Core functionalities Source Code Group-7(Team Chat App)

1.User Registration, sign in, sign out, delete account via clerk : This code wraps up the children content with various providers that handle different functionalities. ClerkProvider manages authentication, ThemeProvider handles UI themes (defaulting to dark mode and not supporting system preferences), SocketProvider facilitates real-time socket communication, and QueryProvider likely deals with data fetching or state management. The layout also ensures accessibility with language settings and supports theme-based background color customization.

```
export default function RootLayout({
 children,
}: {
 children: React.ReactNode
}) {
 return (
   <ClerkProvider>
     <html lang="en" suppressHydrationWarning>
       <body className={cn(
         font.className,
         "bg-white dark:bg-[#313338]"
       )}>
         <ThemeProvider
           attribute="class"
           defaultTheme="dark"
           enableSystem={false}
           storageKey="discord-theme"
         >
           <SocketProvider>
             <ModalProvider />
             <QueryProvider>
               {children}
             </QueryProvider>
           </SocketProvider>
         </ThemeProvider>
       </body>
     </html>
   </ClerkProvider>
 )
}
```

Adding the middle ware to access the routes of the clerk api.

```
export const config = {
 matcher: ['/((?!.+\\.[\\w]+$|_next).*)', '/', '/(api|trpc)(.*)'],
};
```

Then using the sign in, signup and sign in pages provided by the clerk in the app specific routes.

```
import { SignIn } from "@clerk/nextjs";
export default function Page() {
 return <SignIn />;
}
```

```
import { SignUp } from "@clerk/nextjs";
export default function Page() {
 return <SignUp />;
}
```

2. UI Theme toggle(dark, light, system default), navigation bar with create server, user account signout and delete account inside manage account. This asynchronous React component NavigationSidebar fetches the current user's profile and the servers they are a member of from the database. If no profile is found, it redirects to the home page. It then renders a navigation sidebar with UI elements for navigating between servers, a separator, and a scrollable area for server items. At the bottom, it includes a theme toggle and a user button for sign-out functionality, styled specifically for the sidebar context.

```
export const NavigationSidebar = async () => {
 const profile = await currentProfile();

 if (!profile) {
   return redirect("/");
 }

 const servers = await db.server.findMany({
   where: {
     members: {
       some: {
         profileId: profile.id
       }
     }
   }
 });

 return (
   <div
     className="space-y-4 flex flex-col items-center h-full text-primary w-full dark:bg-[#1E1F22]
bg-[#E3E5E8] py-3"
   >
     <NavigationAction />
     <Separator
       className="h-[2px] bg-zinc-300 dark:bg-zinc-700 rounded-md w-10 mx-auto"
     />
     <ScrollArea className="flex-1 w-full">
       {servers.map((server) => (
         <div key={server.id} className="mb-4">
           <NavigationItem
             id={server.id}
             name={server.name}
             imageUrl={server.imageUrl}
           />
         </div>
       ))}
     </ScrollArea>
     <div className="pb-3 mt-auto flex items-center flex-col gap-y-4">
       <ModeToggle />
       <UserButton
         afterSignOutUrl="/"
         appearance={{
           elements: {
             avatarBox: "h-[48px] w-[48px]"
           }
         }}
       />
     </div>
   </div>
 )
}
```

3. Mode toggle for theme UI : The `ModeToggle` function is a React component that renders a button triggering a dropdown menu, allowing users to switch between light, dark, or system UI themes, utilizing the `useTheme` hook for theme management.

```
export function ModeToggle() {
 const { setTheme } = useTheme()
```

```
 return (
   <DropdownMenu>
     <DropdownMenuTrigger asChild>
       <Button className="bg-transparent border-0" variant="outline" size="icon">
         <Sun className="h-[1.2rem] w-[1.2rem] rotate-0 scale-100 transition-all dark:-rotate-90
dark:scale-0" />
         <Moon className="absolute h-[1.2rem] w-[1.2rem] rotate-90 scale-0 transition-all
dark:rotate-0 dark:scale-100" />
         <span className="sr-only">Toggle theme</span>
       </Button>
     </DropdownMenuTrigger>
     <DropdownMenuContent align="end">
       <DropdownMenuItem onClick={() => setTheme("light")}>
         Light
       </DropdownMenuItem>
       <DropdownMenuItem onClick={() => setTheme("dark")}>
         Dark
       </DropdownMenuItem>
       <DropdownMenuItem onClick={() => setTheme("system")}>
         System
       </DropdownMenuItem>
     </DropdownMenuContent>
   </DropdownMenu>
 )
}
```

