Chat.tsx

import React, { useRef, useState, useEffect } from 'react'

import styled from 'styled-components'

import Box from '@mui/material/Box'

import Fab from '@mui/material/Fab'

import Tooltip from '@mui/material/Tooltip'

import IconButton from '@mui/material/IconButton'

import InputBase from '@mui/material/InputBase'

import InsertEmoticonIcon from '@mui/icons-material/InsertEmoticon'

import ChatBubbleOutlineIcon from '@mui/icons-material/ChatBubbleOutline'

import CloseIcon from '@mui/icons-material/Close'

import 'emoji-mart/css/emoji-mart.css'

import { Picker } from 'emoji-mart'

import phaserGame from '../PhaserGame'

import Game from '../scenes/Game'

import { getColorByString } from '../util'

import { useAppDispatch, useAppSelector } from '../hooks'

import { MessageType, setFocused, setShowChat } from '../stores/ChatStore'

const Backdrop = styled.div`

  position: fixed;

  bottom: 60px;

  left: 0;

  height: 400px;

  width: 500px;

  max-height: 50%;

  max-width: 100%;

`

const Wrapper = styled.div`

  position: relative;

  height: 100%;

  padding: 16px;

  display: flex;

  flex-direction: column;

`

const FabWrapper = styled.div`

  margin-top: auto;

`

const ChatHeader = styled.div`

  position: relative;

  height: 35px;

  background: #000000a7;

  border-radius: 10px 10px 0px 0px;

  h3 {

    color: #fff;

    margin: 7px;

    font-size: 17px;

    text-align: center;

  }

  .close {

    position: absolute;

    top: 0;

    right: 0;

  }

`

const ChatBox = styled(Box)`

  height: 100%;

  width: 100%;

  overflow: auto;

  background: #2c2c2c;

  border: 1px solid #00000029;

`

const MessageWrapper = styled.div`

  display: flex;

  flex-wrap: wrap;

  padding: 0px 2px;

  p {

    margin: 3px;

    text-shadow: 0.3px 0.3px black;

    font-size: 15px;

    font-weight: bold;

    line-height: 1.4;

    overflow-wrap: anywhere;

  }

  span {

    color: white;

    font-weight: normal;

  }

  .notification {

    color: grey;

    font-weight: normal;

  }

  :hover {

    background: #3a3a3a;

  }

`

const InputWrapper = styled.form`

  box-shadow: 10px 10px 10px #00000018;

  border: 1px solid #42eacb;

  border-radius: 0px 0px 10px 10px;

  display: flex;

  flex-direction: row;

  background: linear-gradient(180deg, #000000c1, #242424c0);

`

const InputTextField = styled(InputBase)`

  border-radius: 0px 0px 10px 10px;

  input {

    padding: 5px;

  }

`

const EmojiPickerWrapper = styled.div`

  position: absolute;

  bottom: 54px;

  right: 16px;

`

const dateFormatter = new Intl.DateTimeFormat('en', {

  timeStyle: 'short',

  dateStyle: 'short',

})

const Message = ({ chatMessage, messageType }) => {

  const [tooltipOpen, setTooltipOpen] = useState(false)

  return (

    <MessageWrapper

      onMouseEnter={() => {

        setTooltipOpen(true)

      }}

      onMouseLeave={() => {

        setTooltipOpen(false)

      }}

    >

      <Tooltip

        open={tooltipOpen}

        title={dateFormatter.format(chatMessage.createdAt)}

        placement="right"

        arrow

      >

        {messageType === MessageType.REGULAR\_MESSAGE ? (

          <p

            style={{

              color: getColorByString(chatMessage.author),

            }}

          >

            {chatMessage.author}: <span>{chatMessage.content}</span>

          </p>

        ) : (

          <p className="notification">

            {chatMessage.author} {chatMessage.content}

          </p>

        )}

      </Tooltip>

    </MessageWrapper>

  )

}

export default function Chat() {

  const [inputValue, setInputValue] = useState('')

  const [showEmojiPicker, setShowEmojiPicker] = useState(false)

  const [readyToSubmit, setReadyToSubmit] = useState(false)

  const messagesEndRef = useRef<HTMLDivElement>(null)

  const inputRef = useRef<HTMLInputElement>(null)

  const chatMessages = useAppSelector((state) => state.chat.chatMessages)

  const focused = useAppSelector((state) => state.chat.focused)

  const showChat = useAppSelector((state) => state.chat.showChat)

  const dispatch = useAppDispatch()

  const game = phaserGame.scene.keys.game as Game

  const handleChange = (event: React.ChangeEvent<HTMLInputElement>) => {

    setInputValue(event.target.value)

  }

  const handleKeyDown = (event: React.KeyboardEvent<HTMLInputElement>) => {

    if (event.key === 'Escape') {

      // move focus back to the game

      inputRef.current?.blur()

      dispatch(setShowChat(false))

    }

  }

  const handleSubmit = (event: React.FormEvent<HTMLFormElement>) => {

    event.preventDefault()

    // this is added because without this, 2 things happen at the same

    // time when Enter is pressed, (1) the inputRef gets focus (from

    // useEffect) and (2) the form gets submitted (right after the input

    // gets focused)

    if (!readyToSubmit) {

      setReadyToSubmit(true)

      return

    }

    // move focus back to the game

    inputRef.current?.blur()

    const val = inputValue.trim()

    setInputValue('')

    if (val) {

      game.network.addChatMessage(val)

      game.myPlayer.updateDialogBubble(val)

    }

  }

  const scrollToBottom = () => {

    messagesEndRef.current?.scrollIntoView({ behavior: 'smooth' })

  }

  useEffect(() => {

    if (focused) {

      inputRef.current?.focus()

    }

  }, [focused])

  useEffect(() => {

    scrollToBottom()

  }, [chatMessages, showChat])

  return (

    <Backdrop>

      <Wrapper>

        {showChat ? (

          <>

            <ChatHeader>

              <h3>Chat</h3>

              <IconButton

                aria-label="close dialog"

                className="close"

                onClick={() => dispatch(setShowChat(false))}

                size="small"

              >

                <CloseIcon />

              </IconButton>

            </ChatHeader>

            <ChatBox>

              {chatMessages.map(({ messageType, chatMessage }, index) => (

                <Message chatMessage={chatMessage} messageType={messageType} key={index} />

              ))}

              <div ref={messagesEndRef} />

              {showEmojiPicker && (

                <EmojiPickerWrapper>

                  <Picker

                    theme="dark"

                    showSkinTones={false}

                    showPreview={false}

                    onSelect={(emoji) => {

                      setInputValue(inputValue + emoji.native)

                      setShowEmojiPicker(!showEmojiPicker)

                      dispatch(setFocused(true))

                    }}

                    exclude={['recent', 'flags']}

                  />

                </EmojiPickerWrapper>

              )}

            </ChatBox>

            <InputWrapper onSubmit={handleSubmit}>

              <InputTextField

                inputRef={inputRef}

                autoFocus={focused}

                fullWidth

                placeholder="Press Enter to chat"

                value={inputValue}

                onKeyDown={handleKeyDown}

                onChange={handleChange}

                onFocus={() => {

                  if (!focused) {

                    dispatch(setFocused(true))

                    setReadyToSubmit(true)

                  }

                }}

                onBlur={() => {

                  dispatch(setFocused(false))

                  setReadyToSubmit(false)

                }}

              />

              <IconButton aria-label="emoji" onClick={() => setShowEmojiPicker(!showEmojiPicker)}>

                <InsertEmoticonIcon />

              </IconButton>

            </InputWrapper>

          </>

        ) : (

          <FabWrapper>

            <Fab

              color="secondary"

              aria-label="showChat"

              onClick={() => {

                dispatch(setShowChat(true))

                dispatch(setFocused(true))

              }}

            >

              <ChatBubbleOutlineIcon />

            </Fab>

          </FabWrapper>

        )}

      </Wrapper>

    </Backdrop>

  )

}

ComputerDialog.tsk

import React from 'react'

import styled from 'styled-components'

import Button from '@mui/material/Button'

import IconButton from '@mui/material/IconButton'

import CloseIcon from '@mui/icons-material/Close'

import { useAppSelector, useAppDispatch } from '../hooks'

import { closeComputerDialog } from '../stores/ComputerStore'

import Video from './Video'

const Backdrop = styled.div`

  position: fixed;

  top: 0;

  left: 0;

  width: 100%;

  height: 100%;

  overflow: hidden;

  padding: 16px 180px 16px 16px;

`

const Wrapper = styled.div`

  width: 100%;

  height: 100%;

  background: #222639;

  border-radius: 16px;

  padding: 16px;

  color: #eee;

  position: relative;

  display: flex;

  flex-direction: column;

  box-shadow: 0px 0px 5px #0000006f;

  .close {

    position: absolute;

    top: 0px;

    right: 0px;

  }

`

const VideoGrid = styled.div`

  flex: 1;

  min-height: 0;

  display: grid;

  grid-gap: 10px;

  grid-template-columns: repeat(auto-fit, minmax(40%, 1fr));

  .video-container {

    position: relative;

    background: black;

    border-radius: 8px;

    overflow: hidden;

    video {

      position: absolute;

      top: 0;

      left: 0;

      width: 100%;

      height: 100%;

      min-width: 0;

      min-height: 0;

      object-fit: contain;

    }

    .player-name {

      position: absolute;

      bottom: 16px;

      left: 16px;

      color: #fff;

      overflow: hidden;

      text-overflow: ellipsis;

      text-shadow: 0 1px 2px rgb(0 0 0 / 60%), 0 0 2px rgb(0 0 0 / 30%);

      white-space: nowrap;

    }

  }

`

function VideoContainer({ playerName, stream }) {

  return (

    <div className="video-container">

      <Video srcObject={stream} autoPlay></Video>

      {playerName && <div className="player-name">{playerName}</div>}

    </div>

  )

}

export default function ComputerDialog() {

  const dispatch = useAppDispatch()

  const playerNameMap = useAppSelector((state) => state.user.playerNameMap)

  const shareScreenManager = useAppSelector((state) => state.computer.shareScreenManager)

  const myStream = useAppSelector((state) => state.computer.myStream)

  const peerStreams = useAppSelector((state) => state.computer.peerStreams)

  return (

    <Backdrop>

      <Wrapper>

        <IconButton

          aria-label="close dialog"

          className="close"

          onClick={() => dispatch(closeComputerDialog())}

        >

          <CloseIcon />

        </IconButton>

        <div className="toolbar">

          <Button

            variant="contained"

            color="secondary"

            onClick={() => {

              if (shareScreenManager?.myStream) {

                shareScreenManager?.stopScreenShare()

              } else {

                shareScreenManager?.startScreenShare()

              }

            }}

          >

            {shareScreenManager?.myStream ? 'Stop sharing' : 'Share Screen'}

          </Button>

        </div>

        <VideoGrid>

          {myStream && <VideoContainer stream={myStream} playerName="You" />}

          {[...peerStreams.entries()].map(([id, { stream }]) => {

            const playerName = playerNameMap.get(id)

            return <VideoContainer key={id} playerName={playerName} stream={stream} />

          })}

        </VideoGrid>

      </Wrapper>

    </Backdrop>

  )

}

CreatRoomForm.tsx

import React, { useState } from 'react'

import styled from 'styled-components'

import Button from '@mui/material/Button'

import IconButton from '@mui/material/IconButton'

import TextField from '@mui/material/TextField'

import InputAdornment from '@mui/material/InputAdornment'

import Visibility from '@mui/icons-material/Visibility'

import VisibilityOff from '@mui/icons-material/VisibilityOff'

import { IRoomData } from '../../../types/Rooms'

import { useAppSelector } from '../hooks'

import phaserGame from '../PhaserGame'

import Bootstrap from '../scenes/Bootstrap'

const CreateRoomFormWrapper = styled.form`

  display: flex;

  flex-direction: column;

  width: 320px;

  gap: 20px;

`

export const CreateRoomForm = () => {

  const [values, setValues] = useState<IRoomData>({

    name: '',

    description: '',

    password: null,

    autoDispose: true,

  })

  const [showPassword, setShowPassword] = useState(false)

  const [nameFieldEmpty, setNameFieldEmpty] = useState(false)

  const [descriptionFieldEmpty, setDescriptionFieldEmpty] = useState(false)

  const lobbyJoined = useAppSelector((state) => state.room.lobbyJoined)

  const handleChange = (prop: keyof IRoomData) => (event: React.ChangeEvent<HTMLInputElement>) => {

    setValues({ ...values, [prop]: event.target.value })

  }

  const handleSubmit = (event: React.FormEvent<HTMLFormElement>) => {

    event.preventDefault()

    const isValidName = values.name !== ''

    const isValidDescription = values.description !== ''

    if (isValidName === nameFieldEmpty) setNameFieldEmpty(!nameFieldEmpty)

    if (isValidDescription === descriptionFieldEmpty)

      setDescriptionFieldEmpty(!descriptionFieldEmpty)

