bg-secondary">
        <X classNameh-4 w-4" />
        <span classNamesr-only">Close</span>
      </SheetPrimitive.Close>
    </SheetPrimitive.Content>
  </SheetPortal>
))
SheetContent.displayName = SheetPrimitive.Content.displayName

const SheetHeader = ({
  className,
  ...props
}: React.HTMLAttributes<HTMLDivElement>) => (
  <div
    classNamecn(
      "flex flex-col space-y-2 text-center sm:text-left",
      className
    )}
```

```
    {...props}
  />
)
SheetHeader.displayName = "SheetHeader"

const SheetFooter = ({
  className,
  ...props
}: React.HTMLAttributes<HTMLDivElement>) => (
  <div
    className={cn(
      "flex flex-col-reverse sm:flex-row sm:justify-end sm:space-x-2",
      className
    )}
    {...props}
  />
)
SheetFooter.displayName = "SheetFooter"

const SheetTitle = React.forwardRef<
  React.ElementRef<typeof SheetPrimitive.Title>,
  React.ComponentPropsWithoutRef<typeof SheetPrimitive.Title>
>(({ className, ...props }, ref) => (
  <SheetPrimitive.Title
    ref={ref}
    className={cn("text-lg font-semibold text-foreground", className)}
    {...props}
  />
))
SheetTitle.displayName = SheetPrimitive.Title.displayName

const SheetDescription = React.forwardRef<
  React.ElementRef<typeof SheetPrimitive.Description>,
  React.ComponentPropsWithoutRef<typeof SheetPrimitive.Description>
>(({ className, ...props }, ref) => (
  <SheetPrimitive.Description
    ref={ref}
    className={cn("text-sm text-muted-foreground", className)}
    {...props}
  />
```

```
))
SheetDescription.displayName = SheetPrimitive.Description.displayName

export {
  Sheet,
  SheetPortal,
  SheetOverlay,
  SheetTrigger,
  SheetClose,
  SheetContent,
  SheetHeader,
  SheetFooter,
  SheetTitle,
  SheetDescription,
}
```

File: src\components\ui\sidebar.tsx

```
"use client"

import * as React from "react"
import { Slot } from "@radix-ui/react-slot"
import { VariantProps, cva } from "class-variance-authority"
import { PanelLeft } from "lucide-react"

import { useIsMobile } from "@/hooks/use-mobile"
import { cn } from "@/lib/utils"
import { Button } from "@/components/ui/button"
import { Input } from "@/components/ui/input"
import { Separator } from "@/components/ui/separator"
import { Sheet, SheetContent } from "@/components/ui/sheet"
import { Skeleton } from "@/components/ui/skeleton"
import {
  Tooltip,
  TooltipContent,
  TooltipProvider,
  TooltipTrigger,
} from "@/components/ui/tooltip"
```

```
const SIDEBAR_COOKIE_NAME = "sidebar:state"
const SIDEBAR_COOKIE_MAX_AGE = 60 * 60 * 24 * 7
const SIDEBAR_WIDTH = "16rem"
const SIDEBAR_WIDTH_MOBILE = "18rem"
const SIDEBAR_WIDTH_ICON = "3rem"
const SIDEBAR_KEYBOARD_SHORTCUT = "b"

type SidebarContext = {
  state: "expanded" | "collapsed"
  open: boolean
  setOpen: (open: boolean) => void
  openMobile: boolean
  setOpenMobile: (open: boolean) => void
  isMobile: boolean
  toggleSidebar: () => void
}

const SidebarContext = React.createContext<SidebarContext | null>(null)

function useSidebar() {
  const context = React.useContext(SidebarContext)
  if (!context) {
    throw new Error("useSidebar must be used within a SidebarProvider.")
  }

  return context
}

const SidebarProvider = React.forwardRef<
  HTMLDivElement,
  React.ComponentProps<"div"> & {
    defaultOpen?: boolean
    open?: boolean
    onOpenChange?: (open: boolean) => void
  }
>(
  (
    {
      defaultOpen = true,
```

```
      open: openProp,
      onOpenChange: setOpenProp,
      className,
      style,
      children,
      ...props
    },
    ref
  ) => {
    const isMobile = useIsMobile()
    const [openMobile, setOpenMobile] = React.useState(false)

    // This is the internal state of the sidebar.
    // We use openProp and setOpenProp for control from outside the component.
    const [_open, _setOpen] = React.useState(defaultOpen)
    const open = openProp ?? _open
    const setOpen = React.useCallback(
      (value: boolean | ((value: boolean) => boolean)) => {
        const openState = typeof value === "function" ? value(open) : value
        if (setOpenProp) {
          setOpenProp(openState)
        } else {
          _setOpen(openState)
        }

        // This sets the cookie to keep the sidebar state.
        document.cookie = `${SIDEBAR_COOKIE_NAME}=${openState}; path=/;
max-age=${SIDEBAR_COOKIE_MAX_AGE}`
      },
      [setOpenProp, open]
    )

    // Helper to toggle the sidebar.
    const toggleSidebar = React.useCallback(() => {
      return isMobile
        ? setOpenMobile((open) => !open)
        : setOpen((open) => !open)
    }, [isMobile, setOpen, setOpenMobile])