4. create server modal : The `CreateServerModal` component allows users to create a new server by submitting a form with a server name and image, utilizing form management with validation and integrating an Axios POST request for server creation, with UI controls for modal display based on state.

```
export const CreateServerModal = () => {
 const { isOpen, onClose, type } = useModal();
 const router = useRouter();

 const isModalOpen = isOpen && type === "createServer";

 const form = useForm({
   resolver: zodResolver(formSchema),
   defaultValues: {
     name: "",
     imageUrl: "",
   }
 });

 const isLoading = form.formState.isSubmitting;

 const onSubmit = async (values: z.infer<typeof formSchema>) => {
   try {
     await axios.post("/api/servers", values);

     form.reset();
     router.refresh();
     onClose();
   } catch (error) {
     console.log(error);
   }
 }

 const handleClose = () => {
   form.reset();
   onClose();
 }

 return (
   <Dialog open={isModalOpen} onOpenChange={handleClose}>
```

```
    <DialogContent className="bg-white text-black p-0 overflow-hidden">
      <DialogHeader className="pt-8 px-6">
        <DialogTitle className="text-2xl text-center font-bold">
          Customize your server
        </DialogTitle>
        <DialogDescription className="text-center text-zinc-500">
          Give your server a personality with a name and an image. You can always change it later.
        </DialogDescription>
      </DialogHeader>
      <Form {...form}>
        <form onSubmit={form.handleSubmit(onSubmit)} className="space-y-8">
          <div className="space-y-8 px-6">
            <div className="flex items-center justify-center text-center">
              <FormField
                control={form.control}
                name="imageUrl"
                render={({ field }) => (
                  <FormItem>
                    <FormControl>
                      <FileUpload
                        endpoint="serverImage"
                        value={field.value}
                        onChange={field.onChange}
                      />
                    </FormControl>
                  </FormItem>
                )}
              />
            </div>

            <FormField
              control={form.control}
              name="name"
              render={({ field }) => (
                <FormItem>
                  <FormLabel
                    className="uppercase text-xs font-bold text-zinc-500 dark:text-secondary/70"
                  >
                    Server name
                  </FormLabel>
                  <FormControl>
                    <Input
                      disabled={isLoading}
                      className="bg-zinc-300/50 border-0 focus-visible:ring-0 text-black
focus-visible:ring-offset-0"
                      placeholder="Enter server name"
                      {...field}
                    />
                  </FormControl>
                  <FormMessage />
                </FormItem>
              )}
            />
          </div>
          <DialogFooter className="bg-gray-100 px-6 py-4">
            <Button variant="primary" disabled={isLoading}>
              Create
            </Button>
          </DialogFooter>
        </form>
      </Form>
    </DialogContent>
  </Dialog>
 )
}
```

5.Invite code functionality : The `InviteModal` component enables users to invite friends to a server by copying or generating a new invite link, incorporating state management for loading and copied states, and using Axios to update the invite code, all within a modal UI.

```
export const InviteModal = () => {
 const { onOpen, isOpen, onClose, type, data } = useModal();
 const origin = useOrigin();

 const isModalOpen = isOpen && type === "invite";
 const { server } = data;

 const [copied, setCopied] = useState(false);
 const [isLoading, setIsLoading] = useState(false);

 const inviteUrl = `${origin}/invite/${server?.inviteCode}`;

 const onCopy = () => {
   navigator.clipboard.writeText(inviteUrl);
   setCopied(true);

   setTimeout(() => {
     setCopied(false);
   }, 1000);
 };

 const onNew = async () => {
   try {
     setIsLoading(true);
     const response = await axios.patch(`/api/servers/${server?.id}/invite-code`);

     onOpen("invite", { server: response.data });
   } catch (error) {
     console.log(error);
   } finally {
     setIsLoading(false);
   }
 }

 return (
   <Dialog open={isModalOpen} onOpenChange={onClose}>
     <DialogContent className="bg-white text-black p-0 overflow-hidden">
       <DialogHeader className="pt-8 px-6">
         <DialogTitle className="text-2xl text-center font-bold">
           Invite Friends
         </DialogTitle>
       </DialogHeader>
       <div className="p-6">
         <Label
           className="uppercase text-xs font-bold text-zinc-500 dark:text-secondary/70"
         >
           Server invite link
         </Label>
         <div className="flex items-center mt-2 gap-x-2">
           <Input
             disabled={isLoading}
             className="bg-zinc-300/50 border-0 focus-visible:ring-0 text-black
focus-visible:ring-offset-0"
             value={inviteUrl}
           />
           <Button disabled={isLoading} onClick={onCopy} size="icon">
             {copied
               ? <Check className="w-4 h-4" />
               : <Copy className="w-4 h-4" />
             }
           </Button>
         </div>
         <Button
           onClick={onNew}
```

```
            disabled={isLoading}
            variant="link"
            size="sm"
            className="text-xs text-zinc-500 mt-4"
          >
            Generate a new link
            <RefreshCw className="w-4 h-4 ml-2" />
          </Button>
        </div>
      </DialogContent>
    </Dialog>
  )
}
```