    // create custom room if name and description are not empty

    if (isValidName && isValidDescription && lobbyJoined) {

      const bootstrap = phaserGame.scene.keys.bootstrap as Bootstrap

      bootstrap.network

        .createCustom(values)

        .then(() => bootstrap.launchGame())

        .catch((error) => console.error(error))

    }

  }

  return (

    <CreateRoomFormWrapper onSubmit={handleSubmit}>

      <TextField

        label="Name"

        variant="outlined"

        color="secondary"

        autoFocus

        error={nameFieldEmpty}

        helperText={nameFieldEmpty && 'Name is required'}

        onChange={handleChange('name')}

      />

      <TextField

        label="Description"

        variant="outlined"

        color="secondary"

        error={descriptionFieldEmpty}

        helperText={descriptionFieldEmpty && 'Description is required'}

        multiline

        rows={4}

        onChange={handleChange('description')}

      />

      <TextField

        type={showPassword ? 'text' : 'password'}

        label="Password (optional)"

        onChange={handleChange('password')}

        color="secondary"

        InputProps={{

          endAdornment: (

            <InputAdornment position="end">

              <IconButton

                aria-label="toggle password visibility"

                onClick={() => setShowPassword(!showPassword)}

                edge="end"

              >

                {showPassword ? <VisibilityOff /> : <Visibility />}

              </IconButton>

            </InputAdornment>

          ),

        }}

      />

      <Button variant="contained" color="secondary" type="submit">

        Create

      </Button>

    </CreateRoomFormWrapper>

  )

}

CustomroomTable.tsx

import React, { useState } from 'react'

import styled from 'styled-components'

import Button from '@mui/material/Button'

import Paper from '@mui/material/Paper'

import TableContainer from '@mui/material/TableContainer'

import Table from '@mui/material/Table'

import TableRow from '@mui/material/TableRow'

import TableCell from '@mui/material/TableCell'

import TableHead from '@mui/material/TableHead'

import TableBody from '@mui/material/TableBody'

import Tooltip from '@mui/material/Tooltip'

import TextField from '@mui/material/TextField'

import Dialog from '@mui/material/Dialog'

import DialogActions from '@mui/material/DialogActions'

import DialogContent from '@mui/material/DialogContent'

import Alert from '@mui/material/Alert'

import Avatar from '@mui/material/Avatar'

import PeopleAltIcon from '@mui/icons-material/PeopleAlt'

import LockIcon from '@mui/icons-material/Lock'

import { useAppSelector } from '../hooks'

import { getAvatarString, getColorByString } from '../util'

import phaserGame from '../PhaserGame'

import Bootstrap from '../scenes/Bootstrap'

const MessageText = styled.p`

  margin: 10px;

  font-size: 18px;

  color: #eee;

  text-align: center;

`

const CustomRoomTableContainer = styled(TableContainer)<{

  component: React.ElementType

}>`

  max-height: 500px;

  table {

    min-width: 650px;

  }

`

const TableRowWrapper = styled(TableRow)`

  &:last-child td,

  &:last-child th {

    border: 0;

  }

  .avatar {

    height: 30px;

    width: 30px;

    font-size: 15px;

  }

  .name {

    min-width: 100px;

    overflow-wrap: anywhere;

  }

  .description {

    min-width: 200px;

    overflow-wrap: anywhere;

  }

  .join-wrapper {

    display: flex;

    gap: 3px;

    align-items: center;

  }

  .lock-icon {

    font-size: 18px;

  }

`

const PasswordDialog = styled(Dialog)`

  .dialog-content {

    display: flex;

    flex-direction: column;

    gap: 10px;

  }

  .MuiDialog-paper {

    background: #222639;

  }

`

export const CustomRoomTable = () => {

  const [password, setPassword] = useState('')

  const [selectedRoom, setSelectedRoom] = useState('')

  const [showPasswordDialog, setShowPasswordDialog] = useState(false)

  const [showPasswordError, setShowPasswordError] = useState(false)

  const [passwordFieldEmpty, setPasswordFieldEmpty] = useState(false)

  const lobbyJoined = useAppSelector((state) => state.room.lobbyJoined)

  const availableRooms = useAppSelector((state) => state.room.availableRooms)

  const handleJoinClick = (roomId: string, password: string | null) => {

    if (!lobbyJoined) return

    const bootstrap = phaserGame.scene.keys.bootstrap as Bootstrap

    bootstrap.network

      .joinCustomById(roomId, password)

      .then(() => bootstrap.launchGame())

      .catch((error) => {

        console.error(error)

        if (password) setShowPasswordError(true)

      })

  }

  const handlePasswordSubmit = (event: React.FormEvent<HTMLFormElement>) => {

    event.preventDefault()

    const isValidPassword = password !== ''

    if (isValidPassword === passwordFieldEmpty) setPasswordFieldEmpty(!passwordFieldEmpty)

    if (isValidPassword) handleJoinClick(selectedRoom, password)

  }

  const resetPasswordDialog = () => {

    setShowPasswordDialog(false)

    setPassword('')

    setPasswordFieldEmpty(false)

    setShowPasswordError(false)

  }

  return availableRooms.length === 0 ? (

    <MessageText>There are no custom rooms now, create one or join the public lobby.</MessageText>

  ) : (

    <>

      <CustomRoomTableContainer component={Paper}>

        <Table>

          <TableHead>

            <TableRow>

              <TableCell></TableCell>

              <TableCell>Name</TableCell>

              <TableCell>Description</TableCell>

              <TableCell>ID</TableCell>

              <TableCell align="center">

                <PeopleAltIcon />

              </TableCell>

              <TableCell align="right"></TableCell>

            </TableRow>

          </TableHead>

          <TableBody>

            {availableRooms.map((room) => {

              const { roomId, metadata, clients } = room

              const { name, description, hasPassword } = metadata

              return (

                <TableRowWrapper key={roomId}>

                  <TableCell>

                    <Avatar className="avatar" style={{ background: getColorByString(name) }}>

                      {getAvatarString(name)}

                    </Avatar>

                  </TableCell>

                  <TableCell>

                    <div className="name">{name}</div>

                  </TableCell>

                  <TableCell>

                    <div className="description">{description}</div>

                  </TableCell>

                  <TableCell>{roomId}</TableCell>

                  <TableCell align="center">{clients}</TableCell>

                  <TableCell align="center">

                    <Tooltip title={hasPassword ? 'Password required' : ''}>

                      <Button

                        variant="outlined"

                        color="secondary"

                        onClick={() => {

                          if (hasPassword) {

                            setShowPasswordDialog(true)

                            setSelectedRoom(roomId)

                          } else {

                            handleJoinClick(roomId, null)

                          }

                        }}

                      >

                        <div className="join-wrapper">

                          {hasPassword && <LockIcon className="lock-icon" />}

                          Join

                        </div>

                      </Button>

                    </Tooltip>

                  </TableCell>

                </TableRowWrapper>

              )

            })}

          </TableBody>

        </Table>

      </CustomRoomTableContainer>

      <PasswordDialog open={showPasswordDialog} onClose={resetPasswordDialog}>

        <form onSubmit={handlePasswordSubmit}>

          <DialogContent className="dialog-content">

            <MessageText>This a private room, please enter password:</MessageText>

            <TextField

              autoFocus

              fullWidth

              error={passwordFieldEmpty}

              helperText={passwordFieldEmpty && 'Required'}

              value={password}

              label="Password"

              type="password"

              variant="outlined"

              color="secondary"

              onInput={(e) => {

                setPassword((e.target as HTMLInputElement).value)

              }}

            />

            {showPasswordError && (

              <Alert severity="error" variant="outlined">

                Incorrect Password!

              </Alert>

            )}

          </DialogContent>

          <DialogActions>

            <Button color="secondary" onClick={resetPasswordDialog}>

              Cancel

            </Button>

            <Button color="secondary" type="submit">

              Join

            </Button>

          </DialogActions>

        </form>

      </PasswordDialog>

    </>

  )

}

HelperButtonGroup.tsx

import React, { useState } from 'react'

import styled from 'styled-components'

import Fab from '@mui/material/Fab'

import IconButton from '@mui/material/IconButton'

import Avatar from '@mui/material/Avatar'

import Tooltip from '@mui/material/Tooltip'

import HelpOutlineIcon from '@mui/icons-material/HelpOutline'

import ShareIcon from '@mui/icons-material/Share'

import LightModeIcon from '@mui/icons-material/LightMode'

import DarkModeIcon from '@mui/icons-material/DarkMode'

import CloseIcon from '@mui/icons-material/Close'

import LightbulbIcon from '@mui/icons-material/Lightbulb'

import ArrowRightIcon from '@mui/icons-material/ArrowRight'

import GitHubIcon from '@mui/icons-material/GitHub'

import TwitterIcon from '@mui/icons-material/Twitter'

import VideogameAssetIcon from '@mui/icons-material/VideogameAsset'

import VideogameAssetOffIcon from '@mui/icons-material/VideogameAssetOff'

import { BackgroundMode } from '../../../types/BackgroundMode'

import { setShowJoystick, toggleBackgroundMode } from '../stores/UserStore'

import { useAppSelector, useAppDispatch } from '../hooks'

import { getAvatarString, getColorByString } from '../util'

const Backdrop = styled.div`

  position: fixed;

  display: flex;

  gap: 10px;

  bottom: 16px;

  right: 16px;

  align-items: flex-end;

  .wrapper-group {

    display: flex;

    flex-direction: column;

    gap: 10px;

  }

`

const Wrapper = styled.div`

  position: relative;

  font-size: 16px;

  color: #eee;

  background: #222639;

  box-shadow: 0px 0px 5px #0000006f;

  border-radius: 16px;

  padding: 15px 35px 15px 15px;

  display: flex;

  flex-direction: column;

  align-items: center;

  .close {

    position: absolute;

    top: 15px;

    right: 15px;

  }

  .tip {

    margin-left: 12px;

  }

`

const ButtonGroup = styled.div`

  display: flex;

  gap: 10px;

`

const Title = styled.h3`

  font-size: 24px;

  color: #eee;

  text-align: center;

`

const RoomName = styled.div`

  margin: 10px 20px;

  max-width: 460px;

  max-height: 150px;

  overflow-wrap: anywhere;

  overflow-y: auto;

  display: flex;

  gap: 10px;

  justify-content: center;

  align-items: center;

  h3 {

    font-size: 24px;

    color: #eee;

  }

`

const RoomDescription = styled.div`

  margin: 0 20px;

  max-width: 460px;

  max-height: 150px;

  overflow-wrap: anywhere;

  overflow-y: auto;

  font-size: 16px;

  color: #c2c2c2;

  display: flex;

  justify-content: center;

`

const StyledFab = styled(Fab)<{ target?: string }>`

  &:hover {

    color: #1ea2df;

  }

`

export default function HelperButtonGroup() {

  const [showControlGuide, setShowControlGuide] = useState(false)

  const [showRoomInfo, setShowRoomInfo] = useState(false)

  const showJoystick = useAppSelector((state) => state.user.showJoystick)

  const backgroundMode = useAppSelector((state) => state.user.backgroundMode)

  const roomJoined = useAppSelector((state) => state.room.roomJoined)

  const roomId = useAppSelector((state) => state.room.roomId)

  const roomName = useAppSelector((state) => state.room.roomName)

  const roomDescription = useAppSelector((state) => state.room.roomDescription)

  const dispatch = useAppDispatch()

  return (

    <Backdrop>

      <div className="wrapper-group">

        {roomJoined && (

          <Tooltip title={showJoystick ? 'Disable virtual joystick' : 'Enable virtual joystick'}>

            <StyledFab size="small" onClick={() => dispatch(setShowJoystick(!showJoystick))}>

              {showJoystick ? <VideogameAssetOffIcon /> : <VideogameAssetIcon />}

            </StyledFab>

          </Tooltip>

        )}

        {showRoomInfo && (

          <Wrapper>

            <IconButton className="close" onClick={() => setShowRoomInfo(false)} size="small">

              <CloseIcon />

            </IconButton>

            <RoomName>

              <Avatar style={{ background: getColorByString(roomName) }}>

                {getAvatarString(roomName)}

              </Avatar>

              <h3>{roomName}</h3>

            </RoomName>

            <RoomDescription>

              <ArrowRightIcon /> ID: {roomId}

            </RoomDescription>

            <RoomDescription>

              <ArrowRightIcon /> Description: {roomDescription}

            </RoomDescription>

            <p className="tip">

              <LightbulbIcon />

              Shareable link coming up 😄

            </p>

          </Wrapper>

        )}

        {showControlGuide && (

          <Wrapper>

            <Title>Controls</Title>

            <IconButton className="close" onClick={() => setShowControlGuide(false)} size="small">

