"use client";

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

import {

  Card,

  CardContent,

  CardDescription,

  CardFooter,

  CardHeader,

  CardTitle,

} from "@/components/ui/card";

import { Label } from "@/components/ui/label";

import { Input } from "@/components/ui/input";

import { Button } from "@/components/ui/button";

import { Check, CopyIcon } from "lucide-react";

import { useSocket } from "./SP";

import toast from "react-hot-toast";

import { TailSpin } from "react-loader-spinner";

import Peer from "simple-peer";

import FileUpload from "./FU";

import FileUploadBtn from "./FUButton";

import FileDownload from "./FD";

import ShareLink from "./ShareLink";

import { useSearchParams } from "next/navigation";

import { Dots\_v3 } from "@/components/ui/dots";

import { EyeCatchingButton\_v1 } from "@/components/ui/shimmerButton";

const ShareCard = () => {

  const userDetails = useSocket();

  const [partnerId, setpartnerId] = useState("");

  const [isLoading, setisLoading] = useState(false);

  const [isCopied, setisCopied] = useState(false);

  const [currentConnection, setcurrentConnection] = useState(false);

  const peerRef = useRef<any>();

  const [userId, setuserId] = useState<any>();

  const [signalingData, setsignalingData] = useState<any>();

  const [acceptCaller, setacceptCaller] = useState(false);

  const [terminateCall, setterminateCall] = useState(false);

  const [fileUpload, setfileUpload] = useState<any>();

  const fileInputRef = useRef<any>();

  const [downloadFile, setdownloadFile] = useState<any>();

  const [fileUploadProgress, setfileUploadProgress] = useState<number>(0);

  const [fileDownloadProgress, setfileDownloadProgress] = useState<number>(0);

  const [fileNameState, setfileNameState] = useState<any>();

  const [fileSending, setfileSending] = useState(false);

  const [fileReceiving, setfileReceiving] = useState(false);

  const [setname] = useState<any>();

  const searchParams = useSearchParams();

  const [showContent, setShowContent] = useState(false);

  useEffect(() => {

    const timer = setTimeout(() => {

      setShowContent(true);

    }, 1000);

    return () => clearTimeout(timer);

  }, []);

  const workerRef = useRef<Worker>();

  const addUserToSocketDB = () => {

    userDetails.socket.on("connect", () => {

      setuserId(userDetails.userId);

      userDetails.socket.emit("details", {

        socketId: userDetails.socket.id,

        uniqueId: userDetails.userId,

      });

    });

  };

  function CopyToClipboard(value: any) {

    setisCopied(true);

    toast.success("Copied");

    navigator.clipboard.writeText(value);

    setTimeout(() => {

      setisCopied(false);

    }, 3000);

  }

  useEffect(() => {

    workerRef.current = new Worker(new URL("./w.ts", import.meta.url));

    addUserToSocketDB();

    if (searchParams.get("code")) {

      setpartnerId(String(searchParams.get("code")));

    }

    userDetails.socket.on("signaling", (data: any) => {

      setacceptCaller(true);

      setsignalingData(data);

      setpartnerId(data.from);

    });

    workerRef.current?.addEventListener("message", (event: any) => {

      if (event.data?.progress) {

        setfileDownloadProgress(Number(event.data.progress));

      } else if (event.data?.blob) {

        setdownloadFile(event.data?.blob);

        setfileDownloadProgress(0);

        setfileReceiving(false);

      }

    });

    console.log(userDetails.socket);

    return () => {

      peerRef.current?.destroy();

      if (peerRef.current) {

        setacceptCaller(false);

        setacceptCaller(false);

        userDetails.socket.off();

      }

      workerRef.current?.terminate();

    };

  }, []);

  const callUser = () => {

    const peer = new Peer({

      initiator: true,

      trickle: false,

      config: {

        iceServers: [

          {

            urls: "turn:openrelay.metered.ca:80",

            username: "openrelayproject",

            credential: "openrelayproject",

          },

          {

            urls: "turn:numb.viagenie.ca",

            credential: "muazkh",

            username: "webrtc@live.com",

          },

        ],

      },

    });

    peerRef.current = peer;

    peer.on("signal", (data) => {

      userDetails.socket.emit("send-signal", {

        from: userDetails.userId,

        signalData: data,

        to: partnerId,

      });