    // Adds a keyboard shortcut to toggle the sidebar.
```

```
React.useEffect(() => {
  const handleKeyDown = (event: KeyboardEvent) => {
    if (
      event.key === SIDEBAR_KEYBOARD_SHORTCUT &&
      (event.metaKey || event.ctrlKey)
    ) {
      event.preventDefault()
      toggleSidebar()
    }
  }

  window.addEventListener("keydown", handleKeyDown)
  return () => window.removeEventListener("keydown", handleKeyDown)
}, [toggleSidebar])

// We add a state so that we can do data-state="expanded" or "collapsed".
// This makes it easier to style the sidebar with Tailwind classes.
const state = open ? "expanded" : "collapsed"

const contextValue = React.useMemo<SidebarContext>(
  () => ({
    state,
    open,
    setOpen,
    isMobile,
    openMobile,
    setOpenMobile,
    toggleSidebar,
  }),
  [state, open, setOpen, isMobile, openMobile, setOpenMobile, toggleSidebar]
)

return (
  <SidebarContext.Provider value={contextValue}>
    <TooltipProvider delayDuration={0}>
      <div
        style={
          {
            "--sidebar-width": SIDEBAR_WIDTH,
            "--sidebar-width-icon": SIDEBAR_WIDTH_ICON,
```

```
              ...style,
            } as React.CSSProperties
          }
          className={cn(
            "group/sidebar-wrapper flex min-h-svh w-full
has-[[data-variant=inset]]:bg-sidebar",
            className
          )}
          ref={ref}
          {...props}
        >
          {children}
        </div>
      </TooltipProvider>
    </SidebarContext.Provider>
  )
 }
)
SidebarProvider.displayName = "SidebarProvider"

const Sidebar = React.forwardRef<
 HTMLDivElement,
 React.ComponentProps<"div"> & {
  side?: "left" | "right"
  variant?: "sidebar" | "floating" | "inset"
  collapsible?: "offcanvas" | "icon" | "none"
 }
>(
 (
  {
   side = "left",
   variant = "sidebar",
   collapsible = "offcanvas",
   className,
   children,
   ...props
  },
  ref
 ) => {
  const { isMobile, state, openMobile, setOpenMobile } = useSidebar()
```

```
    if (collapsible === "none") {
      return (
        <div
          className={cn(
            "flex h-full w-[--sidebar-width] flex-col bg-sidebar text-sidebar-foreground",
            className
          )}
          ref={ref}
          {...props}
        >
          {children}
        </div>
      )
    }

    if (isMobile) {
      return (
        <Sheet open={openMobile} onOpenChange={setOpenMobile} {...props}>
          <SheetContent
            data-sidebar="sidebar"
            data-mobile="true"
            className="w-[--sidebar-width] bg-sidebar p-0 text-sidebar-foreground
[&>button]:hidden"
            style={
              {
                "--sidebar-width": SIDEBAR_WIDTH_MOBILE,
              } as React.CSSProperties
            }
            side={side}
          >
            <div className="flex h-full w-full flex-col">{children}</div>
          </SheetContent>
        </Sheet>
      )
    }

    return (
      <div
        ref={ref}
```

```
      className="group peer hidden md:block text-sidebar-foreground"
      data-state={state}
      data-collapsible={state === "collapsed" ? collapsible : ""}
      data-variant={variant}
      data-side={side}
    >
      {/* This is what handles the sidebar gap on desktop */}
      <div
        className={cn(
          "duration-200 relative h-svh w-[--sidebar-width] bg-transparent transition-[width]
ease-linear",
          "group-data-[collapsible=offcanvas]:w-0",
          "group-data-[side=right]:rotate-180",
          variant === "floating" || variant === "inset"
            ?
"group-data-[collapsible=icon]:w-[calc(var(--sidebar-width-icon)_+_theme(spacing.4))]"
            : "group-data-[collapsible=icon]:w-[--sidebar-width-icon]"
        )}
      />
      <div
        className={cn(
          "duration-200 fixed inset-y-0 z-10 hidden h-svh w-[--sidebar-width]
transition-[left,right,width] ease-linear md:flex",
          side === "left"
            ? "left-0 group-data-[collapsible=offcanvas]:left-[calc(var(--sidebar-width)*-1)]"
            : "right-0
group-data-[collapsible=offcanvas]:right-[calc(var(--sidebar-width)*-1)]",
          // Adjust the padding for floating and inset variants.
          variant === "floating" || variant === "inset"
            ? "p-2
group-data-[collapsible=icon]:w-[calc(var(--sidebar-width-icon)_+_theme(spacing.4)_+2
px)]"
            : "group-data-[collapsible=icon]:w-[--sidebar-width-icon]
group-data-[side=left]:border-r group-data-[side=right]:border-l",
          className
        )}
        {...props}
      >
        <div
          data-sidebar="sidebar"
```

```
            className="flex h-full w-full flex-col bg-sidebar
group-data-[variant=floating]:rounded-lg group-data-[variant=floating]:border
group-data-[variant=floating]:border-sidebar-border
group-data-[variant=floating]:shadow"
          >
            {children}
          </div>
        </div>
      </div>
    )
  }
)
Sidebar.displayName = "Sidebar"

const SidebarTrigger = React.forwardRef<
  React.ElementRef<typeof Button>,
  React.ComponentProps<typeof Button>
>(({ className, onClick, ...props }, ref) => {
  const { toggleSidebar } = useSidebar()

  return (
    <Button
      ref={ref}
      data-sidebar="trigger"
      variant="ghost"
      size="icon"
      className={cn("h-7 w-7", className)}
      onClick={(event) => {
        onClick?.(event)
        toggleSidebar()
      }}
      {...props}
    >
      <PanelLeft />
      <span className="sr-only">Toggle Sidebar</span>
    </Button>
  )
})
SidebarTrigger.displayName = "SidebarTrigger"
```