              <CloseIcon />

            </IconButton>

            <ul>

              <li>

                <strong>W, A, S, D or arrow keys</strong> to move

              </li>

              <li>

                <strong>E</strong> to sit down (when facing a chair)

              </li>

              <li>

                <strong>R</strong> to use computer to screen share (when facing a computer)

              </li>

              <li>

                <strong>Enter</strong> to open chat

              </li>

              <li>

                <strong>ESC</strong> to close chat

              </li>

            </ul>

            <p className="tip">

              <LightbulbIcon />

              Video connection will start if you are close to someone else

            </p>

          </Wrapper>

        )}

      </div>

      <ButtonGroup>

        {roomJoined && (

          <>

            <Tooltip title="Room Info">

              <StyledFab

                size="small"

                onClick={() => {

                  setShowRoomInfo(!showRoomInfo)

                  setShowControlGuide(false)

                }}

              >

                <ShareIcon />

              </StyledFab>

            </Tooltip>

            <Tooltip title="Control Guide">

              <StyledFab

                size="small"

                onClick={() => {

                  setShowControlGuide(!showControlGuide)

                  setShowRoomInfo(false)

                }}

              >

                <HelpOutlineIcon />

              </StyledFab>

            </Tooltip>

          </>

        )}

        <Tooltip title="Visit Our GitHub">

          <StyledFab

            size="small"

            href="https://github.com/kevinshen56714/SkyOffice"

            target="\_blank"

          >

            <GitHubIcon />

          </StyledFab>

        </Tooltip>

        <Tooltip title="Follow Us on Twitter">

          <StyledFab size="small" href="https://twitter.com/SkyOfficeApp" target="\_blank">

            <TwitterIcon />

          </StyledFab>

        </Tooltip>

        <Tooltip title="Switch Background Theme">

          <StyledFab size="small" onClick={() => dispatch(toggleBackgroundMode())}>

            {backgroundMode === BackgroundMode.DAY ? <DarkModeIcon /> : <LightModeIcon />}

          </StyledFab>

        </Tooltip>

      </ButtonGroup>

    </Backdrop>

  )

}

Joystic.tsx

import { Joystick } from 'react-joystick-component'

export interface JoystickMovement {

  isMoving: boolean

  direction: Direction

}

interface Direction {

  left: boolean

  right: boolean

  up: boolean

  down: boolean

}

interface Props {

  onDirectionChange: (arg: JoystickMovement) => void

}

const angleToDirections = (angle: number): Direction => {

  let outObj: Direction = {

    left: false,

    right: false,

    up: false,

    down: false,

  }

  angle = (angle + 360) % 360

  if (angle > 22.5 && angle <= 67.5) {

    outObj.down = true

    outObj.right = true

  } else if (angle > 67.5 && angle <= 112.5) {

    outObj.down = true

  } else if (angle > 112.5 && angle <= 157.5) {

    outObj.down = true

    outObj.left = true

  } else if (angle > 157.5 && angle <= 202.5) {

    outObj.left = true

  } else if (angle > 202.5 && angle <= 247.5) {

    outObj.left = true

    outObj.up = true

  } else if (angle > 247.5 && angle <= 292.5) {

    outObj.up = true

  } else if (angle > 292.5 && angle <= 337.5) {

    outObj.up = true

    outObj.right = true

  } else {

    outObj.right = true

  }

  return outObj

}

const JoystickItem = (props: Props) => {

  return (

    <Joystick

      size={75}

      baseColor="#4b4b4b70"

      stickColor="#42eacb80"

      stop={() => {

        props.onDirectionChange({

          isMoving: false,

          direction: {

            left: false,

            right: false,

            up: false,

            down: false,

          },

        })

      }}

      move={(event) => {

        const x1 = 0

        const y1 = event.y ?? 0

        const x2 = event.x ?? 0

        const y2 = 0

        var deltaX = x2 - x1 // distance between joystick and center

        var deltaY = y2 - y1 // distance between joystick and center

        var rad = Math.atan2(deltaY, deltaX) // In radians

        var deg = (rad \* 180) / Math.PI // In degrees

        var direction = angleToDirections(deg) // Convert degrees to direction

        props.onDirectionChange({ isMoving: true, direction })

      }}

    />

  )

}

export default JoystickItem

LoginDialog.tsx

import React, { useState } from 'react'

import styled from 'styled-components'

import TextField from '@mui/material/TextField'

import Button from '@mui/material/Button'

import Avatar from '@mui/material/Avatar'

import Alert from '@mui/material/Alert'

import AlertTitle from '@mui/material/AlertTitle'

import ArrowRightIcon from '@mui/icons-material/ArrowRight'

import { Swiper, SwiperSlide } from 'swiper/react'

import { Navigation } from 'swiper'

import 'swiper/css'

import 'swiper/css/navigation'

import Adam from '../images/login/Adam\_login.png'

import Ash from '../images/login/Ash\_login.png'

import Lucy from '../images/login/Lucy\_login.png'

import Nancy from '../images/login/Nancy\_login.png'

import { useAppSelector, useAppDispatch } from '../hooks'

import { setLoggedIn } from '../stores/UserStore'

import { getAvatarString, getColorByString } from '../util'

import phaserGame from '../PhaserGame'

import Game from '../scenes/Game'

const Wrapper = styled.form`

  position: fixed;

  top: 50%;

  left: 50%;

  transform: translate(-50%, -50%);

  background: #222639;

  border-radius: 16px;

  padding: 36px 60px;

  box-shadow: 0px 0px 5px #0000006f;

`

const Title = styled.p`

  margin: 5px;

  font-size: 20px;

  color: #c2c2c2;

  text-align: center;

`

const RoomName = styled.div`

  max-width: 500px;

  max-height: 120px;

  overflow-wrap: anywhere;

  overflow-y: auto;

  display: flex;

  gap: 10px;

  justify-content: center;

  align-items: center;

  h3 {

    font-size: 24px;

    color: #eee;

  }

`

const RoomDescription = styled.div`

  max-width: 500px;

  max-height: 150px;

  overflow-wrap: anywhere;

  overflow-y: auto;

  font-size: 16px;

  color: #c2c2c2;

  display: flex;

  justify-content: center;

`

const SubTitle = styled.h3`

  width: 160px;

  font-size: 16px;

  color: #eee;

  text-align: center;

`

const Content = styled.div`

  display: flex;

  margin: 36px 0;

`

const Left = styled.div`

  margin-right: 48px;

  --swiper-navigation-size: 24px;

  .swiper {

    width: 160px;

    height: 220px;

    border-radius: 8px;

    overflow: hidden;

  }

  .swiper-slide {

    width: 160px;

    height: 220px;

    background: #dbdbe0;

    display: flex;

    justify-content: center;

    align-items: center;

  }

  .swiper-slide img {

    display: block;

    width: 95px;

    height: 136px;

    object-fit: contain;

  }

`

const Right = styled.div`

  width: 300px;

`

const Bottom = styled.div`

  display: flex;

  align-items: center;

  justify-content: center;

`

const Warning = styled.div`

  margin-top: 30px;

  position: relative;

  display: flex;

  flex-direction: column;

  gap: 3px;

`

const avatars = [

  { name: 'adam', img: Adam },

  { name: 'ash', img: Ash },

  { name: 'lucy', img: Lucy },

  { name: 'nancy', img: Nancy },

]

// shuffle the avatars array

for (let i = avatars.length - 1; i > 0; i--) {

  const j = Math.floor(Math.random() \* (i + 1))

  ;[avatars[i], avatars[j]] = [avatars[j], avatars[i]]

}

export default function LoginDialog() {

  const [name, setName] = useState<string>('')

  const [avatarIndex, setAvatarIndex] = useState<number>(0)

  const [nameFieldEmpty, setNameFieldEmpty] = useState<boolean>(false)

  const dispatch = useAppDispatch()

  const videoConnected = useAppSelector((state) => state.user.videoConnected)

  const roomJoined = useAppSelector((state) => state.room.roomJoined)

  const roomName = useAppSelector((state) => state.room.roomName)

  const roomDescription = useAppSelector((state) => state.room.roomDescription)

  const game = phaserGame.scene.keys.game as Game

  const handleSubmit = (event: React.FormEvent<HTMLFormElement>) => {

    event.preventDefault()

    if (name === '') {

      setNameFieldEmpty(true)

    } else if (roomJoined) {

      console.log('Join! Name:', name, 'Avatar:', avatars[avatarIndex].name)

      game.registerKeys()

      game.myPlayer.setPlayerName(name)

      game.myPlayer.setPlayerTexture(avatars[avatarIndex].name)

      game.network.readyToConnect()

      dispatch(setLoggedIn(true))

    }

  }

  return (

    <Wrapper onSubmit={handleSubmit}>

      <Title>Joining</Title>

      <RoomName>

        <Avatar style={{ background: getColorByString(roomName) }}>

          {getAvatarString(roomName)}

        </Avatar>

        <h3>{roomName}</h3>

      </RoomName>

      <RoomDescription>

        <ArrowRightIcon /> {roomDescription}

      </RoomDescription>

      <Content>

        <Left>

          <SubTitle>Select an avatar</SubTitle>

          <Swiper

            modules={[Navigation]}

            navigation

            spaceBetween={0}

            slidesPerView={1}

            onSlideChange={(swiper) => {

              setAvatarIndex(swiper.activeIndex)

            }}

          >

            {avatars.map((avatar) => (

              <SwiperSlide key={avatar.name}>

                <img src={avatar.img} alt={avatar.name} />

              </SwiperSlide>

            ))}

          </Swiper>

        </Left>

        <Right>

          <TextField

            autoFocus

            fullWidth

            label="Name"

            variant="outlined"

            color="secondary"

            error={nameFieldEmpty}

            helperText={nameFieldEmpty && 'Name is required'}

            onInput={(e) => {

              setName((e.target as HTMLInputElement).value)

            }}

          />

          {!videoConnected && (

            <Warning>

              <Alert variant="outlined" severity="warning">

                <AlertTitle>Warning</AlertTitle>

                No webcam/mic connected - <strong>connect one for best experience!</strong>

              </Alert>

              <Button

                variant="outlined"

                color="secondary"

                onClick={() => {

                  game.network.webRTC?.getUserMedia()

                }}

              >

                Connect Webcam

              </Button>

            </Warning>

          )}

          {videoConnected && (

            <Warning>

              <Alert variant="outlined">Webcam connected!</Alert>

            </Warning>

          )}

        </Right>

      </Content>

      <Bottom>

        <Button variant="contained" color="secondary" size="large" type="submit">

          Join

        </Button>

      </Bottom>

    </Wrapper>

  )

}

MobileVirtualJoystick.tsx

import { useEffect, useLayoutEffect, useState } from 'react'

import styled from 'styled-components'

import JoystickItem from './Joystick'

import phaserGame from '../PhaserGame'

import Game from '../scenes/Game'

import { useAppSelector } from '../hooks'

import { JoystickMovement } from './Joystick'

const Backdrop = styled.div`

  position: fixed;

  bottom: 100px;

  right: 32px;

  max-height: 50%;

  max-width: 100%;

`

const Wrapper = styled.div`

  position: relative;

  height: 100%;

  padding: 16px;

  display: flex;

  flex-direction: column;

`

const JoystickWrapper = styled.div`

  margin-top: auto;

  align-self: flex-end;

`

export const minimumScreenWidthSize = 650 //px

const isSmallScreen = (smallScreenSize: number) => {

  const [width, setWidth] = useState(window.innerWidth)

  useEffect(() => {

    const handleResize = () => setWidth(window.innerWidth)

    window.addEventListener('resize', handleResize)

    return () => window.removeEventListener('resize', handleResize)

  }, [])

  return width <= smallScreenSize

}

export default function MobileVirtualJoystick() {

  const showJoystick = useAppSelector((state) => state.user.showJoystick)

  const showChat = useAppSelector((state) => state.chat.showChat)

  const hasSmallScreen = isSmallScreen(minimumScreenWidthSize)

  const game = phaserGame.scene.keys.game as Game

  useEffect(() => {}, [showJoystick, showChat])

  const handleMovement = (movement: JoystickMovement) => {

    game.myPlayer?.handleJoystickMovement(movement)

  }

  return (

    <Backdrop>

      <Wrapper>

        {!(showChat && hasSmallScreen) && showJoystick && (

          <JoystickWrapper>

            <JoystickItem onDirectionChange={handleMovement}></JoystickItem>

          </JoystickWrapper>

        )}

      </Wrapper>

    </Backdrop>

  )