    });

    peer.on("data", (data) => {

      const parsedData = JSON.parse(data);

      if (parsedData.chunk) {

        setfileReceiving(true);

        handleReceivingData(parsedData.chunk);

      } else if (parsedData.done) {

        handleReceivingData(parsedData);

        toast.success("File received successfully");

      } else if (parsedData.info) {

        handleReceivingData(parsedData);

      }

    });

    userDetails.socket.on("callAccepted", (data: any) => {

      peer.signal(data.signalData);

      setisLoading(false);

      setcurrentConnection(true);

      setterminateCall(true);

      toast.success(`Successful connection with ${partnerId}`);

      userDetails.setpeerState(peer);

    });

    peer.on("close", () => {

      setpartnerId("");

      setcurrentConnection(false);

      toast.error(`${partnerId} disconnected`);

      setfileUpload(false);

      setterminateCall(false);

      setpartnerId("");

      userDetails.setpeerState(undefined);

    });

    peer.on("error", (err) => {

      console.log(err);

    });

  };

  const acceptUser = () => {

    const peer = new Peer({

      initiator: false,

      trickle: false,

    });

    peerRef.current = peer;

    userDetails.setpeerState(peer);

    peer.on("signal", (data) => {

      userDetails.socket.emit("accept-signal", {

        signalData: data,

        to: partnerId,

      });

      setcurrentConnection(true);

      setacceptCaller(false);

      setterminateCall(true);

      toast.success(`Successful connection with ${partnerId}`);

    });

    peer.on("data", (data) => {

      const parsedData = JSON.parse(data);

      if (parsedData.chunk) {

        setfileReceiving(true);

        handleReceivingData(parsedData.chunk);

      } else if (parsedData.done) {

        handleReceivingData(parsedData);

        toast.success("File received successfully");

      } else if (parsedData.info) {

        handleReceivingData(parsedData);

      }

    });

    peer.signal(signalingData.signalData);

    peer.on("close", () => {

      setpartnerId("");

      setcurrentConnection(false);

      toast.error(`${partnerId} disconnected`);

      setfileUpload(false);

      setterminateCall(false);

      setpartnerId("");

      userDetails.setpeerState(undefined);

    });

    peer.on("error", (err) => {

      console.log(err);

    });

  };

  const handleConnectionMaking = () => {

    setisLoading(true);

    if (partnerId && partnerId.length == 10) {

      callUser();

    } else {

      setisLoading(false);

      toast.error("Invalid token entered.");

    }

  };

  const handleFileUploadBtn = () => {

    fileInputRef.current.click();

  };

  const handleFileChange = (e: any) => {

    setfileUpload(e.target.files);

  };

  function handleReceivingData(data: any) {

    if (data.info) {

      workerRef.current?.postMessage({

        status: "fileInfo",

        fileSize: data.fileSize,

      });

      setfileNameState(data.fileName);

      setname(data.fileName);

    } else if (data.done) {

      const parsed = data;

      const fileSize = parsed.fileSize;

      workerRef.current?.postMessage("download");

    } else {

      setdownloadFile("sjdf");

      workerRef.current?.postMessage(data);

    }

  }

  const handleWebRTCUpload = () => {

    const peer = peerRef.current;

    const file = fileUpload[0];

    const chunkSize = 16 \* 1024;

    let offset = 0;

    const readAndSendChunk = () => {

      const chunk = file.slice(offset, offset + chunkSize);

      const reader = new FileReader();

      if (offset == 0) {

        setfileSending(true);

        const fileInfo = {

          info: true,

          fileName: file.name,

          fileSize: file.size,

          fileType: file.type,

        };

        peer.write(JSON.stringify(fileInfo));

      }

      reader.onload = (event) => {

        if (event.target?.result) {

          const chunkData: any = event.target.result;

          const uint8ArrayChunk = new Uint8Array(chunkData);

          const progressPayload = {

            chunk: Array.from(uint8ArrayChunk),

            progress: (offset / file.size) \* 100,

          };

          peer.write(JSON.stringify(progressPayload));

          setfileUploadProgress((offset / file.size) \* 100);

          offset += chunkSize;

          if (offset < file.size) {

            readAndSendChunk();

          } else {

            peer.write(

              JSON.stringify({

                done: true,

                fileName: file.name,

                fileSize: file.size,

                fileType: file.type,

              })

            );

            setfileUploadProgress(100);

            setfileSending(false);

            toast.success("Sended file successfully");

          }

        }

      };

      reader.readAsArrayBuffer(chunk);

    };

    readAndSendChunk();

  };

  return (

    <>

      <Card className="sm:max-w-[450px] max-w-[95%] z-10">

        <CardHeader>

          <CardTitle>z1ppie</CardTitle>

          <CardDescription>

            Connect to the same network for P2P to work.