6.Chat input component : The `ChatInput` component allows users to send messages or files within a chat environment, integrating a form for message content submission with additional options for uploading files and inserting emojis, enhancing interactive and dynamic communication capabilities in the application.
ChatItem Component : The `ChatItem` component renders individual chat messages, supporting message editing, deletion, file previews (images/PDFs), and navigating to a member's conversation, leveraging state management for editing mode and integrating custom hooks for forms and modals, all within a responsive design.

```
export const ChatInput = ({
  apiUrl,
  query,
  name,
  type,
}: ChatInputProps) => {
  const { onOpen } = useModal();
  const router = useRouter();

  const form = useForm<z.infer<typeof formSchema>>({
    resolver: zodResolver(formSchema),
    defaultValues: {
      content: "",
    }
  });

  const isLoading = form.formState.isSubmitting;

  const onSubmit = async (values: z.infer<typeof formSchema>) => {
    try {
      const url = qs.stringifyUrl({
        url: apiUrl,
        query,
      });

      await axios.post(url, values);

      form.reset();
      router.refresh();
    } catch (error) {
      console.log(error);
    }
  }

  return (
    <Form {...form}>
      <form onSubmit={form.handleSubmit(onSubmit)}>
        <FormField
          control={form.control}
          name="content"
          render={({ field }) => (
            <FormItem>
              <FormControl>
                <div className="relative p-4 pb-6">
                  <button
```

```
                    type="button"
                    onClick={() => onOpen("messageFile", { apiUrl, query })}
                    className="absolute top-7 left-8 h-[24px] w-[24px] bg-zinc-500 dark:bg-zinc-400
hover:bg-zinc-600 dark:hover:bg-zinc-300 transition rounded-full p-1 flex items-center
justify-center"
                  >
                    <Plus className="text-white dark:text-[#313338]" />
                  </button>
                  <Input
                    disabled={isLoading}
                    className="px-14 py-6 bg-zinc-200/90 dark:bg-zinc-700/75 border-none border-0
focus-visible:ring-0 focus-visible:ring-offset-0 text-zinc-600 dark:text-zinc-200"
                    placeholder={`Message ${type === "conversation" ? name : "#" + name}`}
                    {...field}
                  />
                  <div className="absolute top-7 right-8">
                    <EmojiPicker
                      onChange={(emoji: string) => field.onChange(`${field.value} ${emoji}`)}
                    />
                  </div>
                </div>
              </FormControl>
            </FormItem>
          )}
        />
      </form>
    </Form>
  )
}
```