}

RoomSelectionDIalog.tsx

import React, { useState } from 'react'

import logo from '../images/logo.png'

import styled from 'styled-components'

import Button from '@mui/material/Button'

import IconButton from '@mui/material/IconButton'

import Tooltip from '@mui/material/Tooltip'

import LinearProgress from '@mui/material/LinearProgress'

import Alert from '@mui/material/Alert'

import Snackbar from '@mui/material/Snackbar'

import HelpOutlineIcon from '@mui/icons-material/HelpOutline'

import ArrowBackIcon from '@mui/icons-material/ArrowBack'

import { CustomRoomTable } from './CustomRoomTable'

import { CreateRoomForm } from './CreateRoomForm'

import { useAppSelector } from '../hooks'

import phaserGame from '../PhaserGame'

import Bootstrap from '../scenes/Bootstrap'

const Backdrop = styled.div`

  position: absolute;

  top: 50%;

  left: 50%;

  transform: translate(-50%, -50%);

  display: flex;

  flex-direction: column;

  gap: 60px;

  align-items: center;

`

const Wrapper = styled.div`

  background: #222639;

  border-radius: 16px;

  padding: 36px 60px;

  box-shadow: 0px 0px 5px #0000006f;

`

const CustomRoomWrapper = styled.div`

  position: relative;

  display: flex;

  flex-direction: column;

  gap: 20px;

  align-items: center;

  justify-content: center;

  .tip {

    font-size: 18px;

  }

`

const TitleWrapper = styled.div`

  display: grid;

  width: 100%;

  .back-button {

    grid-column: 1;

    grid-row: 1;

    justify-self: start;

    align-self: center;

  }

  h1 {

    grid-column: 1;

    grid-row: 1;

    justify-self: center;

    align-self: center;

  }

`

const Title = styled.h1`

  font-size: 24px;

  color: #eee;

  text-align: center;

`

const Content = styled.div`

  display: flex;

  flex-direction: column;

  gap: 20px;

  margin: 20px 0;

  align-items: center;

  justify-content: center;

  img {

    border-radius: 8px;

    height: 120px;

  }

`

const ProgressBarWrapper = styled.div`

  display: flex;

  flex-direction: column;

  align-items: center;

  h3 {

    color: #33ac96;

  }

`

const ProgressBar = styled(LinearProgress)`

  width: 360px;

`

export default function RoomSelectionDialog() {

  const [showCustomRoom, setShowCustomRoom] = useState(false)

  const [showCreateRoomForm, setShowCreateRoomForm] = useState(false)

  const [showSnackbar, setShowSnackbar] = useState(false)

  const lobbyJoined = useAppSelector((state) => state.room.lobbyJoined)

  const handleConnect = () => {

    if (lobbyJoined) {

      const bootstrap = phaserGame.scene.keys.bootstrap as Bootstrap

      bootstrap.network

        .joinOrCreatePublic()

        .then(() => bootstrap.launchGame())

        .catch((error) => console.error(error))

    } else {

      setShowSnackbar(true)

    }

  }

  return (

    <>

      <Snackbar

        anchorOrigin={{ vertical: 'top', horizontal: 'center' }}

        open={showSnackbar}

        autoHideDuration={3000}

        onClose={() => {

          setShowSnackbar(false)

        }}

      >

        <Alert

          severity="error"

          variant="outlined"

          // overwrites the dark theme on render

          style={{ background: '#fdeded', color: '#7d4747' }}

        >

          Trying to connect to server, please try again!

        </Alert>

      </Snackbar>

      <Backdrop>

        <Wrapper>

          {showCreateRoomForm ? (

            <CustomRoomWrapper>

              <TitleWrapper>

                <IconButton className="back-button" onClick={() => setShowCreateRoomForm(false)}>

                  <ArrowBackIcon />

                </IconButton>

                <Title>Create Custom Room</Title>

              </TitleWrapper>

              <CreateRoomForm />

            </CustomRoomWrapper>

          ) : showCustomRoom ? (

            <CustomRoomWrapper>

              <TitleWrapper>

                <IconButton className="back-button" onClick={() => setShowCustomRoom(false)}>

                  <ArrowBackIcon />

                </IconButton>

                <Title>

                  Custom Rooms

                  <Tooltip

                    title="We update the results in realtime, no refresh needed!"

                    placement="top"

                  >

                    <IconButton>

                      <HelpOutlineIcon className="tip" />

                    </IconButton>

                  </Tooltip>

                </Title>

              </TitleWrapper>

              <CustomRoomTable />

              <Button

                variant="contained"

                color="secondary"

                onClick={() => setShowCreateRoomForm(true)}

              >

                Create new room

              </Button>

            </CustomRoomWrapper>

          ) : (

            <>

              <Title>Welcome to SkyOffice</Title>

              <Content>

                <img src={logo} alt="logo" />

                <Button variant="contained" color="secondary" onClick={handleConnect}>

                  Connect to public lobby

                </Button>

                <Button

                  variant="outlined"

                  color="secondary"

                  onClick={() => (lobbyJoined ? setShowCustomRoom(true) : setShowSnackbar(true))}

                >

                  Create/find custom rooms

                </Button>

              </Content>

            </>

          )}

        </Wrapper>

        {!lobbyJoined && (

          <ProgressBarWrapper>

            <h3> Connecting to server...</h3>

            <ProgressBar color="secondary" />

          </ProgressBarWrapper>

        )}

      </Backdrop>

    </>

  )

}

Video.tsx

import React, { VideoHTMLAttributes, useEffect, useRef } from 'react'

type PropsType = VideoHTMLAttributes<HTMLVideoElement> & {

  srcObject: MediaStream

}

export default function Video({ srcObject, ...props }: PropsType) {

  const refVideo = useRef<HTMLVideoElement>(null)

  useEffect(() => {

    if (!refVideo.current) return

    refVideo.current.srcObject = srcObject

  }, [srcObject])

  return <video ref={refVideo} {...props} />

}

VIdeoConnectionDIalog.tsx

import React, { useState } from 'react'

import styled from 'styled-components'

import Button from '@mui/material/Button'

import Alert from '@mui/material/Alert'

import AlertTitle from '@mui/material/AlertTitle'

import phaserGame from '../PhaserGame'

import Game from '../scenes/Game'

const Backdrop = styled.div`

  position: fixed;

  top: 0;

  left: 0;

`

const Wrapper = styled.div`

  width: 100%;

  height: 100%;

  padding: 16px;

  position: relative;

  display: flex;

  flex-direction: column;

`

export default function VideoConnectionDialog() {

  const [connectionWarning, setConnectionWarning] = useState(true)

  return (

    <Backdrop>

      <Wrapper>

        {connectionWarning && (

          <Alert

            severity="warning"

            onClose={() => {

              setConnectionWarning(!connectionWarning)

            }}

          >

            <AlertTitle>Warning</AlertTitle>

            No webcam connected

            <br /> <strong>connect one for full experience!</strong>

          </Alert>

        )}

        <Button

          variant="contained"

          color="secondary"

          onClick={() => {

            const game = phaserGame.scene.keys.game as Game

            game.network.webRTC?.getUserMedia()

          }}

        >

          Connect Webcam

        </Button>

      </Wrapper>

    </Backdrop>

  )

}

WhiteboardDialog.tsx

import React from 'react'

import styled from 'styled-components'

import IconButton from '@mui/material/IconButton'

import CloseIcon from '@mui/icons-material/Close'

import { useAppSelector, useAppDispatch } from '../hooks'

import { closeWhiteboardDialog } from '../stores/WhiteboardStore'

const Backdrop = styled.div`

  position: fixed;

  top: 0;

  left: 0;

  width: 100vw;

  height: 100vh;

  overflow: hidden;

  padding: 16px 180px 16px 16px;

  width: 100%;

  height: 100%;

`

const Wrapper = styled.div`

  width: 100%;

  height: 100%;

  background: #222639;

  border-radius: 16px;

  padding: 16px;

  color: #eee;

  position: relative;

  display: flex;

  flex-direction: column;

  min-width: max-content;

  .close {

    position: absolute;

    top: 0px;

    right: 0px;

  }

`

const WhiteboardWrapper = styled.div`

  flex: 1;

  border-radius: 25px;

  overflow: hidden;

  margin-right: 25px;

  iframe {

    width: 100%;

    height: 100%;

    background: #fff;

  }

`

export default function WhiteboardDialog() {

  const whiteboardUrl = useAppSelector((state) => state.whiteboard.whiteboardUrl)

  const dispatch = useAppDispatch()

  return (

    <Backdrop>

      <Wrapper>

        <IconButton

          aria-label="close dialog"

          className="close"

          onClick={() => dispatch(closeWhiteboardDialog())}

        >

          <CloseIcon />

        </IconButton>

        {whiteboardUrl && (

          <WhiteboardWrapper>

            <iframe title="white board" src={whiteboardUrl} />

          </WhiteboardWrapper>

        )}

      </Wrapper>

    </Backdrop>

  )

}

Eventcenter.ts

import Phaser from 'phaser'

export const phaserEvents = new Phaser.Events.EventEmitter()

export enum Event {

  PLAYER\_JOINED = 'player-joined',

  PLAYER\_UPDATED = 'player-updated',

  PLAYER\_LEFT = 'player-left',

  PLAYER\_DISCONNECTED = 'player-disconnected',

  MY\_PLAYER\_READY = 'my-player-ready',

  MY\_PLAYER\_NAME\_CHANGE = 'my-player-name-change',

  MY\_PLAYER\_TEXTURE\_CHANGE = 'my-player-texture-change',

  MY\_PLAYER\_VIDEO\_CONNECTED = 'my-player-video-connected',

  ITEM\_USER\_ADDED = 'item-user-added',

  ITEM\_USER\_REMOVED = 'item-user-removed',

  UPDATE\_DIALOG\_BUBBLE = 'update-dialog-bubble',

}

Scenes/Background.ts

import Phaser from 'phaser'

import { BackgroundMode } from '../../../types/BackgroundMode'

export default class Background extends Phaser.Scene {

  private cloud!: Phaser.Physics.Arcade.Group

  private cloudKey!: string

  private backdropKey!: string

  constructor() {

    super('background')

  }

  create(data: { backgroundMode: BackgroundMode }) {

    const sceneHeight = this.cameras.main.height

    const sceneWidth = this.cameras.main.width

    // set texture of images based on the background mode

    if (data.backgroundMode === BackgroundMode.DAY) {

      this.backdropKey = 'backdrop\_day'

      this.cloudKey = 'cloud\_day'

      this.cameras.main.setBackgroundColor('#c6eefc')

    } else {

      this.backdropKey = 'backdrop\_night'

      this.cloudKey = 'cloud\_night'

      this.cameras.main.setBackgroundColor('#2c4464')

    }

    // Add backdrop image

    const backdropImage = this.add.image(sceneWidth / 2, sceneHeight / 2, this.backdropKey)

    const scale = Math.max(sceneWidth / backdropImage.width, sceneHeight / backdropImage.height)

    backdropImage.setScale(scale).setScrollFactor(0)

    // Add sun or moon image

    const sunMoonImage = this.add.image(sceneWidth / 2, sceneHeight / 2, 'sun\_moon')

    const scale2 = Math.max(sceneWidth / sunMoonImage.width, sceneHeight / sunMoonImage.height)

    sunMoonImage.setScale(scale2).setScrollFactor(0)

    // Add 24 clouds at random positions and with random speeds

    const frames = this.textures.get(this.cloudKey).getFrameNames()

    this.cloud = this.physics.add.group()

    for (let i = 0; i < 24; i++) {

      const x = Phaser.Math.RND.between(-sceneWidth \* 0.5, sceneWidth \* 1.5)

      const y = Phaser.Math.RND.between(sceneHeight \* 0.2, sceneHeight \* 0.8)

      const velocity = Phaser.Math.RND.between(15, 30)

      this.cloud

        .get(x, y, this.cloudKey, frames[i % 6])

        .setScale(3)

        .setVelocity(velocity, 0)

    }

  }

  update(t: number, dt: number) {

    this.physics.world.wrap(this.cloud, 500)

  }

}

Scenes/bootstrap.ts

import Phaser from 'phaser'

import Network from '../services/Network'

import { BackgroundMode } from '../../../types/BackgroundMode'

import store from '../stores'

import { setRoomJoined } from '../stores/RoomStore'

export default class Bootstrap extends Phaser.Scene {

  private preloadComplete = false

  network!: Network

  constructor() {

    super('bootstrap')

  }

  preload() {

    this.load.atlas(

      'cloud\_day',

      'assets/background/cloud\_day.png',

      'assets/background/cloud\_day.json'

    )

    this.load.image('backdrop\_day', 'assets/background/backdrop\_day.png')

    this.load.atlas(

      'cloud\_night',

      'assets/background/cloud\_night.png',

      'assets/background/cloud\_night.json'