          </CardDescription>

        </CardHeader>

        {}

        <CardContent className="mt-1">

          <form>

            <div className="grid w-full items-center gap-4">

              <div className="flex flex-col gap-y-1">

                <Label htmlFor="name">My Token</Label>

                <div className="flex flex-row justify-left items-center space-x-2">

                  <div className="flex border rounded-md px-3 py-2 text-sm h-10 w-full bg-muted">

                    {showContent ? userId ? userId : <Dots\_v3 /> : <Dots\_v3 />}

                  </div>

                  <Button

                    type="button"

                    className="p-4"

                    onClick={() => CopyToClipboard(userDetails?.userId)}

                    disabled={userId ? false : true}

                  >

                    {isCopied ? (

                      <Check size={15} color="green" />

                    ) : (

                      <CopyIcon size={15} />

                    )}

                  </Button>

                  <ShareLink userCode={userId} />

                </div>

              </div>

              <div className="flex flex-col gap-y-1">

                <Label htmlFor="name">Peer's Token</Label>

                <div className="flex flex-row justify-left items-center space-x-2">

                  <Input

                    id="name"

                    placeholder="Input Peer's Token"

                    onChange={(e) => setpartnerId(e.target.value)}

                    disabled={terminateCall}

                    value={partnerId}

                  />

                  <Button

                    type="button"

                    variant="outline"

                    className="flex items-center justify-center p-4 w-[160px]"

                    onClick={handleConnectionMaking}

                    disabled={terminateCall}

                  >

                    {isLoading ? (

                      <>

                        <div className="scale-0 hidden dark:flex dark:scale-100">

                          <TailSpin color="white" height={18} width={18} />

                        </div>

                        <div className="scale-100 flex dark:scale-0 dark:hidden">

                          <TailSpin color="black" height={18} width={18} />

                        </div>

                      </>

                    ) : (

                      <p>Connect</p>

                    )}

                  </Button>

                </div>

              </div>

              <div className="flex flex-col gap-y-1">

                <Label htmlFor="name">Connection Status</Label>

                <div className="flex flex-row justify-left items-center space-x-2">

                  <div className=" border rounded-lg  px-3 py-2 text-sm h-10 w-full ease-in-out duration-500 transition-all select-none">

                    {currentConnection

                      ? `Connected to ${partnerId}`

                      : "No connection"}

                  </div>

                  <>

                    {terminateCall ? (

                      <Button

                        variant="destructive"

                        type="button"

                        // className="p-4 w-[160px] text-red-600 border-red-400 hover:bg-red-300 animate-in slide-in-from-right-[30px]"

                        onClick={() => {

                          peerRef.current.destroy();

                        }}

                      >

                        Terminate

                      </Button>

                    ) : null}

                  </>

                </div>

              </div>

              {/\* file upload \*/}

              <div className="flex flex-col border rounded-lg  px-3 py-2 text-sm w-full ease-in-out duration-500 transition-all gap-y-2">

                <div>

                  <Label className=" font-semibold text-[16px]">

                    Upload a file

                  </Label>

                </div>

                <div>

                  <FileUploadBtn

                    inputRef={fileInputRef}

                    uploadBtn={handleFileUploadBtn}

                    handleFileChange={handleFileChange}

                  />

                </div>

                {fileUpload ? (

                  <FileUpload

                    fileName={fileUpload[0]?.name}

                    fileProgress={fileUploadProgress}

                    handleClick={handleWebRTCUpload}

                    showProgress={fileSending}

                  />

                ) : null}

              </div>

              {/\* download file \*/}

              {downloadFile ? (

                <>

                  <FileDownload

                    fileName={fileNameState}

                    fileReceivingStatus={fileReceiving}

                    fileProgress={fileDownloadProgress}

                    fileRawData={downloadFile}

                  />

                </>

              ) : null}

            </div>

          </form>

        </CardContent>

        {acceptCaller ? (

          <CardFooter className="flex justify-center">

            <div>

              <EyeCatchingButton\_v1 onClick={acceptUser}>

                Click here to connect to {signalingData.from}

              </EyeCatchingButton\_v1>

            </div>

          </CardFooter>

        ) : null}

      </Card>

    </>

  );