ChatItem Component :

```
export const ChatItem = ({
  id,
  content,
  member,
  timestamp,
  fileUrl,
  deleted,
  currentMember,
  isUpdated,
  socketUrl,
  socketQuery
}: ChatItemProps) => {
  const [isEditing, setIsEditing] = useState(false);
  const { onOpen } = useModal();
  const params = useParams();
  const router = useRouter();

  const onMemberClick = () => {
    if (member.id === currentMember.id) {
      return;
    }
     router.push(`/servers/${params?.serverId}/conversations/${member.id}`);
  }

  useEffect(() => {
    const handleKeyDown = (event: any) => {
      if (event.key === "Escape" || event.keyCode === 27) {
        setIsEditing(false);
      }
    };

    window.addEventListener("keydown", handleKeyDown);

    return () => window.removeEventListener("keyDown", handleKeyDown);
  }, []);
```

```
const form = useForm<z.infer<typeof formSchema>>({
  resolver: zodResolver(formSchema),
  defaultValues: {
    content: content
  }
});

const isLoading = form.formState.isSubmitting;

const onSubmit = async (values: z.infer<typeof formSchema>) => {
  try {
    const url = qs.stringifyUrl({
      url: `${socketUrl}/${id}`,
      query: socketQuery,
    });

    await axios.patch(url, values);

    form.reset();
    setIsEditing(false);
  } catch (error) {
    console.log(error);
  }
}

useEffect(() => {
  form.reset({
    content: content,
  })
}, [content]);

const fileType = fileUrl?.split(".").pop();

const isAdmin = currentMember.role === MemberRole.ADMIN;
const isModerator = currentMember.role === MemberRole.MODERATOR;
const isOwner = currentMember.id === member.id;
const canDeleteMessage = !deleted && (isAdmin || isModerator || isOwner);
const canEditMessage = !deleted && isOwner && !fileUrl;
const isPDF = fileType === "pdf" && fileUrl;
const isImage = !isPDF && fileUrl;

return (
  <div className="relative group flex items-center hover:bg-black/5 p-4 transition w-full">
    <div className="group flex gap-x-2 items-start w-full">
      <div onClick={onMemberClick} className="cursor-pointer hover:drop-shadow-md transition">
        <UserAvatar src={member.profile.imageUrl} />
      </div>
      <div className="flex flex-col w-full">
        <div className="flex items-center gap-x-2">
          <div className="flex items-center">
            <p onClick={onMemberClick} className="font-semibold text-sm hover:underline
cursor-pointer">
              {member.profile.name}
            </p>
            <ActionTooltip label={member.role}>
              {roleIconMap[member.role]}
            </ActionTooltip>
          </div>
          <span className="text-xs text-zinc-500 dark:text-zinc-400">
            {timestamp}
          </span>
        </div>
        {isImage && (
          <a
            href={fileUrl}
            target="_blank"
```

```
            rel="noopener noreferrer"
            className="relative aspect-square rounded-md mt-2 overflow-hidden border flex
items-center bg-secondary h-48 w-48"
          >
            <Image
              src={fileUrl}
              alt={content}
              fill
              className="object-cover"
            />
          </a>
        )}
        {isPDF && (
          <div className="relative flex items-center p-2 mt-2 rounded-md bg-background/10">
            <FileIcon className="h-10 w-10 fill-indigo-200 stroke-indigo-400" />
            <a
              href={fileUrl}
              target="_blank"
              rel="noopener noreferrer"
              className="ml-2 text-sm text-indigo-500 dark:text-indigo-400 hover:underline"
            >
              PDF File
            </a>
          </div>
        )}
        {!fileUrl && !isEditing && (
          <p className={cn(
            "text-sm text-zinc-600 dark:text-zinc-300",
            deleted && "italic text-zinc-500 dark:text-zinc-400 text-xs mt-1"
          )}>
            {content}
            {isUpdated && !deleted && (
              <span className="text-[10px] mx-2 text-zinc-500 dark:text-zinc-400">
                (edited)
              </span>
            )}
          </p>
        )}
        {!fileUrl && isEditing && (
          <Form {...form}>
            <form
              className="flex items-center w-full gap-x-2 pt-2"
              onSubmit={form.handleSubmit(onSubmit)}>
                <FormField
                  control={form.control}
                  name="content"
                  render={({ field }) => (
                    <FormItem className="flex-1">
                      <FormControl>
                        <div className="relative w-full">
                          <Input
                            disabled={isLoading}
                            className="p-2 bg-zinc-200/90 dark:bg-zinc-700/75 border-none border-0
focus-visible:ring-0 focus-visible:ring-offset-0 text-zinc-600 dark:text-zinc-200"
                            placeholder="Edited message"
                            {...field}
                          />
                        </div>
                      </FormControl>
                    </FormItem>
                  )}
                />
                <Button disabled={isLoading} size="sm" variant="primary">
                  Save
                </Button>
            </form>
            <span className="text-[10px] mt-1 text-zinc-400">
```

```
              Press escape to cancel, enter to save
            </span>
          </Form>
        )}
      </div>
    </div>
    {canDeleteMessage && (
      <div className="hidden group-hover:flex items-center gap-x-2 absolute p-1 -top-2 right-5
bg-white dark:bg-zinc-800 border rounded-sm">
        {canEditMessage && (
          <ActionTooltip label="Edit">
            <Edit
              onClick={() => setIsEditing(true)}
              className="cursor-pointer ml-auto w-4 h-4 text-zinc-500 hover:text-zinc-600
dark:hover:text-zinc-300 transition"
            />
          </ActionTooltip>
        )}
        <ActionTooltip label="Delete">
          <Trash
            onClick={() => onOpen("deleteMessage", {
              apiUrl: `${socketUrl}/${id}`,
              query: socketQuery,
             })}
            className="cursor-pointer ml-auto w-4 h-4 text-zinc-500 hover:text-zinc-600
dark:hover:text-zinc-300 transition"
          />
        </ActionTooltip>
      </div>
    )}
  </div>
 )
}
```