```
const SidebarRail = React.forwardRef<
  HTMLButtonElement,
  React.ComponentProps<"button">
>(({ className, ...props }, ref) => {
  const { toggleSidebar } = useSidebar()

  return (
    <button
      ref={ref}
      data-sidebar="rail"
      aria-label="Toggle Sidebar"
      tabIndex={-1}
      onClick={toggleSidebar}
      title="Toggle Sidebar"
      className={cn(
        "absolute inset-y-0 z-20 hidden w-4 -translate-x-1/2 transition-all ease-linear
after:absolute after:inset-y-0 after:left-1/2 after:w-[2px] hover:after:bg-sidebar-border
group-data-[side=left]:-right-4 group-data-[side=right]:left-0 sm:flex",
        "[[data-side=left]_&]:cursor-w-resize [[data-side=right]_&]:cursor-e-resize",
        "[[data-side=left][data-state=collapsed]_&]:cursor-e-resize
[[data-side=right][data-state=collapsed]_&]:cursor-w-resize",
        "group-data-[collapsible=offcanvas]:translate-x-0
group-data-[collapsible=offcanvas]:after:left-full
group-data-[collapsible=offcanvas]:hover:bg-sidebar",
        "[[data-side=left][data-collapsible=offcanvas]_&]:-right-2",
        "[[data-side=right][data-collapsible=offcanvas]_&]:-left-2",
        className
      )}
      {...props}
    />
  )
})
SidebarRail.displayName = "SidebarRail"

const SidebarInset = React.forwardRef<
  HTMLDivElement,
  React.ComponentProps<"main">
>(({ className, ...props }, ref) => {
  return (
    <main
```

```
      ref={ref}
      className={cn(
        "relative flex min-h-svh flex-1 flex-col bg-background",
        "peer-data-[variant=inset]:min-h-[calc(100svh-theme(spacing.4))]
md:peer-data-[variant=inset]:m-2
md:peer-data-[state=collapsed]:peer-data-[variant=inset]:ml-2
md:peer-data-[variant=inset]:ml-0 md:peer-data-[variant=inset]:rounded-xl
md:peer-data-[variant=inset]:shadow",
        className
      )}
      {...props}
    />
  )
})
SidebarInset.displayName = "SidebarInset"

const SidebarInput = React.forwardRef<
  React.ElementRef<typeof Input>,
  React.ComponentProps<typeof Input>
>(({ className, ...props }, ref) => {
  return (
    <Input
      ref={ref}
      data-sidebar="input"
      className={cn(
        "h-8 w-full bg-background shadow-none focus-visible:ring-2
focus-visible:ring-sidebar-ring",
        className
      )}
      {...props}
    />
  )
})
SidebarInput.displayName = "SidebarInput"

const SidebarHeader = React.forwardRef<
  HTMLDivElement,
  React.ComponentProps<"div">
>(({ className, ...props }, ref) => {
  return (
```

```
    <div
      ref={ref}
      data-sidebar="header"
      className={cn("flex flex-col gap-2 p-2", className)}
      {...props}
    />
  )
})
SidebarHeader.displayName = "SidebarHeader"

const SidebarFooter = React.forwardRef<
  HTMLDivElement,
  React.ComponentProps<"div">
>(({ className, ...props }, ref) => {
  return (
    <div
      ref={ref}
      data-sidebar="footer"
      className={cn("flex flex-col gap-2 p-2", className)}
      {...props}
    />
  )
})
SidebarFooter.displayName = "SidebarFooter"

const SidebarSeparator = React.forwardRef<
  React.ElementRef<typeof Separator>,
  React.ComponentProps<typeof Separator>
>(({ className, ...props }, ref) => {
  return (
    <Separator
      ref={ref}
      data-sidebar="separator"
      className={cn("mx-2 w-auto bg-sidebar-border", className)}
      {...props}
    />
  )
})
SidebarSeparator.displayName = "SidebarSeparator"
```

```
const SidebarContent = React.forwardRef<
  HTMLDivElement,
  React.ComponentProps<"div">
>(({ className, ...props }, ref) => {
  return (
    <div
      ref={ref}
      data-sidebar="content"
      className={cn(
        "flex min-h-0 flex-1 flex-col gap-2 overflow-auto
group-data-[collapsible=icon]:overflow-hidden",
        className
      )}
      {...props}
    />
  )
})
SidebarContent.displayName = "SidebarContent"

const SidebarGroup = React.forwardRef<
  HTMLDivElement,
  React.ComponentProps<"div">
>(({ className, ...props }, ref) => {
  return (
    <div
      ref={ref}
      data-sidebar="group"
      className={cn("relative flex w-full min-w-0 flex-col p-2", className)}
      {...props}
    />
  )
})
SidebarGroup.displayName = "SidebarGroup"

const SidebarGroupLabel = React.forwardRef<
  HTMLDivElement,
  React.ComponentProps<"div"> & { asChild?: boolean }
>(({ className, asChild = false, ...props }, ref) => {
  const Comp = asChild ? Slot : "div"
```