    )

    this.load.image('backdrop\_night', 'assets/background/backdrop\_night.png')

    this.load.image('sun\_moon', 'assets/background/sun\_moon.png')

    this.load.tilemapTiledJSON('tilemap', 'assets/map/map.json')

    this.load.spritesheet('tiles\_wall', 'assets/map/FloorAndGround.png', {

      frameWidth: 32,

      frameHeight: 32,

    })

    this.load.spritesheet('chairs', 'assets/items/chair.png', {

      frameWidth: 32,

      frameHeight: 64,

    })

    this.load.spritesheet('computers', 'assets/items/computer.png', {

      frameWidth: 96,

      frameHeight: 64,

    })

    this.load.spritesheet('whiteboards', 'assets/items/whiteboard.png', {

      frameWidth: 64,

      frameHeight: 64,

    })

    this.load.spritesheet('vendingmachines', 'assets/items/vendingmachine.png', {

      frameWidth: 48,

      frameHeight: 72,

    })

    this.load.spritesheet('office', 'assets/tileset/Modern\_Office\_Black\_Shadow.png', {

      frameWidth: 32,

      frameHeight: 32,

    })

    this.load.spritesheet('basement', 'assets/tileset/Basement.png', {

      frameWidth: 32,

      frameHeight: 32,

    })

    this.load.spritesheet('generic', 'assets/tileset/Generic.png', {

      frameWidth: 32,

      frameHeight: 32,

    })

    this.load.spritesheet('adam', 'assets/character/adam.png', {

      frameWidth: 32,

      frameHeight: 48,

    })

    this.load.spritesheet('ash', 'assets/character/ash.png', {

      frameWidth: 32,

      frameHeight: 48,

    })

    this.load.spritesheet('lucy', 'assets/character/lucy.png', {

      frameWidth: 32,

      frameHeight: 48,

    })

    this.load.spritesheet('nancy', 'assets/character/nancy.png', {

      frameWidth: 32,

      frameHeight: 48,

    })

    this.load.on('complete', () => {

      this.preloadComplete = true

      this.launchBackground(store.getState().user.backgroundMode)

    })

  }

  init() {

    this.network = new Network()

  }

  private launchBackground(backgroundMode: BackgroundMode) {

    this.scene.launch('background', { backgroundMode })

  }

  launchGame() {

    if (!this.preloadComplete) return

    this.network.webRTC?.checkPreviousPermission()

    this.scene.launch('game', {

      network: this.network,

    })

    // update Redux state

    store.dispatch(setRoomJoined(true))

  }

  changeBackgroundMode(backgroundMode: BackgroundMode) {

    this.scene.stop('background')

    this.launchBackground(backgroundMode)

  }

}

Scenes/games.ts

import Phaser from 'phaser'

// import { debugDraw } from '../utils/debug'

import { createCharacterAnims } from '../anims/CharacterAnims'

import Item from '../items/Item'

import Chair from '../items/Chair'

import Computer from '../items/Computer'

import Whiteboard from '../items/Whiteboard'

import VendingMachine from '../items/VendingMachine'

import '../characters/MyPlayer'

import '../characters/OtherPlayer'

import MyPlayer from '../characters/MyPlayer'

import OtherPlayer from '../characters/OtherPlayer'

import PlayerSelector from '../characters/PlayerSelector'

import Network from '../services/Network'

import { IPlayer } from '../../../types/IOfficeState'

import { PlayerBehavior } from '../../../types/PlayerBehavior'

import { ItemType } from '../../../types/Items'

import store from '../stores'

import { setFocused, setShowChat } from '../stores/ChatStore'

import { NavKeys, Keyboard } from '../../../types/KeyboardState'

export default class Game extends Phaser.Scene {

  network!: Network

  private cursors!: NavKeys

  private keyE!: Phaser.Input.Keyboard.Key

  private keyR!: Phaser.Input.Keyboard.Key

  private map!: Phaser.Tilemaps.Tilemap

  myPlayer!: MyPlayer

  private playerSelector!: Phaser.GameObjects.Zone

  private otherPlayers!: Phaser.Physics.Arcade.Group

  private otherPlayerMap = new Map<string, OtherPlayer>()

  computerMap = new Map<string, Computer>()

  private whiteboardMap = new Map<string, Whiteboard>()

  constructor() {

    super('game')

  }

  registerKeys() {

    this.cursors = {

      ...this.input.keyboard.createCursorKeys(),

      ...(this.input.keyboard.addKeys('W,S,A,D') as Keyboard),

    }

    // maybe we can have a dedicated method for adding keys if more keys are needed in the future

    this.keyE = this.input.keyboard.addKey('E')

    this.keyR = this.input.keyboard.addKey('R')

    this.input.keyboard.disableGlobalCapture()

    this.input.keyboard.on('keydown-ENTER', (event) => {

      store.dispatch(setShowChat(true))

      store.dispatch(setFocused(true))

    })

    this.input.keyboard.on('keydown-ESC', (event) => {

      store.dispatch(setShowChat(false))

    })

  }

  disableKeys() {

    this.input.keyboard.enabled = false

  }

  enableKeys() {

    this.input.keyboard.enabled = true

  }

  create(data: { network: Network }) {

    if (!data.network) {

      throw new Error('server instance missing')

    } else {

      this.network = data.network

    }

    createCharacterAnims(this.anims)

    this.map = this.make.tilemap({ key: 'tilemap' })

    const FloorAndGround = this.map.addTilesetImage('FloorAndGround', 'tiles\_wall')

    const groundLayer = this.map.createLayer('Ground', FloorAndGround)

    groundLayer.setCollisionByProperty({ collides: true })

    // debugDraw(groundLayer, this)

    this.myPlayer = this.add.myPlayer(705, 500, 'adam', this.network.mySessionId)

    this.playerSelector = new PlayerSelector(this, 0, 0, 16, 16)

    // import chair objects from Tiled map to Phaser

    const chairs = this.physics.add.staticGroup({ classType: Chair })

    const chairLayer = this.map.getObjectLayer('Chair')

    chairLayer.objects.forEach((chairObj) => {

      const item = this.addObjectFromTiled(chairs, chairObj, 'chairs', 'chair') as Chair

      // custom properties[0] is the object direction specified in Tiled

      item.itemDirection = chairObj.properties[0].value

    })

    // import computers objects from Tiled map to Phaser

    const computers = this.physics.add.staticGroup({ classType: Computer })

    const computerLayer = this.map.getObjectLayer('Computer')

    computerLayer.objects.forEach((obj, i) => {

      const item = this.addObjectFromTiled(computers, obj, 'computers', 'computer') as Computer

      item.setDepth(item.y + item.height \* 0.27)

      const id = `${i}`

      item.id = id

      this.computerMap.set(id, item)

    })

    // import whiteboards objects from Tiled map to Phaser

    const whiteboards = this.physics.add.staticGroup({ classType: Whiteboard })

    const whiteboardLayer = this.map.getObjectLayer('Whiteboard')

    whiteboardLayer.objects.forEach((obj, i) => {

      const item = this.addObjectFromTiled(

        whiteboards,

        obj,

        'whiteboards',

        'whiteboard'

      ) as Whiteboard

      const id = `${i}`

      item.id = id

      this.whiteboardMap.set(id, item)

    })

    // import vending machine objects from Tiled map to Phaser

    const vendingMachines = this.physics.add.staticGroup({ classType: VendingMachine })

    const vendingMachineLayer = this.map.getObjectLayer('VendingMachine')

    vendingMachineLayer.objects.forEach((obj, i) => {

      this.addObjectFromTiled(vendingMachines, obj, 'vendingmachines', 'vendingmachine')

    })

    // import other objects from Tiled map to Phaser

    this.addGroupFromTiled('Wall', 'tiles\_wall', 'FloorAndGround', false)

    this.addGroupFromTiled('Objects', 'office', 'Modern\_Office\_Black\_Shadow', false)

    this.addGroupFromTiled('ObjectsOnCollide', 'office', 'Modern\_Office\_Black\_Shadow', true)

    this.addGroupFromTiled('GenericObjects', 'generic', 'Generic', false)

    this.addGroupFromTiled('GenericObjectsOnCollide', 'generic', 'Generic', true)

    this.addGroupFromTiled('Basement', 'basement', 'Basement', true)

    this.otherPlayers = this.physics.add.group({ classType: OtherPlayer })

    this.cameras.main.zoom = 1.5

    this.cameras.main.startFollow(this.myPlayer, true)

    this.physics.add.collider([this.myPlayer, this.myPlayer.playerContainer], groundLayer)

    this.physics.add.collider([this.myPlayer, this.myPlayer.playerContainer], vendingMachines)

    this.physics.add.overlap(

      this.playerSelector,

      [chairs, computers, whiteboards, vendingMachines],

      this.handleItemSelectorOverlap,

      undefined,

      this

    )

    this.physics.add.overlap(

      this.myPlayer,

      this.otherPlayers,

      this.handlePlayersOverlap,

      undefined,

      this

    )

    // register network event listeners

    this.network.onPlayerJoined(this.handlePlayerJoined, this)

    this.network.onPlayerLeft(this.handlePlayerLeft, this)

    this.network.onMyPlayerReady(this.handleMyPlayerReady, this)

    this.network.onMyPlayerVideoConnected(this.handleMyVideoConnected, this)

    this.network.onPlayerUpdated(this.handlePlayerUpdated, this)

    this.network.onItemUserAdded(this.handleItemUserAdded, this)

    this.network.onItemUserRemoved(this.handleItemUserRemoved, this)

    this.network.onChatMessageAdded(this.handleChatMessageAdded, this)

  }

  private handleItemSelectorOverlap(playerSelector, selectionItem) {

    const currentItem = playerSelector.selectedItem as Item

    // currentItem is undefined if nothing was perviously selected

    if (currentItem) {

      // if the selection has not changed, do nothing

      if (currentItem === selectionItem || currentItem.depth >= selectionItem.depth) {

        return

      }

      // if selection changes, clear pervious dialog

      if (this.myPlayer.playerBehavior !== PlayerBehavior.SITTING) currentItem.clearDialogBox()

    }

    // set selected item and set up new dialog

    playerSelector.selectedItem = selectionItem

    selectionItem.onOverlapDialog()

  }

  private addObjectFromTiled(

    group: Phaser.Physics.Arcade.StaticGroup,

    object: Phaser.Types.Tilemaps.TiledObject,

    key: string,

    tilesetName: string

  ) {

    const actualX = object.x! + object.width! \* 0.5

    const actualY = object.y! - object.height! \* 0.5

    const obj = group

      .get(actualX, actualY, key, object.gid! - this.map.getTileset(tilesetName).firstgid)

      .setDepth(actualY)

    return obj

  }

  private addGroupFromTiled(

    objectLayerName: string,

    key: string,

    tilesetName: string,

    collidable: boolean

  ) {

    const group = this.physics.add.staticGroup()

    const objectLayer = this.map.getObjectLayer(objectLayerName)

    objectLayer.objects.forEach((object) => {

      const actualX = object.x! + object.width! \* 0.5

      const actualY = object.y! - object.height! \* 0.5

      group

        .get(actualX, actualY, key, object.gid! - this.map.getTileset(tilesetName).firstgid)

        .setDepth(actualY)

    })

    if (this.myPlayer && collidable)

      this.physics.add.collider([this.myPlayer, this.myPlayer.playerContainer], group)

  }

  // function to add new player to the otherPlayer group

  private handlePlayerJoined(newPlayer: IPlayer, id: string) {

    const otherPlayer = this.add.otherPlayer(newPlayer.x, newPlayer.y, 'adam', id, newPlayer.name)

    this.otherPlayers.add(otherPlayer)

    this.otherPlayerMap.set(id, otherPlayer)

  }

  // function to remove the player who left from the otherPlayer group

  private handlePlayerLeft(id: string) {

    if (this.otherPlayerMap.has(id)) {

      const otherPlayer = this.otherPlayerMap.get(id)

      if (!otherPlayer) return

      this.otherPlayers.remove(otherPlayer, true, true)

      this.otherPlayerMap.delete(id)

    }

  }

  private handleMyPlayerReady() {

    this.myPlayer.readyToConnect = true

  }

  private handleMyVideoConnected() {

    this.myPlayer.videoConnected = true

  }

  // function to update target position upon receiving player updates

  private handlePlayerUpdated(field: string, value: number | string, id: string) {

    const otherPlayer = this.otherPlayerMap.get(id)

    otherPlayer?.updateOtherPlayer(field, value)

  }

  private handlePlayersOverlap(myPlayer, otherPlayer) {

    otherPlayer.makeCall(myPlayer, this.network?.webRTC)

  }

  private handleItemUserAdded(playerId: string, itemId: string, itemType: ItemType) {

    if (itemType === ItemType.COMPUTER) {

      const computer = this.computerMap.get(itemId)

      computer?.addCurrentUser(playerId)

    } else if (itemType === ItemType.WHITEBOARD) {

      const whiteboard = this.whiteboardMap.get(itemId)

      whiteboard?.addCurrentUser(playerId)

    }

  }

  private handleItemUserRemoved(playerId: string, itemId: string, itemType: ItemType) {

    if (itemType === ItemType.COMPUTER) {

      const computer = this.computerMap.get(itemId)

      computer?.removeCurrentUser(playerId)