7. SocketProvider : The `SocketProvider` component manages the lifecycle of a WebSocket connection using Socket.io, encapsulating the connection logic to maintain a real-time communication channel. It initializes the connection on component mount, updates connection status on connect and disconnect events, and ensures clean disconnection on unmount, providing a context to access the socket instance and connection status throughout the app.

```
export const SocketProvider = ({
 children
}: {
 children: React.ReactNode
}) => {
 const [socket, setSocket] = useState(null);
 const [isConnected, setIsConnected] = useState(false);

 useEffect(() => {
   const socketInstance = new (ClientIO as any)(process.env.NEXT_PUBLIC_SITE_URL!, {
     path: "/api/socket/io",
     addTrailingSlash: false,
   });

   socketInstance.on("connect", () => {
     setIsConnected(true);
   });

   socketInstance.on("disconnect", () => {
     setIsConnected(false);
   });

   setSocket(socketInstance);

   return () => {
     socketInstance.disconnect();
   }
```

```
 }, []);

 return (
   <SocketContext.Provider value={{ socket, isConnected }}>
     {children}
   </SocketContext.Provider>
 )
}
```

8.Direct message communication between memberOne and memberTwo : This API handler function manages the creation of direct messages within a conversation in a Next.js application, verifying request methods, user authentication, and required fields before persisting the message to the database and broadcasting it to relevant clients using Socket.io.

```
export default async function handler(
 req: NextApiRequest,
 res: NextApiResponseServerIo,
) {
 if (req.method !== "POST") {
   return res.status(405).json({ error: "Method not allowed" });
 }

 try {
   const profile = await currentProfilePages(req);
   const { content, fileUrl } = req.body;
   const { conversationId } = req.query;

   if (!profile) {
     return res.status(401).json({ error: "Unauthorized" });
   }
    if (!conversationId) {
     return res.status(400).json({ error: "Conversation ID missing" });
   }

   if (!content) {
     return res.status(400).json({ error: "Content missing" });
   }


   const conversation = await db.conversation.findFirst({
     where: {
       id: conversationId as string,
       OR: [
         {
           memberOne: {
             profileId: profile.id,
           }
         },
         {
           memberTwo: {
             profileId: profile.id,
           }
         }
       ]
     },
     include: {
       memberOne: {
         include: {
           profile: true,
         }
       },
       memberTwo: {
         include: {
           profile: true,
         }
       }
     }
```

```
    })

    if (!conversation) {
      return res.status(404).json({ message: "Conversation not found" });
    }

    const member = conversation.memberOne.profileId === profile.id ? conversation.memberOne :
conversation.memberTwo

    if (!member) {
      return res.status(404).json({ message: "Member not found" });
    }

    const message = await db.directMessage.create({
      data: {
        content,
        fileUrl,
        conversationId: conversationId as string,
        memberId: member.id,
      },
      include: {
        member: {
          include: {
            profile: true,
          }
        }
      }
    });

    const channelKey = `chat:${conversationId}:messages`;

    res?.socket?.server?.io?.emit(channelKey, message);

    return res.status(200).json(message);
  } catch (error) {
    console.log("[DIRECT_MESSAGES_POST]", error);
    return res.status(500).json({ message: "Internal Error" });
  }
}
```