```
  return (
    <Comp
      ref={ref}
      data-sidebar="group-label"
      className={cn(
        "duration-200 flex h-8 shrink-0 items-center rounded-md px-2 text-xs font-medium
text-sidebar-foreground/70 outline-none ring-sidebar-ring transition-[margin,opa]
ease-linear focus-visible:ring-2 [&>svg]:size-4 [&>svg]:shrink-0",
        "group-data-[collapsible=icon]:-mt-8 group-data-[collapsible=icon]:opacity-0",
        className
      )}
      {...props}
    />
  )
})
SidebarGroupLabel.displayName = "SidebarGroupLabel"

const SidebarGroupAction = React.forwardRef<
  HTMLButtonElement,
  React.ComponentProps<"button"> & { asChild?: boolean }
>(({ className, asChild = false, ...props }, ref) => {
  const Comp = asChild ? Slot : "button"

  return (
    <Comp
      ref={ref}
      data-sidebar="group-action"
      className={cn(
        "absolute right-3 top-3.5 flex aspect-square w-5 items-center justify-center
rounded-md p-0 text-sidebar-foreground outline-none ring-sidebar-ring
transition-transform hover:bg-sidebar-accent hover:text-sidebar-accent-foreground
focus-visible:ring-2 [&>svg]:size-4 [&>svg]:shrink-0",
        // Increases the hit area of the button on mobile.
        "after:absolute after:-inset-2 after:md:hidden",
        "group-data-[collapsible=icon]:hidden",
        className
      )}
      {...props}
    />
  )
```

```
})
SidebarGroupAction.displayName = "SidebarGroupAction"

const SidebarGroupContent = React.forwardRef<
  HTMLDivElement,
  React.ComponentProps<"div">
>(({ className, ...props }, ref) => (
  <div
    ref={ref}
    data-sidebar="group-content"
    className={cn("w-full text-sm", className)}
    {...props}
  />
))
SidebarGroupContent.displayName = "SidebarGroupContent"

const SidebarMenu = React.forwardRef<
  HTMLUListElement,
  React.ComponentProps<"ul">
>(({ className, ...props }, ref) => (
  <ul
    ref={ref}
    data-sidebar="menu"
    className={cn("flex w-full min-w-0 flex-col gap-1", className)}
    {...props}
  />
))
SidebarMenu.displayName = "SidebarMenu"

const SidebarMenuItem = React.forwardRef<
  HTMLLIElement,
  React.ComponentProps<"li">
>(({ className, ...props }, ref) => (
  <li
    ref={ref}
    data-sidebar="menu-item"
    className={cn("group/menu-item relative", className)}
    {...props}
  />
))
```

```
SidebarMenuItem.displayName = "SidebarMenuItem"

const sidebarMenuButtonVariants = cva(
  "peer/menu-button flex w-full items-center gap-2 overflow-hidden rounded-md p-2
text-left text-sm outline-none ring-sidebar-ring transition-[width,height,padding]
hover:bg-sidebar-accent hover:text-sidebar-accent-foreground focus-visible:ring-2
active:bg-sidebar-accent active:text-sidebar-accent-foreground
disabled:pointer-events-none disabled:opacity-50
group-has-[[data-sidebar=menu-action]]/menu-item:pr-8
aria-disabled:pointer-events-none aria-disabled:opacity-50
data-[active=true]:bg-sidebar-accent data-[active=true]:font-medium
data-[active=true]:text-sidebar-accent-foreground
data-[state=open]:hover:bg-sidebar-accent
data-[state=open]:hover:text-sidebar-accent-foreground
group-data-[collapsible=icon]:!size-8 group-data-[collapsible=icon]:!p-2
[&>span:last-child]:truncate [&>svg]:size-4 [&>svg]:shrink-0",
  {
    variants: {
      variant: {
        default: "hover:bg-sidebar-accent hover:text-sidebar-accent-foreground",
        outline:
          "bg-background shadow-[0_0_0_1px_hsl(var(--sidebar-border))]
hover:bg-sidebar-accent hover:text-sidebar-accent-foreground
hover:shadow-[0_0_0_1px_hsl(var(--sidebar-accent))]",
      },
      size: {
        default: "h-8 text-sm",
        sm: "h-7 text-xs",
        lg: "h-12 text-sm group-data-[collapsible=icon]:!p-0",
      },
    },
    defaultVariants: {
      variant: "default",
      size: "default",
    },
  }
)

const SidebarMenuButton = React.forwardRef<
  HTMLButtonElement,
```

```
  React.ComponentProps<"button"> & {
    asChild?: boolean
    isActive?: boolean
    tooltip?: string | React.ComponentProps<typeof TooltipContent>
  } & VariantProps<typeof sidebarMenuButtonVariants>
>(
  (
    {
      asChild = false,
      isActive = false,
      variant = "default",
      size = "default",
      tooltip,
      className,
      ...props
    },
    ref
  ) => {
    const Comp = asChild ? Slot : "button"
    const { isMobile, state } = useSidebar()

    const button = (
      <Comp
        ref={ref}
        data-sidebar="menu-button"
        data-size={size}
        data-active={isActive}
        className={cn(sidebarMenuButtonVariants({ variant, size }), className)}
        {...props}
      />
    )

    if (!tooltip) {
      return button
    }

    if (typeof tooltip === "string") {
      tooltip = {
        children: tooltip,
      }
```