    } else if (itemType === ItemType.WHITEBOARD) {

      const whiteboard = this.whiteboardMap.get(itemId)

      whiteboard?.removeCurrentUser(playerId)

    }

  }

  private handleChatMessageAdded(playerId: string, content: string) {

    const otherPlayer = this.otherPlayerMap.get(playerId)

    otherPlayer?.updateDialogBubble(content)

  }

  update(t: number, dt: number) {

    if (this.myPlayer && this.network) {

      this.playerSelector.update(this.myPlayer, this.cursors)

      this.myPlayer.update(this.playerSelector, this.cursors, this.keyE, this.keyR, this.network)

    }

  }

}

Services/network.ts

import { Client, Room } from 'colyseus.js'

import { IComputer, IOfficeState, IPlayer, IWhiteboard } from '../../../types/IOfficeState'

import { Message } from '../../../types/Messages'

import { IRoomData, RoomType } from '../../../types/Rooms'

import { ItemType } from '../../../types/Items'

import WebRTC from '../web/WebRTC'

import { phaserEvents, Event } from '../events/EventCenter'

import store from '../stores'

import { setSessionId, setPlayerNameMap, removePlayerNameMap } from '../stores/UserStore'

import {

  setLobbyJoined,

  setJoinedRoomData,

  setAvailableRooms,

  addAvailableRooms,

  removeAvailableRooms,

} from '../stores/RoomStore'

import {

  pushChatMessage,

  pushPlayerJoinedMessage,

  pushPlayerLeftMessage,

} from '../stores/ChatStore'

import { setWhiteboardUrls } from '../stores/WhiteboardStore'

export default class Network {

  private client: Client

  private room?: Room<IOfficeState>

  private lobby!: Room

  webRTC?: WebRTC

  mySessionId!: string

  constructor() {

    const protocol = window.location.protocol.replace('http', 'ws')

    const endpoint =

      process.env.NODE\_ENV === 'production'

        ? import.meta.env.VITE\_SERVER\_URL

        : `${protocol}//${window.location.hostname}:2567`

    this.client = new Client(endpoint)

    this.joinLobbyRoom().then(() => {

      store.dispatch(setLobbyJoined(true))

    })

    phaserEvents.on(Event.MY\_PLAYER\_NAME\_CHANGE, this.updatePlayerName, this)

    phaserEvents.on(Event.MY\_PLAYER\_TEXTURE\_CHANGE, this.updatePlayer, this)

    phaserEvents.on(Event.PLAYER\_DISCONNECTED, this.playerStreamDisconnect, this)

  }

  /\*\*

   \* method to join Colyseus' built-in LobbyRoom, which automatically notifies

   \* connected clients whenever rooms with "realtime listing" have updates

   \*/

  async joinLobbyRoom() {

    this.lobby = await this.client.joinOrCreate(RoomType.LOBBY)

    this.lobby.onMessage('rooms', (rooms) => {

      store.dispatch(setAvailableRooms(rooms))

    })

    this.lobby.onMessage('+', ([roomId, room]) => {

      store.dispatch(addAvailableRooms({ roomId, room }))

    })

    this.lobby.onMessage('-', (roomId) => {

      store.dispatch(removeAvailableRooms(roomId))

    })

  }

  // method to join the public lobby

  async joinOrCreatePublic() {

    this.room = await this.client.joinOrCreate(RoomType.PUBLIC)

    this.initialize()

  }

  // method to join a custom room

  async joinCustomById(roomId: string, password: string | null) {

    this.room = await this.client.joinById(roomId, { password })

    this.initialize()

  }

  // method to create a custom room

  async createCustom(roomData: IRoomData) {

    const { name, description, password, autoDispose } = roomData

    this.room = await this.client.create(RoomType.CUSTOM, {

      name,

      description,

      password,

      autoDispose,

    })

    this.initialize()

  }

  // set up all network listeners before the game starts

  initialize() {

    if (!this.room) return

    this.lobby.leave()

    this.mySessionId = this.room.sessionId

    store.dispatch(setSessionId(this.room.sessionId))

    this.webRTC = new WebRTC(this.mySessionId, this)

    // new instance added to the players MapSchema

    this.room.state.players.onAdd = (player: IPlayer, key: string) => {

      if (key === this.mySessionId) return

      // track changes on every child object inside the players MapSchema

      player.onChange = (changes) => {

        changes.forEach((change) => {

          const { field, value } = change

          phaserEvents.emit(Event.PLAYER\_UPDATED, field, value, key)

          // when a new player finished setting up player name

          if (field === 'name' && value !== '') {

            phaserEvents.emit(Event.PLAYER\_JOINED, player, key)

            store.dispatch(setPlayerNameMap({ id: key, name: value }))

            store.dispatch(pushPlayerJoinedMessage(value))

          }

        })

      }

    }

    // an instance removed from the players MapSchema

    this.room.state.players.onRemove = (player: IPlayer, key: string) => {

      phaserEvents.emit(Event.PLAYER\_LEFT, key)

      this.webRTC?.deleteVideoStream(key)

      this.webRTC?.deleteOnCalledVideoStream(key)

      store.dispatch(pushPlayerLeftMessage(player.name))

      store.dispatch(removePlayerNameMap(key))

    }

    // new instance added to the computers MapSchema

    this.room.state.computers.onAdd = (computer: IComputer, key: string) => {

      // track changes on every child object's connectedUser

      computer.connectedUser.onAdd = (item, index) => {

        phaserEvents.emit(Event.ITEM\_USER\_ADDED, item, key, ItemType.COMPUTER)

      }

      computer.connectedUser.onRemove = (item, index) => {

        phaserEvents.emit(Event.ITEM\_USER\_REMOVED, item, key, ItemType.COMPUTER)

      }

    }

    // new instance added to the whiteboards MapSchema

    this.room.state.whiteboards.onAdd = (whiteboard: IWhiteboard, key: string) => {

      store.dispatch(

        setWhiteboardUrls({

          whiteboardId: key,

          roomId: whiteboard.roomId,

        })

      )

      // track changes on every child object's connectedUser

      whiteboard.connectedUser.onAdd = (item, index) => {

        phaserEvents.emit(Event.ITEM\_USER\_ADDED, item, key, ItemType.WHITEBOARD)

      }

      whiteboard.connectedUser.onRemove = (item, index) => {

        phaserEvents.emit(Event.ITEM\_USER\_REMOVED, item, key, ItemType.WHITEBOARD)

      }

    }

    // new instance added to the chatMessages ArraySchema

    this.room.state.chatMessages.onAdd = (item, index) => {

      store.dispatch(pushChatMessage(item))

    }

    // when the server sends room data

    this.room.onMessage(Message.SEND\_ROOM\_DATA, (content) => {

      store.dispatch(setJoinedRoomData(content))

    })

    // when a user sends a message

    this.room.onMessage(Message.ADD\_CHAT\_MESSAGE, ({ clientId, content }) => {

      phaserEvents.emit(Event.UPDATE\_DIALOG\_BUBBLE, clientId, content)

    })

    // when a peer disconnects with myPeer

    this.room.onMessage(Message.DISCONNECT\_STREAM, (clientId: string) => {

      this.webRTC?.deleteOnCalledVideoStream(clientId)

    })

    // when a computer user stops sharing screen

    this.room.onMessage(Message.STOP\_SCREEN\_SHARE, (clientId: string) => {

      const computerState = store.getState().computer

      computerState.shareScreenManager?.onUserLeft(clientId)

    })

  }

  // method to register event listener and call back function when a item user added

  onChatMessageAdded(callback: (playerId: string, content: string) => void, context?: any) {

    phaserEvents.on(Event.UPDATE\_DIALOG\_BUBBLE, callback, context)

  }

  // method to register event listener and call back function when a item user added

  onItemUserAdded(

    callback: (playerId: string, key: string, itemType: ItemType) => void,

    context?: any

  ) {

    phaserEvents.on(Event.ITEM\_USER\_ADDED, callback, context)

  }

  // method to register event listener and call back function when a item user removed

  onItemUserRemoved(

    callback: (playerId: string, key: string, itemType: ItemType) => void,

    context?: any

  ) {

    phaserEvents.on(Event.ITEM\_USER\_REMOVED, callback, context)

  }

  // method to register event listener and call back function when a player joined

  onPlayerJoined(callback: (Player: IPlayer, key: string) => void, context?: any) {

    phaserEvents.on(Event.PLAYER\_JOINED, callback, context)

  }

  // method to register event listener and call back function when a player left

  onPlayerLeft(callback: (key: string) => void, context?: any) {

    phaserEvents.on(Event.PLAYER\_LEFT, callback, context)

  }

  // method to register event listener and call back function when myPlayer is ready to connect

  onMyPlayerReady(callback: (key: string) => void, context?: any) {

    phaserEvents.on(Event.MY\_PLAYER\_READY, callback, context)

  }

  // method to register event listener and call back function when my video is connected

  onMyPlayerVideoConnected(callback: (key: string) => void, context?: any) {

    phaserEvents.on(Event.MY\_PLAYER\_VIDEO\_CONNECTED, callback, context)

  }

  // method to register event listener and call back function when a player updated

  onPlayerUpdated(

    callback: (field: string, value: number | string, key: string) => void,

    context?: any

  ) {

    phaserEvents.on(Event.PLAYER\_UPDATED, callback, context)

  }

  // method to send player updates to Colyseus server

  updatePlayer(currentX: number, currentY: number, currentAnim: string) {

    this.room?.send(Message.UPDATE\_PLAYER, { x: currentX, y: currentY, anim: currentAnim })

  }

  // method to send player name to Colyseus server

  updatePlayerName(currentName: string) {

    this.room?.send(Message.UPDATE\_PLAYER\_NAME, { name: currentName })

  }

  // method to send ready-to-connect signal to Colyseus server

  readyToConnect() {

    this.room?.send(Message.READY\_TO\_CONNECT)

    phaserEvents.emit(Event.MY\_PLAYER\_READY)

  }

  // method to send ready-to-connect signal to Colyseus server

  videoConnected() {

    this.room?.send(Message.VIDEO\_CONNECTED)

    phaserEvents.emit(Event.MY\_PLAYER\_VIDEO\_CONNECTED)

  }

  // method to send stream-disconnection signal to Colyseus server

  playerStreamDisconnect(id: string) {

    this.room?.send(Message.DISCONNECT\_STREAM, { clientId: id })

    this.webRTC?.deleteVideoStream(id)

  }

  connectToComputer(id: string) {

    this.room?.send(Message.CONNECT\_TO\_COMPUTER, { computerId: id })

  }

  disconnectFromComputer(id: string) {

    this.room?.send(Message.DISCONNECT\_FROM\_COMPUTER, { computerId: id })

  }

  connectToWhiteboard(id: string) {

    this.room?.send(Message.CONNECT\_TO\_WHITEBOARD, { whiteboardId: id })

  }

  disconnectFromWhiteboard(id: string) {

    this.room?.send(Message.DISCONNECT\_FROM\_WHITEBOARD, { whiteboardId: id })

  }

  onStopScreenShare(id: string) {

    this.room?.send(Message.STOP\_SCREEN\_SHARE, { computerId: id })

  }

  addChatMessage(content: string) {

    this.room?.send(Message.ADD\_CHAT\_MESSAGE, { content: content })

  }

}

Chatstore.ts

import { createSlice, PayloadAction } from '@reduxjs/toolkit'

import { IChatMessage } from '../../../types/IOfficeState'

import phaserGame from '../PhaserGame'

import Game from '../scenes/Game'

export enum MessageType {

  PLAYER\_JOINED,

  PLAYER\_LEFT,

  REGULAR\_MESSAGE,

}

export const chatSlice = createSlice({

  name: 'chat',

  initialState: {

    chatMessages: new Array<{ messageType: MessageType; chatMessage: IChatMessage }>(),

    focused: false,

    showChat: true,

  },

  reducers: {

    pushChatMessage: (state, action: PayloadAction<IChatMessage>) => {

      state.chatMessages.push({

        messageType: MessageType.REGULAR\_MESSAGE,

        chatMessage: action.payload,

      })

    },

    pushPlayerJoinedMessage: (state, action: PayloadAction<string>) => {

      state.chatMessages.push({

        messageType: MessageType.PLAYER\_JOINED,

        chatMessage: {

          createdAt: new Date().getTime(),

          author: action.payload,

          content: 'joined the lobby',

        } as IChatMessage,

      })

    },

    pushPlayerLeftMessage: (state, action: PayloadAction<string>) => {

      state.chatMessages.push({

        messageType: MessageType.PLAYER\_LEFT,

        chatMessage: {

          createdAt: new Date().getTime(),

          author: action.payload,

          content: 'left the lobby',

        } as IChatMessage,

      })