};

export default ShareCard;

server/index.js

const dotenv = require("dotenv");

dotenv.config({ silent: process.env.NODE\_ENV === "production" });

const express = require("express");

const http = require("http");

const { Server } = require("socket.io");

const cors = require("cors");

const app = express();

const allowedOrigins = [

  "http://localhost:3000",

  process.env.URL,

  "https://z1ppie.vercel.app" // Add your frontend URL here

];

app.use(

  cors({

    origin: allowedOrigins,

    credentials: true,

  })

);

const httpServer = http.createServer(app);

app.get("/", (req, res) => {

  res.send("hello from server");

});

const io = new Server(httpServer, {

  cors: {

    origin: allowedOrigins,

  },

});

// Maps to track users and rooms

const userRoomMap = new Map(); // Maps socket IDs to room numbers

const userIdMap = new Map(); // Maps socket IDs to unique IDs

const uniqueIdMap = new Map(); // Maps unique IDs to socket IDs

io.on("connection", (socket) => {

  socket.on("joinRoom", (roomNumber) => {

    socket.join(Number(roomNumber));

    userRoomMap.set(socket.id, Number(roomNumber));

    socket.emit("ack", `You have joined room ${roomNumber}`);

  });

  socket.on("message", (messageContent) => {

    const roomNum = userRoomMap.get(socket.id);

    io.to(roomNum).emit("roomMsg", messageContent);

  });

  socket.on("details", (userData) => {

    const userSocketId = userData.socketId;

    const uniqueId = userData.uniqueId;

    userIdMap.set(userSocketId, uniqueId);

    uniqueIdMap.set(uniqueId, userSocketId);

    console.log("New User added");

    for (let [key, value] of userIdMap) {

      console.log(`${key} = ${value}`);

    }

  });

  socket.on("send-signal", (signalData) => {

    console.log(signalData);

    const targetUniqueId = signalData.to;

    const partnerSocketId = uniqueIdMap.get(targetUniqueId);

    io.to(partnerSocketId).emit("signaling", {

      from: signalData.from,

      signalData: signalData.signalData,

      to: signalData.to,

    });

  });

  socket.on("accept-signal", (signalData) => {

    console.log(signalData);

    const targetUniqueId = signalData.to;

    const partnerSocketId = uniqueIdMap.get(targetUniqueId);

    console.log(partnerSocketId);

    io.to(partnerSocketId).emit("callAccepted", {

      signalData: signalData.signalData,

      to: signalData.to,

    });

  });

  socket.on("disconnect", () => {

    console.log(`Socket disconnected: ${socket.id}`);

    const userSocketId = socket.id;

    const associatedUniqueId = userIdMap.get(userSocketId);

    userIdMap.delete(userSocketId);

    uniqueIdMap.delete(associatedUniqueId);

    console.log("Updated userIdMap:");

    for (let [key, value] of userIdMap) {

      console.log(`${key} = ${value}`);

    }

    console.log("Updated uniqueIdMap:");

    for (let [key, value] of uniqueIdMap) {

      console.log(`${key} = ${value}`);

    }

  });

});

httpServer.listen(process.env.PORT || 8000, () => {

  console.log(`Listening on ${process.env.PORT ? process.env.PORT : "8000"}`);

});

New sharecard

"use client";

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

import {

  Card,

  CardContent,

  CardDescription,

  CardFooter,

  CardHeader,

  CardTitle,

} from "@/components/ui/card";

import { Label } from "@/components/ui/label";

import { Input } from "@/components/ui/input";

import { Button } from "@/components/ui/button";

import { Check, CopyIcon } from "lucide-react";

import { useSocket } from "./SP";

import toast from "react-hot-toast";

import { TailSpin } from "react-loader-spinner";

import Peer from "simple-peer";

import FileUpload from "./FU";

import FileUploadBtn from "./FUButton";

import FileDownload from "./FD";

import ShareLink from "./ShareLink";

import { useSearchParams } from "next/navigation";

import { Dots\_v3 } from "@/components/ui/dots";

import { EyeCatchingButton\_v1 } from "@/components/ui/shimmerButton";

const ShareCard = () => {

  const userDetails = useSocket();