```
    }

    return (
      <Tooltip>
        <TooltipTrigger asChild>{button}</TooltipTrigger>
        <TooltipContent
          side="right"
          align="center"
          hidden={state !== "collapsed" || isMobile}
          {...tooltip}
        />
      </Tooltip>
    )
  }
)
SidebarMenuButton.displayName = "SidebarMenuButton"

const SidebarMenuAction = React.forwardRef<
  HTMLButtonElement,
  React.ComponentProps<"button"> & {
    asChild?: boolean
    showOnHover?: boolean
  }
>(({ className, asChild = false, showOnHover = false, ...props }, ref) => {
  const Comp = asChild ? Slot : "button"

  return (
    <Comp
      ref={ref}
      data-sidebar="menu-action"
      className={cn(
        "absolute right-1 top-1.5 flex aspect-square w-5 items-center justify-center
rounded-md p-0 text-sidebar-foreground outline-none ring-sidebar-ring
transition-transform hover:bg-sidebar-accent hover:text-sidebar-accent-foreground
focus-visible:ring-2 peer-hover/menu-button:text-sidebar-accent-foreground
[&>svg]:size-4 [&>svg]:shrink-0",
        // Increases the hit area of the button on mobile.
        "after:absolute after:-inset-2 after:md:hidden",
        "peer-data-[size=sm]/menu-button:top-1",
        "peer-data-[size=default]/menu-button:top-1.5",
```

```
        "peer-data-[size=lg]/menu-button:top-2.5",
        "group-data-[collapsible=icon]:hidden",
        showOnHover &&
          "group-focus-within/menu-item:opacity-100 group-hover/menu-item:opacity-100
data-[state=open]:opacity-100
peer-data-[active=true]/menu-button:text-sidebar-accent-foreground md:opacity-0",
        className
      )}
      {...props}
    />
  )
})
SidebarMenuAction.displayName = "SidebarMenuAction"

const SidebarMenuBadge = React.forwardRef<
  HTMLDivElement,
  React.ComponentProps<"div">
>(({ className, ...props }, ref) => (
  <div
    ref={ref}
    data-sidebar="menu-badge"
    className={cn(
      "absolute right-1 flex h-5 min-w-5 items-center justify-center rounded-md px-1
text-xs font-medium tabular-nums text-sidebar-foreground select-none
pointer-events-none",
      "peer-hover/menu-button:text-sidebar-accent-foreground
peer-data-[active=true]/menu-button:text-sidebar-accent-foreground",
      "peer-data-[size=sm]/menu-button:top-1",
      "peer-data-[size=default]/menu-button:top-1.5",
      "peer-data-[size=lg]/menu-button:top-2.5",
      "group-data-[collapsible=icon]:hidden",
      className
    )}
    {...props}
  />
))
SidebarMenuBadge.displayName = "SidebarMenuBadge"

const SidebarMenuSkeleton = React.forwardRef<
  HTMLDivElement,
```

```
  React.ComponentProps<"div"> & {
    showIcon?: boolean
  }
>(({ className, showIcon = false, ...props }, ref) => {
  // Random width between 50 to 90%.
  const width = React.useMemo(() => {
    return `${Math.floor(Math.random() * 40) + 50}%`
  }, [])

  return (
    <div
      ref={ref}
      data-sidebar="menu-skeleton"
      className={cn("rounded-md h-8 flex gap-2 px-2 items-center", className)}
      {...props}
    >
      {showIcon && (
        <Skeleton
          className="size-4 rounded-md"
          data-sidebar="menu-skeleton-icon"
        />
      )}
      <Skeleton
        className="h-4 flex-1 max-w-[--skeleton-width]"
        data-sidebar="menu-skeleton-text"
        style={
          {
            "--skeleton-width": width,
          } as React.CSSProperties
        }
      />
    </div>
  )
})
SidebarMenuSkeleton.displayName = "SidebarMenuSkeleton"

const SidebarMenuSub = React.forwardRef<
  HTMLUListElement,
  React.ComponentProps<"ul">
>(({ className, ...props }, ref) => (
```

```
  <ul
    ref={ref}
    data-sidebar="menu-sub"
    className={cn(
      "mx-3.5 flex min-w-0 translate-x-px flex-col gap-1 border-l border-sidebar-border
px-2.5 py-0.5",
      "group-data-[collapsible=icon]:hidden",
      className
    )}
    {...props}
  />
))
SidebarMenuSub.displayName = "SidebarMenuSub"

const SidebarMenuSubItem = React.forwardRef<
  HTMLLIElement,
  React.ComponentProps<"li">
>(({ ...props }, ref) => <li ref={ref} {...props} />)
SidebarMenuSubItem.displayName = "SidebarMenuSubItem"

const SidebarMenuSubButton = React.forwardRef<
  HTMLAnchorElement,
  React.ComponentProps<"a"> & {
    asChild?: boolean
    size?: "sm" | "md"
    isActive?: boolean
  }
>(({ asChild = false, size = "md", isActive, className, ...props }, ref) => {
  const Comp = asChild ? Slot : "a"

  return (
    <Comp
      ref={ref}
      data-sidebar="menu-sub-button"
      data-size={size}
      data-active={isActive}
      className={cn(
        "flex h-7 min-w-0 -translate-x-px items-center gap-2 overflow-hidden rounded-md
px-2 text-sidebar-foreground outline-none ring-sidebar-ring hover:bg-sidebar-accent
hover:text-sidebar-accent-foreground focus-visible:ring-2 active:bg-sidebar-accent
```