    },

    setFocused: (state, action: PayloadAction<boolean>) => {

      const game = phaserGame.scene.keys.game as Game

      action.payload ? game.disableKeys() : game.enableKeys()

      state.focused = action.payload

    },

    setShowChat: (state, action: PayloadAction<boolean>) => {

      state.showChat = action.payload

    },

  },

})

export const {

  pushChatMessage,

  pushPlayerJoinedMessage,

  pushPlayerLeftMessage,

  setFocused,

  setShowChat,

} = chatSlice.actions

export default chatSlice.reducer

computerStore.ts

import Peer from 'peerjs'

import { createSlice, PayloadAction } from '@reduxjs/toolkit'

import ShareScreenManager from '../web/ShareScreenManager'

import phaserGame from '../PhaserGame'

import Game from '../scenes/Game'

import { sanitizeId } from '../util'

interface ComputerState {

  computerDialogOpen: boolean

  computerId: null | string

  myStream: null | MediaStream

  peerStreams: Map<

    string,

    {

      stream: MediaStream

      call: Peer.MediaConnection

    }

  >

  shareScreenManager: null | ShareScreenManager

}

const initialState: ComputerState = {

  computerDialogOpen: false,

  computerId: null,

  myStream: null,

  peerStreams: new Map(),

  shareScreenManager: null,

}

export const computerSlice = createSlice({

  name: 'computer',

  initialState,

  reducers: {

    openComputerDialog: (

      state,

      action: PayloadAction<{ computerId: string; myUserId: string }>

    ) => {

      if (!state.shareScreenManager) {

        state.shareScreenManager = new ShareScreenManager(action.payload.myUserId)

      }

      const game = phaserGame.scene.keys.game as Game

      game.disableKeys()

      state.shareScreenManager.onOpen()

      state.computerDialogOpen = true

      state.computerId = action.payload.computerId

    },

    closeComputerDialog: (state) => {

      // Tell server the computer dialog is closed.

      const game = phaserGame.scene.keys.game as Game

      game.enableKeys()

      game.network.disconnectFromComputer(state.computerId!)

      for (const { call } of state.peerStreams.values()) {

        call.close()

      }

      state.shareScreenManager?.onClose()

      state.computerDialogOpen = false

      state.myStream = null

      state.computerId = null

      state.peerStreams.clear()

    },

    setMyStream: (state, action: PayloadAction<null | MediaStream>) => {

      state.myStream = action.payload

    },

    addVideoStream: (

      state,

      action: PayloadAction<{ id: string; call: Peer.MediaConnection; stream: MediaStream }>

    ) => {

      state.peerStreams.set(sanitizeId(action.payload.id), {

        call: action.payload.call,

        stream: action.payload.stream,

      })

    },

    removeVideoStream: (state, action: PayloadAction<string>) => {

      state.peerStreams.delete(sanitizeId(action.payload))

    },

  },

})

export const {

  closeComputerDialog,

  openComputerDialog,

  setMyStream,

  addVideoStream,

  removeVideoStream,

} = computerSlice.actions

export default computerSlice.reducer

stores/Index.ts

import { enableMapSet } from 'immer'

import { configureStore } from '@reduxjs/toolkit'

import userReducer from './UserStore'

import computerReducer from './ComputerStore'

import whiteboardReducer from './WhiteboardStore'

import chatReducer from './ChatStore'

import roomReducer from './RoomStore'

enableMapSet()

const store = configureStore({

  reducer: {

    user: userReducer,

    computer: computerReducer,

    whiteboard: whiteboardReducer,

    chat: chatReducer,

    room: roomReducer,

  },

  // Temporary disable serialize check for redux as we store MediaStream in ComputerStore.

  // https://stackoverflow.com/a/63244831

  middleware: (getDefaultMiddleware) =>

    getDefaultMiddleware({

      serializableCheck: false,

    }),

})

// Infer the `RootState` and `AppDispatch` types from the store itself

export type RootState = ReturnType<typeof store.getState>

// Inferred type: {posts: PostsState, comments: CommentsState, users: UsersState}

export type AppDispatch = typeof store.dispatch

export default store

RoomStore.ts

import { createSlice, PayloadAction } from '@reduxjs/toolkit'

import { RoomAvailable } from 'colyseus.js'

import { RoomType } from '../../../types/Rooms'

interface RoomInterface extends RoomAvailable {

  name?: string

}

/\*\*

 \* Colyseus' real time room list always includes the public lobby so we have to remove it manually.

 \*/

const isCustomRoom = (room: RoomInterface) => {

  return room.name === RoomType.CUSTOM

}

export const roomSlice = createSlice({

  name: 'room',

  initialState: {

    lobbyJoined: false,

    roomJoined: false,

    roomId: '',

    roomName: '',

    roomDescription: '',

    availableRooms: new Array<RoomAvailable>(),

  },

  reducers: {

    setLobbyJoined: (state, action: PayloadAction<boolean>) => {

      state.lobbyJoined = action.payload

    },

    setRoomJoined: (state, action: PayloadAction<boolean>) => {

      state.roomJoined = action.payload

    },

    setJoinedRoomData: (

      state,

      action: PayloadAction<{ id: string; name: string; description: string }>

    ) => {

      state.roomId = action.payload.id

      state.roomName = action.payload.name

      state.roomDescription = action.payload.description

    },

    setAvailableRooms: (state, action: PayloadAction<RoomAvailable[]>) => {

      state.availableRooms = action.payload.filter((room) => isCustomRoom(room))

    },

    addAvailableRooms: (state, action: PayloadAction<{ roomId: string; room: RoomAvailable }>) => {

      if (!isCustomRoom(action.payload.room)) return

      const roomIndex = state.availableRooms.findIndex(

        (room) => room.roomId === action.payload.roomId

      )

      if (roomIndex !== -1) {

        state.availableRooms[roomIndex] = action.payload.room

      } else {

        state.availableRooms.push(action.payload.room)

      }

    },

    removeAvailableRooms: (state, action: PayloadAction<string>) => {

      state.availableRooms = state.availableRooms.filter((room) => room.roomId !== action.payload)

    },

  },

})

export const {

  setLobbyJoined,

  setRoomJoined,

  setJoinedRoomData,

  setAvailableRooms,

  addAvailableRooms,

  removeAvailableRooms,

} = roomSlice.actions

export default roomSlice.reducer

userStore.ts

import { createSlice, PayloadAction } from '@reduxjs/toolkit'

import { sanitizeId } from '../util'

import { BackgroundMode } from '../../../types/BackgroundMode'

import phaserGame from '../PhaserGame'

import Bootstrap from '../scenes/Bootstrap'

export function getInitialBackgroundMode() {

  const currentHour = new Date().getHours()

  return currentHour > 6 && currentHour <= 18 ? BackgroundMode.DAY : BackgroundMode.NIGHT

}

export const userSlice = createSlice({

  name: 'user',

  initialState: {

    backgroundMode: getInitialBackgroundMode(),

    sessionId: '',

    videoConnected: false,

    loggedIn: false,

    playerNameMap: new Map<string, string>(),

    showJoystick: window.innerWidth < 650,

  },

  reducers: {

    toggleBackgroundMode: (state) => {

      const newMode =

        state.backgroundMode === BackgroundMode.DAY ? BackgroundMode.NIGHT : BackgroundMode.DAY

      state.backgroundMode = newMode

      const bootstrap = phaserGame.scene.keys.bootstrap as Bootstrap

      bootstrap.changeBackgroundMode(newMode)

    },

    setSessionId: (state, action: PayloadAction<string>) => {

      state.sessionId = action.payload

    },

    setVideoConnected: (state, action: PayloadAction<boolean>) => {

      state.videoConnected = action.payload

    },

    setLoggedIn: (state, action: PayloadAction<boolean>) => {

      state.loggedIn = action.payload

    },

    setPlayerNameMap: (state, action: PayloadAction<{ id: string; name: string }>) => {

      state.playerNameMap.set(sanitizeId(action.payload.id), action.payload.name)

    },

    removePlayerNameMap: (state, action: PayloadAction<string>) => {

      state.playerNameMap.delete(sanitizeId(action.payload))

    },

    setShowJoystick: (state, action: PayloadAction<boolean>) => {

      state.showJoystick = action.payload

    },

  },

})

export const {

  toggleBackgroundMode,

  setSessionId,

  setVideoConnected,

  setLoggedIn,

  setPlayerNameMap,

  removePlayerNameMap,

  setShowJoystick,

} = userSlice.actions

export default userSlice.reducer

WhitebordStore.ts

import { createSlice, PayloadAction } from '@reduxjs/toolkit'

import phaserGame from '../PhaserGame'

import Game from '../scenes/Game'

interface WhiteboardState {

  whiteboardDialogOpen: boolean

  whiteboardId: null | string

  whiteboardUrl: null | string

  urls: Map<string, string>

}

const initialState: WhiteboardState = {

  whiteboardDialogOpen: false,

  whiteboardId: null,

  whiteboardUrl: null,

  urls: new Map(),

}

export const whiteboardSlice = createSlice({

  name: 'whiteboard',

  initialState,

  reducers: {

    openWhiteboardDialog: (state, action: PayloadAction<string>) => {

      state.whiteboardDialogOpen = true

      state.whiteboardId = action.payload

      const url = state.urls.get(action.payload)

      if (url) state.whiteboardUrl = url

      const game = phaserGame.scene.keys.game as Game

      game.disableKeys()

    },

    closeWhiteboardDialog: (state) => {

      const game = phaserGame.scene.keys.game as Game

      game.enableKeys()

      game.network.disconnectFromWhiteboard(state.whiteboardId!)

      state.whiteboardDialogOpen = false

      state.whiteboardId = null

      state.whiteboardUrl = null

    },

    setWhiteboardUrls: (state, action: PayloadAction<{ whiteboardId: string; roomId: string }>) => {

      state.urls.set(

        action.payload.whiteboardId,

        `https://wbo.ophir.dev/boards/sky-office-${action.payload.roomId}`

      )

    },

  },

})

export const { openWhiteboardDialog, closeWhiteboardDialog, setWhiteboardUrls } =

  whiteboardSlice.actions

export default whiteboardSlice.reducer

debug.ts

import Phaser from 'phaser'

export const debugDraw = (layer: Phaser.Tilemaps.TilemapLayer, scene: Phaser.Scene) => {

  const debugGraphics = scene.add.graphics().setAlpha(0.7)

  layer.renderDebug(debugGraphics, {

    tileColor: null,

    collidingTileColor: new Phaser.Display.Color(243, 234, 48, 255),

    faceColor: new Phaser.Display.Color(40, 39, 37, 255),

  })

}

// export { debugDraw }

Helpers.ts

export function openURL(url: string) {

  const canOpenNewTab = window.open(url, '\_blank')

  // if the browser blocks the new tab, open the url in the current tab

  // this is the case for Safari on iOS

  if (!canOpenNewTab) {

    window.location.href = url

  }

}

SharescreenManager.ts

import Peer from 'peerjs'

import store from '../stores'

import { setMyStream, addVideoStream, removeVideoStream } from '../stores/ComputerStore'

import phaserGame from '../PhaserGame'

import Game from '../scenes/Game'

export default class ShareScreenManager {

  private myPeer: Peer

  myStream?: MediaStream

  constructor(private userId: string) {

    const sanatizedId = this.makeId(userId)

    this.myPeer = new Peer(sanatizedId)

    this.myPeer.on('error', (err) => {

      console.log('ShareScreenWebRTC err.type', err.type)

      console.error('ShareScreenWebRTC', err)

    })

    this.myPeer.on('call', (call) => {

      call.answer()

      call.on('stream', (userVideoStream) => {

        store.dispatch(addVideoStream({ id: call.peer, call, stream: userVideoStream }))

      })

      // we handled on close on our own

    })

  }

  onOpen() {

    if (this.myPeer.disconnected) {

      this.myPeer.reconnect()

    }

  }

  onClose() {

    this.stopScreenShare(false)

    this.myPeer.disconnect()

  }

  // PeerJS throws invalid\_id error if it contains some characters such as that colyseus generates.

  // https://peerjs.com/docs.html#peer-id

  // Also for screen sharing ID add a `-ss` at the end.

  private makeId(id: string) {

    return `${id.replace(/[^0-9a-z]/gi, 'G')}-ss`

  }

  startScreenShare() {

    // @ts-ignore

    navigator.mediaDevices

      ?.getDisplayMedia({

        video: true,

        audio: true,

      })

      .then((stream) => {

        // Detect when user clicks "Stop sharing" outside of our UI.