  const [partnerId, setpartnerId] = useState("");

  const [isLoading, setisLoading] = useState(false);

  const [isCopied, setisCopied] = useState(false);

  const [currentConnection, setcurrentConnection] = useState(false);

  const peerRef = useRef<any>();

  const [userId, setuserId] = useState<any>();

  const [signalingData, setsignalingData] = useState<any>();

  const [acceptCaller, setacceptCaller] = useState(false);

  const [terminateCall, setterminateCall] = useState(false);

  const [fileUpload, setfileUpload] = useState<any>();

  const fileInputRef = useRef<any>();

  const [downloadFile, setdownloadFile] = useState<any>();

  const [fileUploadProgress, setfileUploadProgress] = useState<number>(0);

  const [fileDownloadProgress, setfileDownloadProgress] = useState<number>(0);

  const [fileNameState, setfileNameState] = useState<any>();

  const [fileSending, setfileSending] = useState(false);

  const [fileReceiving, setfileReceiving] = useState(false);

  const [setname] = useState<any>();

  const searchParams = useSearchParams();

  const [showContent, setShowContent] = useState(false);

  useEffect(() => {

    const timer = setTimeout(() => {

      setShowContent(true);

    }, 1000);

    return () => clearTimeout(timer);

  }, []);

  const workerRef = useRef<Worker>();

  const addUserToSocketDB = () => {

    userDetails.socket.on("connect", () => {

      setuserId(userDetails.userId);

      userDetails.socket.emit("details", {

        socketId: userDetails.socket.id,

        uniqueId: userDetails.userId,

      });

    });

  };

  function CopyToClipboard(value: any) {

    setisCopied(true);

    toast.success("Copied");

    navigator.clipboard.writeText(value);

    setTimeout(() => {

      setisCopied(false);

    }, 3000);

  }

  useEffect(() => {

    workerRef.current = new Worker(new URL("./w.ts", import.meta.url));

    addUserToSocketDB();

    if (searchParams.get("code")) {

      setpartnerId(String(searchParams.get("code")));

    }

    userDetails.socket.on("signaling", (data: any) => {

      setacceptCaller(true);

      setsignalingData(data);

      setpartnerId(data.from);

    });

    workerRef.current?.addEventListener("message", (event: any) => {

      if (event.data?.progress) {

        setfileDownloadProgress(Number(event.data.progress));

      } else if (event.data?.blob) {

        setdownloadFile(event.data?.blob);

        setfileDownloadProgress(0);

        setfileReceiving(false);

      }

    });

    console.log(userDetails.socket);

    return () => {

      peerRef.current?.destroy();

      if (peerRef.current) {

        setacceptCaller(false);

        setacceptCaller(false);

        userDetails.socket.off();

      }

      workerRef.current?.terminate();

    };

  }, []);

  const callUser = () => {

    const peer = new Peer({

      initiator: true,

      trickle: false,

      config: {

        iceServers: [

          {

            urls: "turn:openrelay.metered.ca:80",

            username: "openrelayproject",

            credential: "openrelayproject",

          },

          {

            urls: "turn:numb.viagenie.ca",

            credential: "muazkh",

            username: "webrtc@live.com",

          },

        ],

      },

    });

    peerRef.current = peer;

    peer.on("signal", (data) => {

      userDetails.socket.emit("send-signal", {

        from: userDetails.userId,

        signalData: data,

        to: partnerId,

      });

    });

    peer.on("data", (data) => {

      const parsedData = JSON.parse(data);

      if (parsedData.chunk) {

        setfileReceiving(true);

        handleReceivingData(parsedData.chunk);

      } else if (parsedData.done) {

        handleReceivingData(parsedData);

        toast.success("File received successfully");

      } else if (parsedData.info) {

        handleReceivingData(parsedData);

      }

    });

    userDetails.socket.on("callAccepted", (data: any) => {

      peer.signal(data.signalData);

      setisLoading(false);

      setcurrentConnection(true);

      setterminateCall(true);

      toast.success(`Successful connection with ${partnerId}`);

      userDetails.setpeerState(peer);

    });

    peer.on("close", () => {

      setpartnerId("");

      setcurrentConnection(false);

      toast.error(`${partnerId} disconnected`);

      setfileUpload(false);

      setterminateCall(false);

      setpartnerId("");

      userDetails.setpeerState(undefined);