```
active:text-sidebar-accent-foreground disabled:pointer-events-none disabled:opacity-50
aria-disabled:pointer-events-none aria-disabled:opacity-50 [&>span:last-child]:truncate
[&>svg]:size-4 [&>svg]:shrink-0 [&>svg]:text-sidebar-accent-foreground",
        "data-[active=true]:bg-sidebar-accent
data-[active=true]:text-sidebar-accent-foreground",
        size === "sm" && "text-xs",
        size === "md" && "text-sm",
        "group-data-[collapsible=icon]:hidden",
        className
      )}
      {...props}
    />
  )
})
SidebarMenuSubButton.displayName = "SidebarMenuSubButton"

export {
  Sidebar,
  SidebarContent,
  SidebarFooter,
  SidebarGroup,
  SidebarGroupAction,
  SidebarGroupContent,
  SidebarGroupLabel,
  SidebarHeader,
  SidebarInput,
  SidebarInset,
  SidebarMenu,
  SidebarMenuAction,
  SidebarMenuBadge,
  SidebarMenuButton,
  SidebarMenuItem,
  SidebarMenuSkeleton,
  SidebarMenuSub,
  SidebarMenuSubButton,
  SidebarMenuSubItem,
  SidebarProvider,
  SidebarRail,
  SidebarSeparator,
  SidebarTrigger,
```

```
  useSidebar,
}


File: src\components\ui\skeleton.tsx
--------------------

import { cn } from "@/lib/utils"

function Skeleton({
  className,
  ...props
}: React.HTMLAttributes<HTMLDivElement>) {
  return (
    <div
      className={cn("animate-pulse rounded-md bg-muted", className)}
      {...props}
    />
  )
}

export { Skeleton }


File: src\components\ui\switch.tsx
--------------------

"use client"

import * as React from "react"
import * as SwitchPrimitives from "@radix-ui/react-switch"

import { cn } from "@/lib/utils"

const Switch = React.forwardRef<
  React.ElementRef<typeof SwitchPrimitives.Root>,
  React.ComponentPropsWithoutRef<typeof SwitchPrimitives.Root>
>(({ className, ...props }, ref) => (
  <SwitchPrimitives.Root
    className={cn(
```

```
      "peer inline-flex h-6 w-11 shrink-0 cursor-pointer items-center rounded-full border-2
border-transparent transition-colors focus-visible:outline-none focus-visible:ring-2
focus-visible:ring-ring focus-visible:ring-offset-2 focus-visible:ring-offset-background
disabled:cursor-not-allowed disabled:opacity-50 data-[state=checked]:bg-primary
data-[state=unchecked]:bg-input",
      className
    )}
    {...props}
    ref={ref}
  >
    <SwitchPrimitives.Thumb
      className={cn(
        "pointer-events-none block h-5 w-5 rounded-full bg-background shadow-lg ring-0
transition-transform data-[state=checked]:translate-x-5
data-[state=unchecked]:translate-x-0"
      )}
    />
  </SwitchPrimitives.Root>
))
Switch.displayName = SwitchPrimitives.Root.displayName

export { Switch }


File: src\components\ui\tabs.tsx
--------------------

"use client"

import * as React from "react"
import * as TabsPrimitive from "@radix-ui/react-tabs"

import { cn } from "@/lib/utils"

const Tabs = TabsPrimitive.Root

const TabsList = React.forwardRef<
  React.ElementRef<typeof TabsPrimitive.List>,
  React.ComponentPropsWithoutRef<typeof TabsPrimitive.List>
>(({ className, ...props }, ref) => (
```

```
  <TabsPrimitive.List
    ref={ref}
    className={cn(
      "inline-flex h-10 items-center justify-center rounded-md bg-muted p-1
text-muted-foreground",
      className
    )}
    {...props}
  />
))
TabsList.displayName = TabsPrimitive.List.displayName

const TabsTrigger = React.forwardRef<
  React.ElementRef<typeof TabsPrimitive.Trigger>,
  React.ComponentPropsWithoutRef<typeof TabsPrimitive.Trigger>
>(({ className, ...props }, ref) => (
  <TabsPrimitive.Trigger
    ref={ref}
    className={cn(
      "inline-flex items-center justify-center whitespace-nowrap rounded-sm px-3 py-1.5
text-sm font-medium ring-offset-background transition-all focus-visible:outline-none
focus-visible:ring-2 focus-visible:ring-ring focus-visible:ring-offset-2
disabled:pointer-events-none disabled:opacity-50 data-[state=active]:bg-background
data-[state=active]:text-foreground data-[state=active]:shadow-sm",
      className
    )}
    {...props}
  />
))
TabsTrigger.displayName = TabsPrimitive.Trigger.displayName

const TabsContent = React.forwardRef<
  React.ElementRef<typeof TabsPrimitive.Content>,
  React.ComponentPropsWithoutRef<typeof TabsPrimitive.Content>
>(({ className, ...props }, ref) => (
  <TabsPrimitive.Content
    ref={ref}
    className={cn(
      "mt-2 ring-offset-background focus-visible:outline-none focus-visible:ring-2
focus-visible:ring-ring focus-visible:ring-offset-2",
```