9.Channel Message component : This API handler manages the creation of messages within a specific channel on a server, validating user authentication, server, and channel membership before saving the message to the database and broadcasting it via WebSocket to update the channel in real-time for all members.

```
export default async function handler(
  req: NextApiRequest,
  res: NextApiResponseServerIo,
) {
  if (req.method !== "POST") {
    return res.status(405).json({ error: "Method not allowed" });
  }

  try {
    const profile = await currentProfilePages(req);
    const { content, fileUrl } = req.body;
    const { serverId, channelId } = req.query;

    if (!profile) {
      return res.status(401).json({ error: "Unauthorized" });
    }
     if (!serverId) {
      return res.status(400).json({ error: "Server ID missing" });
    }

    if (!channelId) {
      return res.status(400).json({ error: "Channel ID missing" });
```

```
    }

    if (!content) {
      return res.status(400).json({ error: "Content missing" });
    }

    const server = await db.server.findFirst({
      where: {
        id: serverId as string,
        members: {
          some: {
            profileId: profile.id
          }
        }
      },
      include: {
        members: true,
      }
    });

    if (!server) {
      return res.status(404).json({ message: "Server not found" });
    }

    const channel = await db.channel.findFirst({
      where: {
        id: channelId as string,
        serverId: serverId as string,
      }
    });

    if (!channel) {
      return res.status(404).json({ message: "Channel not found" });
    }

    const member = server.members.find((member) => member.profileId === profile.id);

    if (!member) {
      return res.status(404).json({ message: "Member not found" });
    }

    const message = await db.message.create({
      data: {
        content,
        fileUrl,
        channelId: channelId as string,
        memberId: member.id,
      },
      include: {
        member: {
          include: {
            profile: true,
          }
        }
      }
    });

    const channelKey = `chat:${channelId}:messages`;

    res?.socket?.server?.io?.emit(channelKey, message);

    return res.status(200).json(message);
  } catch (error) {
    console.log("[MESSAGES_POST]", error);
    return res.status(500).json({ message: "Internal Error" });
  }
}
```

10.Emoji Picker : The `EmojiPicker` component integrates an emoji selection tool into the application, automatically adapting its theme to match the current user interface theme and enabling users to enrich their messages with emojis.

```
export const EmojiPicker = ({
 onChange,
}: EmojiPickerProps) => {
 const { resolvedTheme } = useTheme();

 return (
   <Popover>
     <PopoverTrigger>
       <Smile
         className="text-zinc-500 dark:text-zinc-400 hover:text-zinc-600 dark:hover:text-zinc-300
transition"
       />
     </PopoverTrigger>
     <PopoverContent
       side="right"
       sideOffset={40}
       className="bg-transparent border-none shadow-none drop-shadow-none mb-16"
     >
       <Picker
         theme={resolvedTheme}
         data={data}
         onEmojiSelect={(emoji: any) => onChange(emoji.native)}
       />
     </PopoverContent>
   </Popover>
 )
}
```

11.File Upload : The `FileUpload` component utilizes UploadThing to manage file uploads, allowing users to upload, display, and remove files directly within the application. For image files, it showcases a thumbnail preview with a convenient option to delete the file. PDF uploads are represented by an icon and file name, including a delete button for easy removal. When no file is selected, the component presents a user-friendly dropzone, powered by UploadThing, enabling seamless file sharing and management within the chat environment.

```
export const FileUpload = ({
 onChange,
 value,
 endpoint
}: FileUploadProps) => {
 const fileType = value?.split(".").pop();


 if (value && fileType !== "pdf") {
   return (
     <div className="relative h-20 w-20">
       <Image
         fill
         src={value}
         alt="Upload"
         className="rounded-full"
       />
       <button
         onClick={() => onChange("")}
```

```
        className="bg-rose-500 text-white p-1 rounded-full absolute top-0 right-0 shadow-sm"
        type="button"
      >
        <X className="h-4 w-4" />
      </button>
    </div>
  )
}

if (value && fileType === "pdf") {
  return (
    <div className="relative flex items-center p-2 mt-2 rounded-md bg-background/10">
      <FileIcon className="h-10 w-10 fill-indigo-200 stroke-indigo-400" />
      <a
        href={value}
        target="_blank"
        rel="noopener noreferrer"
        className="ml-2 text-sm text-indigo-500 dark:text-indigo-400 hover:underline"
      >
        {value}
      </a>
      <button
        onClick={() => onChange("")}
        className="bg-rose-500 text-white p-1 rounded-full absolute -top-2 -right-2 shadow-sm"
        type="button"
      >
        <X className="h-4 w-4" />
      </button>
    </div>
  )
}

return (
  <UploadDropzone
    endpoint={endpoint}
    onClientUploadComplete={(res) => {
      onChange(res?.[0].url);
    }}
    onUploadError={(error: Error) => {
      console.log(error);
    }}
  />
)
}
```