    });

    peer.on("error", (err) => {

      console.log(err);

    });

  };

  const acceptUser = () => {

    const peer = new Peer({

      initiator: false,

      trickle: false,

      config: {

        iceServers: [

          {

            urls: "turn:openrelay.metered.ca:80",

            username: "openrelayproject",

            credential: "openrelayproject",

          },

          {

            urls: "turn:numb.viagenie.ca",

            credential: "muazkh",

            username: "webrtc@live.com",

          },

        ],

      },

    });

    peerRef.current = peer;

    userDetails.setpeerState(peer);

    peer.on("signal", (data) => {

      userDetails.socket.emit("accept-signal", {

        signalData: data,

        to: partnerId,

      });

      setcurrentConnection(true);

      setacceptCaller(false);

      setterminateCall(true);

      toast.success(`Successful connection with ${partnerId}`);

    });

    peer.on("data", (data) => {

      const parsedData = JSON.parse(data);

      if (parsedData.chunk) {

        setfileReceiving(true);

        handleReceivingData(parsedData.chunk);

      } else if (parsedData.done) {

        handleReceivingData(parsedData);

        toast.success("File received successfully");

      } else if (parsedData.info) {

        handleReceivingData(parsedData);

      }

    });

    peer.signal(signalingData.signalData);

    peer.on("close", () => {

      setpartnerId("");

      setcurrentConnection(false);

      toast.error(`${partnerId} disconnected`);

      setfileUpload(false);

      setterminateCall(false);

      setpartnerId("");

      userDetails.setpeerState(undefined);

    });

    peer.on("error", (err) => {

      console.log(err);

    });

  };

  const handleConnectionMaking = () => {

    setisLoading(true);

    if (partnerId && partnerId.length == 10) {

      callUser();

    } else {

      setisLoading(false);

      toast.error("Invalid token entered.");

    }

  };

  const handleFileUploadBtn = () => {

    fileInputRef.current.click();

  };

  const handleFileChange = (e: any) => {

    setfileUpload(e.target.files);

  };

  function handleReceivingData(data: any) {

    if (data.info) {

      workerRef.current?.postMessage({

        status: "fileInfo",

        fileSize: data.fileSize,

      });

      setfileNameState(data.fileName);

      setname(data.fileName);

    } else if (data.done) {

      const parsed = data;

      const fileSize = parsed.fileSize;

      workerRef.current?.postMessage("download");

    } else {

      setdownloadFile("sjdf");

      workerRef.current?.postMessage(data);

    }

  }

  const handleWebRTCUpload = () => {

    const peer = peerRef.current;

    const file = fileUpload[0];

    const chunkSize = 16 \* 1024;

    let offset = 0;

    const readAndSendChunk = () => {

      const chunk = file.slice(offset, offset + chunkSize);

      const reader = new FileReader();

      if (offset == 0) {

        setfileSending(true);

        const fileInfo = {

          info: true,

          fileName: file.name,

          fileSize: file.size,

          fileType: file.type,

        };

        peer.write(JSON.stringify(fileInfo));

      }

      reader.onload = (event) => {

        if (event.target?.result) {

          const chunkData: any = event.target.result;

          const uint8ArrayChunk = new Uint8Array(chunkData);

          const progressPayload = {

            chunk: Array.from(uint8ArrayChunk),

            progress: (offset / file.size) \* 100,

          };

          peer.write(JSON.stringify(progressPayload));

          setfileUploadProgress((offset / file.size) \* 100);

          offset += chunkSize;

          if (offset < file.size) {

            readAndSendChunk();

          } else {

            peer.write(

              JSON.stringify({

                done: true,

                fileName: file.name,

                fileSize: file.size,

                fileType: file.type,

              })

            );

            setfileUploadProgress(100);

            setfileSending(false);

            toast.success("Sended file successfully");

          }

        }

      };

      reader.readAsArrayBuffer(chunk);

    };

    readAndSendChunk();

  };

  return (

    <>

      <Card className="sm:max-w-[450px] max-w-[95%] z-10">

        <CardHeader>

          <CardTitle>z1ppie</CardTitle>

          <CardDescription>

            Connect to the same network for P2P to work.