```
      className
    )}
    {...props}
  />
))
TabsContent.displayName = TabsPrimitive.Content.displayName

export { Tabs, TabsList, TabsTrigger, TabsContent }


File: src\components\ui\textarea.tsx
--------------------

import * as React from "react"

import { cn } from "@/lib/utils"

const Textarea = React.forwardRef<
  HTMLTextAreaElement,
  React.ComponentProps<"textarea">
>(({ className, ...props }, ref) => {
  return (
    <textarea
      className={cn(
        "flex min-h-[80px] w-full rounded-md border border-input bg-background px-3 py-2
text-base ring-offset-background placeholder:text-muted-foreground
focus-visible:outline-none focus-visible:ring-2 focus-visible:ring-ring
focus-visible:ring-offset-2 disabled:cursor-not-allowed disabled:opacity-50 md:text-sm",
        className
      )}
      ref={ref}
      {...props}
    />
  )
})
Textarea.displayName = "Textarea"

export { Textarea }
```

File: src\components\ui\toast.tsx

```
"use client"

import * as React from "react"
import * as ToastPrimitives from "@radix-ui/react-toast"
import { cva, type VariantProps } from "class-variance-authority"
import { X } from "lucide-react"

import { cn } from "@/lib/utils"

const ToastProvider = ToastPrimitives.Provider

const ToastViewport = React.forwardRef<
  React.ElementRef<typeof ToastPrimitives.Viewport>,
  React.ComponentPropsWithoutRef<typeof ToastPrimitives.Viewport>
>(({ className, ...props }, ref) => (
  <ToastPrimitives.Viewport
    ref={ref}
    className={cn(
      "fixed top-0 z-[100] flex max-h-screen w-full flex-col-reverse p-4 sm:bottom-0
sm:right-0 sm:top-auto sm:flex-col md:max-w-[420px]",
      className
    )}
    {...props}
  />
))
ToastViewport.displayName = ToastPrimitives.Viewport.displayName

const toastVariants = cva(
  "group pointer-events-auto relative flex w-full items-center justify-between space-x-4
overflow-hidden rounded-md border p-6 pr-8 shadow-lg transition-all
data-[swipe=cancel]:translate-x-0
data-[swipe=end]:translate-x-[var(--radix-toast-swipe-end-x)]
data-[swipe=move]:translate-x-[var(--radix-toast-swipe-move-x)]
data-[swipe=move]:transition-none data-[state=open]:animate-in
data-[state=closed]:animate-out data-[swipe=end]:animate-out
data-[state=closed]:fade-out-80 data-[state=closed]:slide-out-to-right-full
data-[state=open]:slide-in-from-top-full data-[state=open]:sm:slide-in-from-bottom-full",
```

```
  {
    variants: {
      variant: {
        default: "border bg-background text-foreground",
        destructive:
          "destructive group border-destructive bg-destructive text-destructive-foreground",
      },
    },
    defaultVariants: {
      variant: "default",
    },
  }
)

const Toast = React.forwardRef<
  React.ElementRef<typeof ToastPrimitives.Root>,
  React.ComponentPropsWithoutRef<typeof ToastPrimitives.Root> &
    VariantProps<typeof toastVariants>
>(({ className, variant, ...props }, ref) => {
  return (
    <ToastPrimitives.Root
      ref={ref}
      className={cn(toastVariants({ variant }), className)}
      {...props}
    />
  )
})
Toast.displayName = ToastPrimitives.Root.displayName

const ToastAction = React.forwardRef<
  React.ElementRef<typeof ToastPrimitives.Action>,
  React.ComponentPropsWithoutRef<typeof ToastPrimitives.Action>
>(({ className, ...props }, ref) => (
  <ToastPrimitives.Action
    ref={ref}
    className={cn(
      "inline-flex h-8 shrink-0 items-center justify-center rounded-md border
bg-transparent px-3 text-sm font-medium ring-offset-background transition-colors
hover:bg-secondary focus:outline-none focus:ring-2 focus:ring-ring focus:ring-offset-2
disabled:pointer-events-none disabled:opacity-50 group-[.destructive]:border-muted/40
```

```
group-[.destructive]:hover:border-destructive/30
group-[.destructive]:hover:bg-destructive
group-[.destructive]:hover:text-destructive-foreground
group-[.destructive]:focus:ring-destructive",
      className
    )}
    {...props}
  />
))
ToastAction.displayName = ToastPrimitives.Action.displayName

const ToastClose = React.forwardRef<
  React.ElementRef<typeof ToastPrimitives.Close>,
  React.ComponentPropsWithoutRef<typeof ToastPrimitives.Close>
>(({ className, ...props }, ref) => (
  <ToastPrimitives.Close
    ref={ref}
    className={cn(
      "absolute right-2 top-2 rounded-md p-1 text-foreground/50 opacity-0
transition-opacity hover:text-foreground focus:opacity-100 focus:outline-none
focus:ring-2 group-hover:opacity-100 group-[.destructive]:text-red-300
group-[.destructive]:hover:text-red-50 group-[.destructive]:focus:ring-red-400
group-[.destructive]:focus:ring-offset-red-600",
      className
    )}
    toast-close=""
    {...props}
  >
    <X className="h-4 w-4" />
  </ToastPrimitives.Close>
))
ToastClose.displayName = ToastPrimitives.Close.displayName

const ToastTitle = React.forwardRef<
  React.ElementRef<typeof ToastPrimitives.Title>,
  React.ComponentPropsWithoutRef<typeof ToastPrimitives.Title>
>(({ className, ...props }, ref) => (
  <ToastPrimitives.Title
    ref={ref}
    className={cn("text-sm font-semibold", className)}
```