        // https://stackoverflow.com/a/25179198

        const track = stream.getVideoTracks()[0]

        if (track) {

          track.onended = () => {

            this.stopScreenShare()

          }

        }

        this.myStream = stream

        store.dispatch(setMyStream(stream))

        // Call all existing users.

        const game = phaserGame.scene.keys.game as Game

        const computerItem = game.computerMap.get(store.getState().computer.computerId!)

        if (computerItem) {

          for (const userId of computerItem.currentUsers) {

            this.onUserJoined(userId)

          }

        }

      })

  }

  // TODO(daxchen): Fix this trash hack, if we call store.dispatch here when calling

  // from onClose, it causes redux reducer cycle, this may be fixable by using thunk

  // or something.

  stopScreenShare(shouldDispatch = true) {

    this.myStream?.getTracks().forEach((track) => track.stop())

    this.myStream = undefined

    if (shouldDispatch) {

      store.dispatch(setMyStream(null))

      // Manually let all other existing users know screen sharing is stopped

      const game = phaserGame.scene.keys.game as Game

      game.network.onStopScreenShare(store.getState().computer.computerId!)

    }

  }

  onUserJoined(userId: string) {

    if (!this.myStream || userId === this.userId) return

    const sanatizedId = this.makeId(userId)

    this.myPeer.call(sanatizedId, this.myStream)

  }

  onUserLeft(userId: string) {

    if (userId === this.userId) return

    const sanatizedId = this.makeId(userId)

    store.dispatch(removeVideoStream(sanatizedId))

  }

}

WebRtc.ts

import Peer from 'peerjs'

import Network from '../services/Network'

import store from '../stores'

import { setVideoConnected } from '../stores/UserStore'

export default class WebRTC {

  private myPeer: Peer

  private peers = new Map<string, { call: Peer.MediaConnection; video: HTMLVideoElement }>()

  private onCalledPeers = new Map<string, { call: Peer.MediaConnection; video: HTMLVideoElement }>()

  private videoGrid = document.querySelector('.video-grid')

  private buttonGrid = document.querySelector('.button-grid')

  private myVideo = document.createElement('video')

  private myStream?: MediaStream

  private network: Network

  constructor(userId: string, network: Network) {

    const sanitizedId = this.replaceInvalidId(userId)

    this.myPeer = new Peer(sanitizedId)

    this.network = network

    console.log('userId:', userId)

    console.log('sanitizedId:', sanitizedId)

    this.myPeer.on('error', (err) => {

      console.log(err.type)

      console.error(err)

    })

    // mute your own video stream (you don't want to hear yourself)

    this.myVideo.muted = true

    // config peerJS

    this.initialize()

  }

  // PeerJS throws invalid\_id error if it contains some characters such as that colyseus generates.

  // https://peerjs.com/docs.html#peer-id

  private replaceInvalidId(userId: string) {

    return userId.replace(/[^0-9a-z]/gi, 'G')

  }

  initialize() {

    this.myPeer.on('call', (call) => {

      if (!this.onCalledPeers.has(call.peer)) {

        call.answer(this.myStream)

        const video = document.createElement('video')

        this.onCalledPeers.set(call.peer, { call, video })

        call.on('stream', (userVideoStream) => {

          this.addVideoStream(video, userVideoStream)

        })

      }

      // on close is triggered manually with deleteOnCalledVideoStream()

    })

  }

  // check if permission has been granted before

  checkPreviousPermission() {

    const permissionName = 'microphone' as PermissionName

    navigator.permissions?.query({ name: permissionName }).then((result) => {

      if (result.state === 'granted') this.getUserMedia(false)

    })

  }

  getUserMedia(alertOnError = true) {

    // ask the browser to get user media

    navigator.mediaDevices

      ?.getUserMedia({

        video: true,

        audio: true,

      })

      .then((stream) => {

        this.myStream = stream

        this.addVideoStream(this.myVideo, this.myStream)

        this.setUpButtons()

        store.dispatch(setVideoConnected(true))

        this.network.videoConnected()

      })

      .catch((error) => {

        if (alertOnError) window.alert('No webcam or microphone found, or permission is blocked')

      })

  }

  // method to call a peer

  connectToNewUser(userId: string) {

    if (this.myStream) {

      const sanitizedId = this.replaceInvalidId(userId)

      if (!this.peers.has(sanitizedId)) {

        console.log('calling', sanitizedId)

        const call = this.myPeer.call(sanitizedId, this.myStream)

        const video = document.createElement('video')

        this.peers.set(sanitizedId, { call, video })

        call.on('stream', (userVideoStream) => {

          this.addVideoStream(video, userVideoStream)

        })

        // on close is triggered manually with deleteVideoStream()

      }

    }

  }

  // method to add new video stream to videoGrid div

  addVideoStream(video: HTMLVideoElement, stream: MediaStream) {

    video.srcObject = stream

    video.playsInline = true

    video.addEventListener('loadedmetadata', () => {

      video.play()

    })

    if (this.videoGrid) this.videoGrid.append(video)

  }

  // method to remove video stream (when we are the host of the call)

  deleteVideoStream(userId: string) {

    const sanitizedId = this.replaceInvalidId(userId)

    if (this.peers.has(sanitizedId)) {

      const peer = this.peers.get(sanitizedId)

      peer?.call.close()

      peer?.video.remove()

      this.peers.delete(sanitizedId)

    }

  }

  // method to remove video stream (when we are the guest of the call)

  deleteOnCalledVideoStream(userId: string) {

    const sanitizedId = this.replaceInvalidId(userId)

    if (this.onCalledPeers.has(sanitizedId)) {

      const onCalledPeer = this.onCalledPeers.get(sanitizedId)

      onCalledPeer?.call.close()

      onCalledPeer?.video.remove()

      this.onCalledPeers.delete(sanitizedId)

    }

  }

  // method to set up mute/unmute and video on/off buttons

  setUpButtons() {

    const audioButton = document.createElement('button')

    audioButton.innerText = 'Mute'

    audioButton.addEventListener('click', () => {

      if (this.myStream) {

        const audioTrack = this.myStream.getAudioTracks()[0]

        if (audioTrack.enabled) {

          audioTrack.enabled = false

          audioButton.innerText = 'Unmute'

        } else {

          audioTrack.enabled = true

          audioButton.innerText = 'Mute'

        }

      }

    })

    const videoButton = document.createElement('button')

    videoButton.innerText = 'Video off'

    videoButton.addEventListener('click', () => {

      if (this.myStream) {

        const audioTrack = this.myStream.getVideoTracks()[0]

        if (audioTrack.enabled) {

          audioTrack.enabled = false

          videoButton.innerText = 'Video on'

        } else {

          audioTrack.enabled = true

          videoButton.innerText = 'Video off'

        }

      }

    })

    this.buttonGrid?.append(audioButton)

    this.buttonGrid?.append(videoButton)

  }

}

App.tsx

import React from 'react'

import styled from 'styled-components'

import { useAppSelector } from './hooks'

import RoomSelectionDialog from './components/RoomSelectionDialog'

import LoginDialog from './components/LoginDialog'

import ComputerDialog from './components/ComputerDialog'

import WhiteboardDialog from './components/WhiteboardDialog'

import VideoConnectionDialog from './components/VideoConnectionDialog'

import Chat from './components/Chat'

import HelperButtonGroup from './components/HelperButtonGroup'

import MobileVirtualJoystick from './components/MobileVirtualJoystick'

const Backdrop = styled.div`

  position: absolute;

  height: 100%;

  width: 100%;

`

function App() {

  const loggedIn = useAppSelector((state) => state.user.loggedIn)

  const computerDialogOpen = useAppSelector((state) => state.computer.computerDialogOpen)

  const whiteboardDialogOpen = useAppSelector((state) => state.whiteboard.whiteboardDialogOpen)

  const videoConnected = useAppSelector((state) => state.user.videoConnected)

  const roomJoined = useAppSelector((state) => state.room.roomJoined)

  let ui: JSX.Element

  if (loggedIn) {

    if (computerDialogOpen) {

      /\* Render ComputerDialog if user is using a computer. \*/

      ui = <ComputerDialog />

    } else if (whiteboardDialogOpen) {

      /\* Render WhiteboardDialog if user is using a whiteboard. \*/

      ui = <WhiteboardDialog />

    } else {

      ui = (

        /\* Render Chat or VideoConnectionDialog if no dialogs are opened. \*/

        <>

          <Chat />

          {/\* Render VideoConnectionDialog if user is not connected to a webcam. \*/}

          {!videoConnected && <VideoConnectionDialog />}

          <MobileVirtualJoystick />

        </>

      )

    }

  } else if (roomJoined) {

    /\* Render LoginDialog if not logged in but selected a room. \*/

    ui = <LoginDialog />

  } else {

    /\* Render RoomSelectionDialog if yet selected a room. \*/

    ui = <RoomSelectionDialog />

  }

  return (

    <Backdrop>

      {ui}

      {/\* Render HelperButtonGroup if no dialogs are opened. \*/}

      {!computerDialogOpen && !whiteboardDialogOpen && <HelperButtonGroup />}

    </Backdrop>

  )

}

export default App

hooks.ts

import { TypedUseSelectorHook, useDispatch, useSelector } from 'react-redux'

import type { RootState, AppDispatch } from './stores'

// Use throughout your app instead of plain `useDispatch` and `useSelector`

export const useAppDispatch = () => useDispatch<AppDispatch>()

export const useAppSelector: TypedUseSelectorHook<RootState> = useSelector

index.tsx

import 'regenerator-runtime/runtime'

import React from 'react'

import { createRoot } from 'react-dom/client'

import { Provider } from 'react-redux'

import { ThemeProvider } from '@mui/material/styles'

import './index.scss'

import './PhaserGame'

import muiTheme from './MuiTheme'

import App from './App'

import store from './stores'

const container = document.getElementById('root')

const root = createRoot(container!)

root.render(

  <React.StrictMode>

    <Provider store={store}>

      <ThemeProvider theme={muiTheme}>

        <App />

      </ThemeProvider>

    </Provider>

  </React.StrictMode>

)

Muitheme.ts

import { createTheme } from '@mui/material/styles'

const muiTheme = createTheme({

  palette: {

    mode: 'dark',

    primary: {

      main: '#426dea',

    },

    secondary: {

      main: '#42eacb',

    },

  },

})

export default muiTheme

phaserGame.ts

import Phaser from 'phaser'

import Game from './scenes/Game'

import Background from './scenes/Background'

import Bootstrap from './scenes/Bootstrap'

const config: Phaser.Types.Core.GameConfig = {

  type: Phaser.AUTO,

  parent: 'phaser-container',

  backgroundColor: '#93cbee',

  pixelArt: true, // Prevent pixel art from becoming blurred when scaled.

  scale: {

    mode: Phaser.Scale.ScaleModes.RESIZE,

    width: window.innerWidth,

    height: window.innerHeight,

  },

  physics: {

    default: 'arcade',

    arcade: {

      gravity: { y: 0 },

      debug: false,

    },

  },

  autoFocus: true,

  scene: [Bootstrap, Background, Game],

}

const phaserGame = new Phaser.Game(config)

;(window as any).game = phaserGame

export default phaserGame

util.ts

export function sanitizeId(id: string) {

  let sanitized = id

  if (sanitized.length > 9 && sanitized.endsWith('-ss')) {

    sanitized = sanitized.substring(0, sanitized.length - 3)

  }

  return sanitized.replace(/[^0-9a-z]/gi, 'G')

}

const colorArr = [

  '#7bf1a8',

  '#ff7e50',

  '#9acd32',

  '#daa520',

  '#ff69b4',

  '#c085f6',

  '#1e90ff',

  '#5f9da0',

]

// determine name color by first character charCode

export function getColorByString(string: string) {

  return colorArr[Math.floor(string.charCodeAt(0) % colorArr.length)]

}

export function getAvatarString(name: string) {

  const part = name.split(' ')

  return part.length < 2 ? part[0][0] : part[0][0] + part[1][0]

}

Package.json

{

  "name": "skyoffice",

  "private": true,

  "description": "Client for SkyOffice",

  "version": "1.0.0",

  "scripts": {

    "dev": "vite",

    "build": "tsc && vite build",

    "preview": "vite preview"

  },

  "dependencies": {

    "@emotion/react": "^11.9.0",

    "@emotion/styled": "^11.8.1",

    "@mui/icons-material": "^5.0.1",

    "@mui/material": "^5.6.4",

    "@mui/styled-engine-sc": "^5.6.1",

    "@reduxjs/toolkit": "^1.6.1",

    "@vitejs/plugin-react": "^2.0.1",

    "colyseus.js": "^0.14.13",

    "emoji-mart": "^3.0.1",

    "peerjs": "^1.3.2",

    "phaser": "^3.55.2",

    "react": "^18.2.0",

    "react-dom": "^18.2.0",

    "react-joystick-component": "^6.0.0",

    "react-redux": "^7.2.5",

    "sass": "^1.42.1",

    "styled-components": "^5.3.5",

    "swiper": "^8.1.4",

    "vite": "^3.0.9"

  },

  "devDependencies": {

    "@types/jest": "^26.0.24",

    "@types/node": "^12.20.27",

    "@types/react": "^18.0.17",

    "@types/react-dom": "^18.0.6",

    "@types/styled-components": "^5.1.25",

    "typescript": "^4.8.2"

  }

}