          </CardDescription>

        </CardHeader>

        {}

        <CardContent className="mt-1">

          <form>

            <div className="grid w-full items-center gap-4">

              <div className="flex flex-col gap-y-1">

                <Label htmlFor="name">My Token</Label>

                <div className="flex flex-row justify-left items-center space-x-2">

                  <div className="flex border rounded-md px-3 py-2 text-sm h-10 w-full bg-muted">

                    {showContent ? userId ? userId : <Dots\_v3 /> : <Dots\_v3 />}

                  </div>

                  <Button

                    type="button"

                    className="p-4"

                    onClick={() => CopyToClipboard(userDetails?.userId)}

                    disabled={userId ? false : true}

                  >

                    {isCopied ? (

                      <Check size={15} color="green" />

                    ) : (

                      <CopyIcon size={15} />

                    )}

                  </Button>

                  <ShareLink userCode={userId} />

                </div>

              </div>

              <div className="flex flex-col gap-y-1">

                <Label htmlFor="name">Peer's Token</Label>

                <div className="flex flex-row justify-left items-center space-x-2">

                  <Input

                    id="name"

                    placeholder="Input Peer's Token"

                    onChange={(e) => setpartnerId(e.target.value)}

                    disabled={terminateCall}

                    value={partnerId}

                  />

                  <Button

                    type="button"

                    variant="outline"

                    className="flex items-center justify-center p-4 w-[160px]"

                    onClick={handleConnectionMaking}

                    disabled={terminateCall}

                  >

                    {isLoading ? (

                      <>

                        <div className="scale-0 hidden dark:flex dark:scale-100">

                          <TailSpin color="white" height={18} width={18} />

                        </div>

                        <div className="scale-100 flex dark:scale-0 dark:hidden">

                          <TailSpin color="black" height={18} width={18} />

                        </div>

                      </>

                    ) : (

                      <p>Connect</p>

                    )}

                  </Button>

                </div>

              </div>

              <div className="flex flex-col gap-y-1">

                <Label htmlFor="name">Connection Status</Label>

                <div className="flex flex-row justify-left items-center space-x-2">

                  <div className=" border rounded-lg  px-3 py-2 text-sm h-10 w-full ease-in-out duration-500 transition-all select-none">

                    {currentConnection

                      ? `Connected to ${partnerId}`

                      : "No connection"}

                  </div>

                  <>

                    {terminateCall ? (

                      <Button

                        variant="destructive"

                        type="button"

                        // className="p-4 w-[160px] text-red-600 border-red-400 hover:bg-red-300 animate-in slide-in-from-right-[30px]"

                        onClick={() => {

                          peerRef.current.destroy();

                        }}

                      >

                        Terminate

                      </Button>

                    ) : null}

                  </>

                </div>

              </div>

              {/\* file upload \*/}

              <div className="flex flex-col border rounded-lg  px-3 py-2 text-sm w-full ease-in-out duration-500 transition-all gap-y-2">

                <div>

                  <Label className=" font-semibold text-[16px]">

                    Upload a file

                  </Label>

                </div>

                <div>

                  <FileUploadBtn

                    inputRef={fileInputRef}

                    uploadBtn={handleFileUploadBtn}

                    handleFileChange={handleFileChange}

                  />

                </div>

                {fileUpload ? (

                  <FileUpload

                    fileName={fileUpload[0]?.name}

                    fileProgress={fileUploadProgress}

                    handleClick={handleWebRTCUpload}

                    showProgress={fileSending}

                  />

                ) : null}

              </div>

              {/\* download file \*/}

              {downloadFile ? (

                <>

                  <FileDownload

                    fileName={fileNameState}

                    fileReceivingStatus={fileReceiving}

                    fileProgress={fileDownloadProgress}

                    fileRawData={downloadFile}

                  />

                </>

              ) : null}

            </div>

          </form>

        </CardContent>

        {acceptCaller ? (

          <CardFooter className="flex justify-center">

            <div>

              <EyeCatchingButton\_v1 onClick={acceptUser}>

                Click here to connect to {signalingData.from}

              </EyeCatchingButton\_v1>

            </div>

          </CardFooter>

        ) : null}

      </Card>

    </>

  );

};

export default ShareCard;

z1ppie/app/f.ts

export function truncateString(input: string): string {

    if (input.length <= 30) {

      return input;

    } else {

      const truncatedPart = input.substring(0, 27); // Take the first 17 characters

      const lastThreeCharacters = input.substring(input.length - 3); // Take the last 3 characters

      return truncatedPart + "..." + lastThreeCharacters;

    }

  }