```
    {...props}
  />
))
ToastTitle.displayName = ToastPrimitives.Title.displayName

const ToastDescription = React.forwardRef<
  React.ElementRef<typeof ToastPrimitives.Description>,
  React.ComponentPropsWithoutRef<typeof ToastPrimitives.Description>
>(({ className, ...props }, ref) => (
  <ToastPrimitives.Description
    ref={ref}
    className={cn("text-sm opacity-90", className)}
    {...props}
  />
))
ToastDescription.displayName = ToastPrimitives.Description.displayName

type ToastProps = React.ComponentPropsWithoutRef<typeof Toast>

type ToastActionElement = React.ReactElement<typeof ToastAction>

export {
  type ToastProps,
  type ToastActionElement,
  ToastProvider,
  ToastViewport,
  Toast,
  ToastTitle,
  ToastDescription,
  ToastClose,
  ToastAction,
}
```

File: src\components\ui\toaster.tsx

```
"use client"

import { useToast } from "@/hooks/use-toast"
```

```
import {
  Toast,
  ToastClose,
  ToastDescription,
  ToastProvider,
  ToastTitle,
  ToastViewport,
} from "@/components/ui/toast"

export function Toaster() {
  const { toasts } = useToast()

  return (
    <ToastProvider>
      {toasts.map(function ({ id, title, description, action, ...props }) {
        return (
          <Toast key={id} {...props}>
            <div className="grid gap-1">
              {title && <ToastTitle>{title}</ToastTitle>}
              {description && (
                <ToastDescription>{description}</ToastDescription>
              )}
            </div>
            {action}
            <ToastClose />
          </Toast>
        )
      })}
      <ToastViewport />
    </ToastProvider>
  )
}
```

File: src\components\ui\tooltip.tsx

```
"use client"

import * as React from "react"
```

```
import * as TooltipPrimitive from "@radix-ui/react-tooltip"

import { cn } from "@/lib/utils"

const TooltipProvider = TooltipPrimitive.Provider

const Tooltip = TooltipPrimitive.Root

const TooltipTrigger = TooltipPrimitive.Trigger

const TooltipContent = React.forwardRef<
  React.ElementRef<typeof TooltipPrimitive.Content>,
  React.ComponentPropsWithoutRef<typeof TooltipPrimitive.Content>
>(({ className, sideOffset = 4, ...props }, ref) => (
  <TooltipPrimitive.Content
    ref={ref}
    sideOffset={sideOffset}
    className={cn(
      "z-50 overflow-hidden rounded-md border bg-popover px-3 py-1.5 text-sm
text-popover-foreground shadow-md animate-in fade-in-0 zoom-in-95
data-[state=closed]:animate-out data-[state=closed]:fade-out-0
data-[state=closed]:zoom-out-95 data-[side=bottom]:slide-in-from-top-2
data-[side=left]:slide-in-from-right-2 data-[side=right]:slide-in-from-left-2
data-[side=top]:slide-in-from-bottom-2",
      className
    )}
    {...props}
  />
))
TooltipContent.displayName = TooltipPrimitive.Content.displayName

export { Tooltip, TooltipTrigger, TooltipContent, TooltipProvider }


==========
Directory: src\components\ui\custom-ui
==========

File: src\components\ui\custom-ui\back-button.tsx
----------
```

```tsx
"use client";

import React, { useEffect } from "react";
import { Button } from "@/components/ui/button";
import { ArrowLeft } from "lucide-react";
import { useRouter, usePathname } from "next/navigation";

export function BackButton() {
  const router = useRouter();
  const pathname = usePathname();

  const getParentPath = (path: string) => {
    const segments = path.split("/").filter(Boolean);
    const parentSegments = segments.slice(0, -1);
    return parentSegments.length ? `/${parentSegments.join("/")}` : "/";
  };

  const checkPageExists = async (path: string): Promise<boolean> => {
    try {
      const response = await fetch(path);
      return response.status !== 404;
    } catch (error) {
      return false;
    }
  };

  const findValidParentPath = async (currentPath: string): Promise<string> => {
    if (currentPath === "/") return "/";

    const exists = await checkPageExists(currentPath);
    if (exists) return currentPath;

    const parentPath = getParentPath(currentPath);
    return findValidParentPath(parentPath);
  };

  const handleBack = async () => {
    const initialParentPath = getParentPath(pathname);
    const validPath = await findValidParentPath(initialParentPath);
```

```
    router.push(validPath);
  };

  return (
    <Button variant="ghost" onClick={handleBack}>
      <ArrowLeft className="h-4 w-4" />
      Back
    </Button>
  );
}
```